Algoritmo Adaptativo de Retransmissão Seletiva para Distribuição de Vídeo em Redes Sobrepostas*

Carlos Eduardo Lenz, Lau Cheuk Lung and Frank Siqueira
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina - Florianópolis - Brazil
lenz@inf.ufsc.br, lau.lung@inf.ufsc.br, frank@inf.ufsc.br

ABSTRACT

With the widespread adoption of broadband Internet access, home users have started to consume several multimedia services, such as IP Telephony, radio and video on demand. Meanwhile, the main video services are on-demand and need buffering, while TV is live. They also require a server bandwidth that grows linearly with the number of clients. IP Multicast does not have this bandwidth problem, but is unavailable, so overlay networks are used. Nonetheless, these networks do not specify the packet loss handling: ignore or recover and how. After the analysis on the properties of the H.264 standard for digital video compression, this work presents clear criteria for retransmission based on the several different types of frame coding. It establishes priorities among these types and defines an algorithm for the selection of lost parts for recovery. The algorithm uses a movable target, lowering the target each time the retransmission of a segment is requested. To test it, an overlay network (SeRViSO) is used, comparing the algorithm to another where the target is fixed. The advantages of a clear loss handling are the greater adaptation to the network conditions and less important losses, which do not propagate error for many frames.

RESUMO

Com a popularização da banda larga os usuários domésticos passaram a consumir diversos serviços multimídia pela Internet. Entretanto, os principais serviços de vídeo disponíveis são sob-demanda e precisam de *buferização*, ao contrário da TV, que é ao vivo. Também requerem dos servidores uma largura de banda que cresce linearmente com o número de clientes. O *IP Multicast* não apresenta esse problema de largura de banda, mas não está disponível, restando as redes sobrepostas. Mas essas redes não especificam o tratamento dos pacotes pertidos: ignorá-los ou recuperadas e de que forma. Depois da análise dos recursos do padrão de compressão de vídeo digital H.264, este trabalho apresenta critérios para retransmissão baseados nos diferentes tipos de codificação de quadro. Estabelece prioridades a partir deles e define um algoritmo para a escolha de partes perdidas para a recuperação. O algoritmo usa uma meta variável, que é reduzida a cada vez que a retransmissão de um segmento é solicitada. Para testá-lo usa-se uma rede sobreposta (SeRViSO), comparando o algoritmo a outro de meta fixa. As vantagens de um tratamento claro das perdas são maior adaptação às condições da rede e perdas menos importantes, ou seja, que não redundem em propagação de erro por muitos quadros.

*Selective Retransmission Adaptive Algorithm for Video Streaming in Overlay Networks

Categories and Subject Descriptors

C.2.2 [**Computer Communication networks**]: Network Protocols — *Applications, Routing protocols*; C.2.4 [**Computer Communication networks**]: Distributed Systems — *Distributed applications*

General Terms

Video Streaming, Distributed Multimedia

Keywords

peer-to-peer, application level multicast, data-driven overlay networks, live video transmission

1. INTRODUÇÃO

A oferta de banda larga a usuários residenciais tem aumentado, passando a atender aos requisitos da transmissão de multimídia em tempo real. Preços mais acessíveis e maior velocidade também ajudam a popularizar este serviço. Entre as novas tecnologias oferecidas estão as sem-fio, como o 3G, que também oferecem mobilidade em amplas áreas de cobertura.

Telefonia IP e o rádio pela Internet já estão disponíveis, visto que seus requisitos em termos de largura de banda já são atendidos a contento. Já a programação multimídia oferecida não é ao-vivo como a TV. Os serviços de vídeo na internet[1] consistem de vídeos disponíveis sob-demanda, que devido à transmissão confiável dependem de *buferização*.

Em geral, as arquiteturas tradicionais para distribuir conteúdo enfrentam problemas de escabalibilidade porque a largura de banda necessária nos servidores cresce linearmente

[1] `http://youtube.com` e `http:/vimeo.com`

com o número de clientes. Embora o IP *Multicast* pudesse ser usado para isso, os provedores de serviço o mantém desabilitado [2]. Resta a criação de protocolos de *multicast* na camada de aplicação, em arquiteturas em que os clientes assumem parte da responsabilidade de distribuir o conteúdo, semelhantes às redes *peer-to-peer*.

As primeiras abordagens para *streaming* de vídeo distribuído são muito semelhantes às técnicas de *multicast*. Uma árvore é criada contendo todos os nós clientes, e o servidor na raiz. O vídeo flui em mensagens periódicas "empurradas" (*PUSH*) da raiz às folhas [7]. Abordagens seguintes passaram a usar redes sobrepostas com transmissão dos dados por *PULL* ("puxar"), deixando que os nós se associem livremente e dados fluam entre os nós da forma mais conveniente. Os nós estabelecem parcerias entre si para a transmissão dos dados, que devem ser solicitados explicitamente de parceiros que os possuem [12]. Técnicas seguintes também ofereceram *PUSH* em redes sobrepostas [8, 11]. A questão do controle de erros foi, entretanto, ignorada.

O objetivo deste trabalho é propor um mecanismo de distribuição de vídeo digital baseado nas técnicas anteriores. Denominado SeRViSO, ele realiza a retransmissão dos pacotes perdidos com base nos critérios definidos. Um novo algoritmo é proposto para selecionar pacotes para retransmissão, baseado nos tipos de codificação de quadros mais importates para a qualidade de reprodução. Um protótipo da proposta foi então implementado para comparar o algoritmo de retransmissão e outro com uma recuperação total que ignora o conteúdo do vídeo.

O trabalho se baseia nas características do mais recente padrão de vídeo digital – o H.264 [10] – a fim de definir critérios que auxiliem na criação de um mecanismo inteligente de recuperação de perdas. Neste contexto, inteligente significa que as perdas que impactam mais severamente a qualidade do vídeo têm prioridade na recuperação. Adicionalmente, este trabalho realiza uma análise das perdas e formas de recuperação, problemas que não vinham sendo tratadas em outros trabalhos [7, 12, 8, 11].

A rede sobreposta apresentada é orientada a dados pura – isto é, não-híbrida, pois segue a abordagem de *PULL* apenas [12] – deste modo o trabalho foca principalmente nos critérios de retransmissão. Entretanto, a organização da rede é ortogonal à forma de transmissão, de forma que protocolo pode ser complementado com mecanismos *PUSH*. Os testes com o algoritmo de restransmissão seletiva proposto mostraram que ele aceita perdas mais altas em relação a um modo de recuperação total. Isso é benéfico por dois motivos: menor tráfego com retransmissões e perdas 'inteligentes', isto é, que afetem o mínimo possível a qualidade da apresentação.

2. VÍDEO DIGITAL

A codificação de vídeo digital busca encontrar as repetições que ocorrem naturalmente dentro das imagens e em sequências de imagens. Assim, as repetições são codificadas apenas uma vez cada, mas usadas na decodificação várias vezes, denotando a taxa de compressão. Mas para evitar que os erros causados pelas perdas se acumulem, usam-se pontos de sincronização (figuras auto-contidas). Eles garantem que as figuras anteriores não sejam usadas na compressão das posteriores. Há uma relação de dependência entre as figuras de um grupo, especiamente com a figura de sincronização (a primeira). Um grupo de figuras inclui quadros codificados com diferentes técnicas de predição (figura 1) para atingir

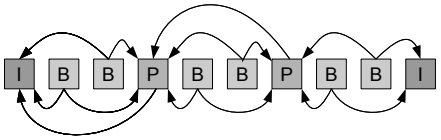

Figura 1: Grupo de Figuras.

maior compressão. Cada figura pode ser dividida em diferentes fatias para facilitar o seu processamento e transmissão. Existem três técnicas de predição:

***I*ntra** esse tipo de quadro é autocontido. Permite apenas predição espacial dentro da mesma fatia e realiza a sincronização, mas atinge a menor compressão. Sua perda afeta a reprodução do grupo inteiro;

***P*reditiva** predito com auxílio de quadro anterior **I** ou **P**. Oferece compressão média e tem custo (em termos de uso de *buffers* e complexidade computacional) inferior aos quadros **B**. Em caso de perda, afeta a reprodução dos quadros **B** vizinhos, do **P** seguinte (diretamente) e possivelmente dos subsequentes (indiretamente);

***B*idirecional** predito com auxílio de um quadro anterior e outro posterior (ambos **I** ou **P**). Permite a maior compressão dos dados e sua perda não afeta a decodificação de outras figuras.

Comparando o padrão H.264 [10] em relação ao MPEG-4 "Parte 2 - Visual" e aos padrões anteriores (MPEG-2), muitas das melhorias consistiram em flexibilizar o uso de procedimentos já existentes, de forma que os programas de codificação pudessem alcançar maiores taxas de compressão. Introduziu também as partições, que permitem dividir os dados de uma fatia. Existem três tipos, apresentados aqui em ordem crescente de tamanho e decrescente de importância: **A** – Contém os cabeçalhos e permite a reprodução parcial; **B** – Contém as texturas de codificação **I**; **C** – Contém as texturas de predição **P** ou **B**.

A camada de abstração de rede veio para permitir a codificação segundo padrões anteriores de transmissão de vídeo ao mesmo tempo que uma nova representação em mais baixo nível é usada. São os *Network Abstraction Layer Units* (NALU), pacotes que facilitam a adaptação ao MTU (*Maximum Trasmission Unit*) da rede. Podem conter [9]: uma fatia, partição, conjunto de parâmetros de figura ou de sequência (são informações pertinentes à decodificação de uma figura ou de uma sequência inteira). Ele possui um cabeçalho simples de um *byte* que indica o tipo dos dados contidos. No padrão também existem variantes das fatias **I** e **P** que permitem usar diferentes figuras como referência. Esses aspectos não são usados neste trabalho, as variantes são tratadas como os tipo originais.

3. TRABALHOS RELACIONADOS

As primeiras abordagens para *streaming* de vídeo distribuído ainda eram muito próximas das técnicas de *multicast*. Uma árvore é criada contendo o servidor na raiz e todos os clientes nos nós internos e nas folhas. O vídeo flui em mensagens periódicas da raiz até as folhas, de modo que os dados são "empurrados" (*PUSH*) por cada nó logo que são recebidos. Visto que árvores de muita profundidade levariam a um atraso excessivo, o ZIGZAG [7] propos, então,

manter os clientes em grupos (*clusters*) e construir a árvore a partir deles, de forma que a profundidade final da árvore seja limitada. Numa aplicação onde os nós entram e saem do grupo a todo instante, muitas vezes sem uma aviso prévio, a árvore de multicast sofre modificações constantes. Para acelerar a sua adaptação e delimitar o fluxo de mensagens de controle existe a Organização Administrativa, à qual todos os clientes devem pertencer. Entretanto, as redes baseadas em árvores de *PUSH* exibem os seguintes problemas:

1. São sensíveis ao abandono ou falha nos nós. Mesmo com as melhorias propostas para amenizar esse problema ainda há um atraso entre uma falha e o funcionamento da árvore ser completamente corrigido.

2. Variações na carga de rede de um nó não passam despercebidas pelos nós descendentes na árvore.

3. As folhas não contribuem, aumentando a carga sobre os nós internos.

As redes *PULL* tem características diferentes das redes *PUSH*, visando eliminar os problemas das anteriores. Não possuem árvores de distribuição, mas as parcerias que estabelecem para a troca dos dados geram uma topologia que foi rotulada como malha (*mesh*) no trabalho [8]. Os dados são requisitados explicitamente entre os parceiros, de modo que a tolerância a faltas nos nós é automática. Para permitir as solicitações, os nós devem enviar relatórios periódicos das partes disponíveis. E por não haverem folhas, o largura de banda dos nós é compartilhada com maior igualdade, ou de acordo com a capacidade de cada nó. Como não é escalável que os nós conheçam todos uns aos outros, um protocolo adicional é usado para informar quais membros fazem parte da rede, de forma que os nós precisem conhecer apenas um subconjunto dos participantes. Os dados são divididos em segmentos com aproximadamente 1 segundo pela taxa de *bits* da mídia, sendo que cada segmento pode ser solicitado de um parceiro diferente. De posse da lista de segmentos disponíveis em cada parceiro, o nó precisa buscar os segmentos que não possui. O DONet [12] propôs um algoritmo de escalonamento que seleciona primeiro os segmentos de menor disponibilidade, e frente a alternativas, privilegia o parceiro com maior largura de banda e tempo disponível. Entre os problemas das redes *PULL* estão: maior atraso (fato intrínseco, ao se comparar *PUSH* e *PULL*) e por isso maior *bufferização*.

Pensando em remediar os problemas das redes *PUSH* e das *PULL*, alguns trabalhos propuseram redes híbridas, que procurem operar por *PUSH* a maior parte do tempo, mas que usem o *PULL* quando necessário. Elas são constituídas de malhas para *PULL*, mas adotam cada uma delas algum mecanismo extra para permitir o *PUSH*. O mTreebone [8] acrescenta uma árvore para realizar o *PUSH* dos dados. Assim, os nós considerados estáveis pertencem à arvore realizando *PUSH*, enquanto os outros ficam na borda (*outskirts*) e só enviam dados em resposta a requisições *PULL*. O GridMedia [11] oferece *PUSH* sem o uso de uma árvore. Logo que entre na rede, o nó só pode operar por *PULL*, tendo que esperar o próximo intervalo para usar o *PUSH*. A cada intervalo, os nós devem se negociar com os vizinhos para receber os pacotes por *PUSH*.

3.1 Técnicas de Retransmissão Seletiva

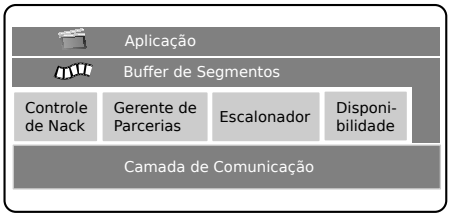

Figura 2: Componentes do Protótipo.

As técnicas a seguir ponderam a retransmissão em função do tipo de dados perdidos. Elas se baseiam no fato que os padrões atuais de vídeo digital aplicam técnicas de codificação temporal. Essas técnicas, entretanto, não foram aplicadas a redes sobrepostas.

O trabalho [1] desenvolve um simples protocolo *multicast* semi-confiável baseado na hierarquia de quadros do MPEG-4, sem considerar a evolução do H.264. Ele também utiliza os receptores vizinhos no grupo para realizar a retransmissão, aliviando a carga do emissor. Tanto o envio inicial dos dados, como as mensagens de **NACK** e as retransmissões são difundidas para todos os nós do grupo via *multicast*.

Em [4] é proposto um controle de retransmissão baseado no protocolo DCCP [5], o qual fornece as informações que a técnica necessita: detecção dos pacotes perdidos e a capacidade atual do canal. Como a relação dos pacotes perdidos e a capacidade do canal são conhecidos pelo emissor no DCCP, a retransmissão é controlada inteiramente nele e de acordo com sua capacidade, sem o envio de NACKs pelo cliente. A partir da detecção de perdas, se verifica se o *throughput* é maior que a taxa de *bits* do vídeo somada às retransmissões.

4. ARQUITETURA PROPOSTA

Para evitar atrasos com retransmissão de pacotes, as redes sobrepostas anteriores adotaram o UDP. Ou seja, não há correção de erros e, portanto, nenhuma garantia de entrega. Apenas alguns trabalhos propuseram mecanismos de recuperação seletiva, mas nenhum propôs um sistema visando escalabilidade ao nível da Internet. Desta forma, resta considerar as seguintes opções naturais para agregar tratamento de perdas às redes sobrepostas. **Ignorar as perdas**: isso equivale a negar a existência ou importância das perdas. Na prática, entretanto, a confiabilidade da Internet não pode ser garantida. **Retransmitir cada perda**: do ponto de vista da qualidade percebida pelo usuário, a recepção a tempo é mais importante que a confiabilidade completa, justamente o que esta opção contraria. **Escolher algumas partes para a recuperação**: é a linha proposta neste trabalho.

Visto que podem existir inúmeras formas de priorizar retransmissões, este trabalho propõe uma alternativa baseada em características do H.264. À técnica proposta foi dado o nome de "*Selective Retransmission Video Streaming Overlay*" (SeRViSO). Uma visão geral com os principais componentes que compõe a arquitetura pode ver vista na figura 2. A função de cada um deles é descrita em seguida.

Aplicação: componente de integração com um tocador de mídia externo. Um novo segmento deve ser enviado para exibição a cada segundo. Após o primeiro segmento ser recebido, o componente espera alguns segundos (quantidade fixa) para permitir que um *buffer* seja formado, e então inicia a executar. ***Buffer* de Segmentos**: este componente armazena os segmentos que já foram recebidos, total ou par-

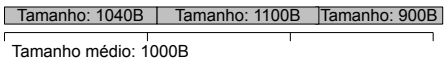

Figura 3: Os segmentos têm tamanho variável.

cialmente. É indexado pelo número do segmento e, dentro deste, pelo endereço do elemento. **Controle de NACK**: componente que detecta as perdas, avalia a necessidade de retransmissão de acordo com o algoritmo e envia mensagens de NACK. **Disponibilidade**: há uma troca de informações sobre a disponibilidade de segmentos entre os parceiros. Este componente envia essas informações frequentemente. **Escalonador**: para receber os dados os parceiros são acionados, por isso esse componente escolhe entre os que dispõem de cada segmento necessário. **Gerente de Parcerias**: controla para que o número de parcerias (e de nós conhecidos) esteja dentro dos limites desejados, solicitando e respondendo solicitações quando necessário. **Camada de Comunicação**: cuida do envio e recebimento de mensagens e dados. São suas responsabilidades: armazenar as informações de disponibilidade recebidas dos parceiros para uso pelo escalonador; preparar as mensagens contendo os segmentos solicitados nas requisições e NACKs recebidos, colocando-os na fila para entrega; armazenar no *Buffer* os dados e metadados recebidos; e encapsular as mensagens de dados no protocolo TFRC [3] de controle de congestionamento. Por isso, a mensagem pode demorar a ser enviada se o *link* entre os nós estiver ruim. Já as mensagens de controle (requisições, NACKs, etc) são enviadas diretamente.

Se a segmentação ocorrer igual às técnicas anteriores, fatalmente os **elementos** (este trabalho se refere a pedaço da mídia que pode ser um NALU, um pacote de áudio ou outro desconhecido) serão quebradas ao meio, o que aumenta a perda no caso de um segmento perdido. Por isso, quando um segmento termina no meio de uma unidade, ele é estendido até o fim da mesma (figura 3).

4.1 Construção e Dinâmica da Rede Sobreposta

Esta sessão descreve como os componentes listados na sessão anterior operam a entrada e saída dos nós da rede e o estabelecimento de parcerias, o que constitui a dinâmica do grafo de associações da rede. Para entrar na rede cada nó envia uma mensagem ***ENTER*** a um dos Rendezvous, que trabalham de forma replicada, trocando mensagens de sincronização. Os Rendezvous são nós que apenas ajudam na construção e manutenção da rede, mantendo uma lista de nós ativos. A mensagem ***ENTER*** precisa ser repetida, como um marca-passo, para demonstrar que o nó continua ativo. Numa saída controlada, o próprio nó envia uma mensagem ***LEAVE*** ao rendezvous, e também a cada parceiro. Entretanto, após um *timeout* sem receber mensagens de um nó, seu parceiro detecta a sua falha. Neste caso, ele envia uma mensagem ***LEAVE*** no nome do ex-parceiro ao rendezvous.

Cada nó precisa conhecer apenas um subconjunto entre todos os nós presentes na rede sobreposta. Para isso, ele deve manter uma lista com os nós por ele conhecidos. Esta lista é limitada em tamanho, de forma que os nós há mais tempo sem comunicação podem ser removidos. Nós podem ser acrescentados nesta lista quando uma mensagem de qualquer tipo é recebida do nó ou através de uma lista de nós presente em uma mensagem ***NODES*** recebida em uma das seguintes condições:

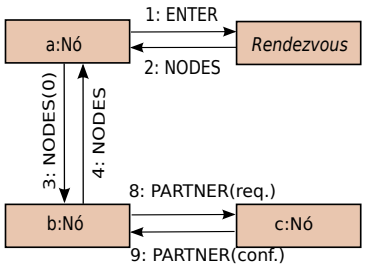

Figura 4: Troca de mensagens para descoberta de nós e estabelecimento de parcerias.

1. Cada mensagem ***ENTER*** enviada por um nó ao Rendezvous é respondida com uma pequena parte dos nós ativos de sua lista;

2. Se a lista de nós conhecidos estiver muito pequena, o nó envia uma mensagem ***NODES*** vazia a algum nó de sua lista. Essa mensagem é interpretada como requisição por alguns do nós. O nó que a recebe responde com um conjunto de nós como se fosse o Rendezvous.

Nem todos os nós conhecidos são usados para transferência de dados, apenas alguns com os quais são estabelecidas parcerias. Desta forma, cada nó busca estabelecer uma lista de nós parceiros, subconjunto da lista de nós conhecidos, com limites superior e inferior de tamanho. Enquanto o limite inferior não for alcançado, o nó busca entre os nós conhecidos os que aceitem formar parceria. Mensagens ***PARTNER*** são usadas para realizar a requisição e confirmação de parcerias, diferenciadas por um *bit* de marcação. Enquanto a lista de nós ativos de um nó não atingir seu limite superior, este vai confirmar todas as parcerias solicitadas. Quando um nó já não pode mais aceitar novas parcerias ele para de responder às mensagens ***PARTNER***, de forma que o nó requisitante, ao final do *timeout* da espera por uma resposta, procure parcerias com outro nó. O uso das mensagens ***ENTER***, ***NODES*** e ***PARTNER*** está ilustrado na figura 4.1.

Toda vez que recebe uma mensagem ***ENTER***, o Rendezvous acrescenta o nó na lista de nós ativos, se ele ainda não estiver. Um nó sai da lista quando uma mensagem ***LEAVE*** for recebida ou depois que um *timeout* sem que o nó tenha enviado nenhuma mensagem ao rendezvous para demonstrar que continua ativo.

4.2 Transferência dos Dados

A partir do estabelecimento das parcerias, os nós começam a trocar mensagens ***BMAP*** entre si. Seu propósito é informar os segmentos da mídia disponíveis, de forma que possam ser solicitados uns dos outros. Disponibilidade significa que o algoritmo de seleção para NACK parou de solicitar qualquer elemento, o que significa que está satisfeito com as partes do segmento que possui.

A figura 5 define "janelas" que delimitam sequências de segmentos sobre as quais diferentes componentes do SeRViSO atuam. ***Buffer***: sequência ampla de segmentos, usada para delimitar as mensagens ***BMAP***. É definido em torno do ponteiro do tocador, sendo metade da janela antes e a outra metade depois dele. Normalmente são usados intervalos de um ou dois minutos. **Escalonador**: sequência menor que limita sobre quais segmentos o algoritmo escalonador vai atuar. Apenas segmentos à frente do tocador fazem parte

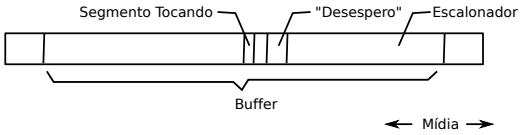

Figura 5: Janelas.

desta janela. **"Desespero"**: uma sequência anterior à janela do escalonador, mas ainda assim alguns segundos à frente do tocador. Usada pelo modo homônimo descrito mais à frente.

Antes de solicitar os segmentos necessários à exibição do vídeo, um nó primeiro deve fazer uma seleção que busque os de maior interesse entre os que tiveram sua disponibilidade avisada pelos parceiros. O algoritmo adotado pelo escalonador do SeRViSO é uma variante do algoritmo descrito em [12], alterado de modo a impor uma janela mais restrita. A partir das listas de segmentos que o algoritmo retorna é enviada uma mensagem ***REQUEST*** a cada um dos parceiros escolhidos.

Quando um nó recebe uma requisição solicitando um ou mais segmentos, ele se prepara para atendê-la se possuir os segmentos solicitados (o que acontece se a requisição está de acordo com a mensagem ***BMAP*** enviada). Os elementos do segmento são divididos em sequências, e cada uma delas é encapsulada em uma mensagem ***DATA***, que é inserida em uma fila de mensagem para entrega. As sequências obedecem os limites dos elementos, sem quebrá-los. Para cada parceiro, um nó deve manter uma fila de mensagens separada. Essas filas podem ter mensagens de **Controle** – pequenas, enviadas diretamente na primeira oportunidade – e **Dados** – maiores, que carregam partes do vídeo.

Em cada mensagem ***DATA*** o nó receptor obtêm uma sequência de segmentos, mas apenas os dados e seus limites (início e fim). Então não há como saber onde estão localizados os elementos ali contidos. Como essas informações serão necessárias no algoritmo de seleção para retransmissão, faz-se necessária uma mensagem adicional. Assim, toda vez que um segmento é requisitado, em meio às mensagens ***DATA*** é enviada uma mensagem ***METADATA***, contendo detalhes (localização e tipo do elemento) de todos os elementos no segmento. Essa mensagem também indica os elementos que o parceiro não possui. Tais informações são importantes para o **Controle de NACK**, visto que de outra forma não há como saber a importância dos elementos perdidos. Por isso, se a mensagem ***METADATA*** não for recebida (devido a uma perda, por exemplo), uma mensagem de sinalização é enviada para que ela seja retransmitida.

Pelo modo de funcionamento do TFRC e devido à equação que ele usa para determinar a taxa de transferência, mensagens pequenas reduzem a velocidade permitida, chegando ao ponto de tornar a obtenção de vídeo em tempo real impossível. As mensagens ***DATA***, fruto da divisão de um segmento, podem ser ajustadas a fim de atingir um tamanho razoável, tendo em vista essa característica do TFRC. Mas mensagens ***DATA*** geradas em resposta a NACKs podem não ter essa opção, devido às sequências requisitadas serem muito pequenas. Neste caso, um grupo de mensagens ***DATA*** pode ser encapsulado em uma mensagem ***MULTI***, de forma que esta nova mensagem seja posta na fila e posteriormente enviada através do TFRC sem diminuir a taxa de transferência. A figura 4.2 demonstra como as mensagens relacionadas à transferência de dados interagem. As mensagens ***NACK***, ***QNACK*** e ***QDATA*** são apresentadas na próxima seção.

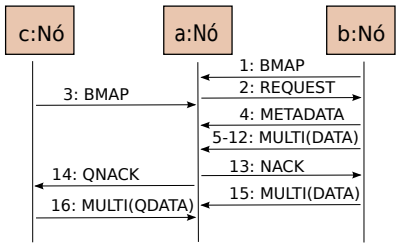

Figura 6: Troca de mensagens para transmissão de dados.

4.3 Retransmissão

De posse dos *timestamps* da recepção dos segmentos, o componente **Controle de NACK** determina para quais segmentos disparar o processo de NACK. Esse disparo ocorre quando o *timestamp* é muito antigo ou um segmento posterior recebido do mesmo parceiro tem *timestamp* mais recente. Nesse momento, o processo de NACK chama o algoritmo de seleção. Para mandar uma mensagem de ***NACK***, primeiro os segmentos escolhidos devem ser separados em sequências contínuas, a fim de reduzir a quantidade de dados transferidos. Mas é importante distinguir as mensagens dirigidas a outro parceiro (escolhido aleatoriamente) que não o que originalmente enviou o resto do segmento, o que acontece quando fornecedor do segmento possui todos os seus elementos. Neste caso, é usada uma mensagem ***QNACK***, que pode ser ignorada total ou em parte caso o parceiro escolhido também não possua algum elemento. Quando isso acontece, posteriormente outro parceiro pode ser escolhido aleatoriamente. As mensagens ***QNACK*** são respondidas por mensagens ***QDATA***, que diferem das mensagens ***DATA*** apenas pelo identificador no cabeçalho.

4.4 Algoritmo de Seleção

O algoritmo deve trabalhar e avaliar cada segmento isoladamente. Antes da sua execução deve ser precedido por uma rotina que varre os elementos do segmento, atribuindo um peso através do algoritmo 1, cujo cálculo leva em conta principalmente o tipo de quadro H.264 que o elemento contém, mas também o seu tamanho.

Algoritmo 1 – Rotinas para definir o peso (prioridade) de cada elemento.

function elementWeight(element) : real
return $MIN(sizeWeight + kindWeight, 3)$
end function

function sizeWeight(element) : real
return $\frac{MAX(10-log10(size(element)),0)}{10}$
end function

function kindWeight(element) : real
when *not NALU* $\implies$ 1.5
when NON_IDR, IDR $\implies$ check slice type (I, P, B ...)
when I, PARTITION_A $\implies$ 3
when P $\implies$ 2
when B, PARTITION_B, PARTITION_C $\implies$ 1
when *delimiter* $\implies$ 0
otherwise $\implies$ 1.5
end function

O algoritmo SeRViSO original [6] parte do somatório dos pesos dos elementos presentes e do somatório dos pesos de todos os elementos (presentes e faltando) do segmento. De posse desses valores, o algoritmo busca obter todos os quadros de peso três, principalmente quadros I, mas também 90% dos elementos considerados ponderadamente e 70% do tamanho total do segmento em *bytes*.

O novo algoritmo apresentado aqui (algoritmo 2) é uma variação adaptativa do anterior. Ou seja, ele busca flexibilizar as metas que eram fixas de acordo com o estado da rede. Isto significa que ele pode buscar metas superiores às do algoritmo original ou tolerar perdas maiores, de acordo com o que a rede permitir.

Entretanto, desenvolver uma heurística adequada para avaliar a rede exige considerações especiais. Enquanto as informações determinadas pelo TFRC poderiam ser usadas, isso introduziria um acoplamento indesejável, impedindo a substituição posterior desse protocolo. A solução encontrada foi baseada na contagem das vezes que o algoritmo de seleção é executado para cada segmento (linha 2). Desta forma, o número de NACKs é usado para determinar as metas ponderadas e de tamanho (linhas 3 e 4). O resto do algoritmo permanece igual ao anterior. Destacam-se: as somas dos pesos de todos elementos do segmento (linha 5) e dos perdidos (linha 8); a escolha automática dos elementos prioritários (linha 10); a ordenação dos elementos perdidos pela prioridade (linha 14); e a iteração enquanto as metas não forem alcançadas (linha 15), adicionados os elementos de maior peso (linha16).

Algoritmo 2 – SeRViSO Adaptativo.

1: **function** selectMissingElements(segment) : list
2: $nrNacks \leftarrow numberOfNacks(segment)$
3: $minRate \leftarrow acceptableRate(nrNacks)$
4: $minBytes \leftarrow minimumBytes(nrNacks)$
5: $sumWeights \leftarrow sum\{\forall\ element \in segment \rightarrow elementWeight(element)\}$
6: $sumBytes \leftarrow sum\{\forall\ element \in segment \mid \neg\ missing(element) \rightarrow size(element)\}$
7: $missing \leftarrow \{\forall\ element \in segment \mid missing(element) \rightarrow element)\}$
8: $sumMiss \leftarrow sum\{\forall\ element \in missing \rightarrow elementWeight(element)\}$
9: $sumAvail \leftarrow sumAll - sumMiss$
10: $always \leftarrow \{\forall\ element \in missing \mid elementWeight(element) = 3 \rightarrow element\}$
11: $missing \leftarrow missing - always$
12: $sumAvail \leftarrow sumAvail\ + sum\{\forall\ element \in always \rightarrow elementWeight(element)\}$
13: $sumBytes \leftarrow sumBytes\ + sum\{\forall\ elementinalways \rightarrow size(element)\}$
14: $sort\_by(missing, elementWeight, 'descending')$
15: **while** $missing \neq \emptyset\ \wedge (sumAvail/sumWeights < minRate\ \vee sumBytes < minBytes)$ **do**
16: $element \leftarrow first(missing)$
17: $selected \leftarrow selected \cup \{element\}$
18: $missing \leftarrow missing - \{element\}$
19: $sumAvail \leftarrow sumAvail + elementWeight(element)$
20: $sumBytes \leftarrow sumBytes + size(element)$
21: **end while**
22: **return** $always + selected$
23: **end function**

Para ser adaptativo, o esforço de retransmissão expresso nas metas deve decrescer à medida que mais NACKs são enviados para o mesmo segmento. Por isso, o algoritmo busca 100% do segmento na meta ponderada para o primeiro NACK, 95% para o seguinte, depois 90% e assim por diante (algoritmo 3). Já na meta de tamanho, os índices são 100%, 90%, 80%, etc. Essas reduções foram definidos para permitir retransmissões mais agressivas nas primeiras duas tentativas, e mais leves a partir da quarta.

Algoritmo 3 – Ajustando a meta ponderada ao número de NACKs.

function acceptableRate(numberOfNacks) : real
return $1 - 0.05 \times numberOfNacks$
end function

function mininumBytes(numberOfNacks) : real
return $(1 - 0.1 \times numberOfNacks) \times SEGMENT\_SIZE$
end function

4.5 Modo "Desespero"

Nos intervalos das janelas do *Buffer* e do Escalonador pode-se observar que existe um momento a partir do qual um segmento não será mais escalonado, mesmo que ele esteja à frente do tocador. Este cenário, um segmento apenas ser "descoberto" (isto é, disponibilizado por um parceiro) muito próximo de seu momento de exibição (portanto fora da janela do Escalonador), pode acontecer algumas vezes numa rede sobreposta. Se ocorresse o escalonamento, o parceiro selecionado teria que mandar todas as partes do segmento em sequência, onde possivelmente nem todos chegariam a tempo, com ou sem NACK. Entretanto, um tratamento diferenciado permite que o segmento seja obtido em paralelo, o que reduz o tempo necessário para tanto, possibilitando que todas as mensagens cheguem a tempo. Quando isso acontece é aplicado o modo "desespero". Ele divide o segmento em intervalos de tamanho fixo, exceto possivelmente o último. Então cada parte é solicitada de um parceiro diferente que possui o segmento, usando mensagens ***QNACK***. Ele também solicita os metadados a um dos parceiros, de modo que possa usar o **Controle de NACK** posteriormente, ao invés de dividir o segmento em intervalos de tamanho fixo.

4.6 Otimizações e Restrições

Como o servidor avisa a disponibilidade dos segmentos antes que qualquer outro nó o faça, é o primeiro alvo para requisições. Para evitar que ele transmita tudo a cada parceiro, ele restringe a disponibilidade de cada segmento para apenas 2 dos seus parceiros. Também foi implementado um modo de 'recuperação total' no protótipo do SeRViSO. Ele tem funcionamento mais simples e emprega uma série de otimizações que o tornam semelhante à redes sobrepostas anteriores [12], apenas com o acréscimo do mecanismo de recuperação. Elas são necessárias para evitar carregá-lo com características específicas do algoritmo seletivo, que tem custo mas seriam irrelevantes neste modo. Foram adotadas as seguintes otimizações: a mídia não é processada à procura de NALUs, visto que esta informação não seria usada; cada segmento é simplesmente dividido em oito partes de tamanho fixo para transmissão, permitindo a simplificação de algumas mensagens; não há necessidades de mensagens

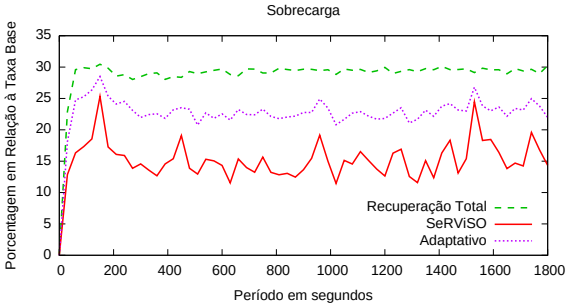

Figura 7: Sobrecarga das Retransmissões.

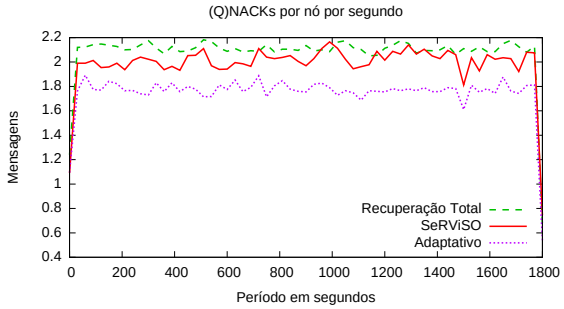

Figura 8: NACKs.

Modo	Retrans. (%)	NACKs (%)	Perdas / seg.	Perdas: I / Total (%)
Rec. Total	29.27	2.08	3.8	110.3
SeRViSO	15.28	1.98	16.1	57.1
Adaptativo	23.03	1.75	10.0	65.3

Tabela 1: Médias das sessões.

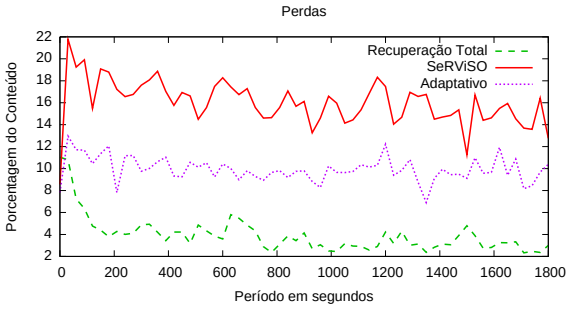

Figura 9: Perda.

METADATA, nem de ***MULTI***; e o algoritmo de NACK é mais simples: tenta recuperar todas as partes perdidas. Este modo foi usado como base de comparação nos experimentos, visto não ter sido possível obter com os autores a implementação de redes sobrepostas anteriores [12, 8].

5. EXPERIMENTO

Foi construído um protótipo na linguagem *Python* que implementa o algoritmo proposto e toda a rede sobreposta SeRViSO. Como ambiente de teste foi adotado o PlanetLab[2]. Em média 142 nós completaram cada teste, visto que nós com problemas de conectividade abandovam o teste. Para avaliar a retransmissão foram introduzidos erros aleatórios a uma taxa de 20%, sem contar os erros que ocorreram naturalmente na rede. Níveis maiores de perdas não foram usados porque o TFRC usa o cálculo das perdas (entre outros fatores) para determinar a taxa máxima de transferência. Desta forma, perdas maiores levam a uma taxa insuficiente para a transmissão de vídeo em tempo real. Outros protocolos de controle de congestionamento mais agressivos, entretanto, podem oferecer melhores resultados que o TFRC. Os dados foram registrados nos nós durante a execução de cada teste e transferidos após seu final para a extração das medições apresentadas a seguir. Cada algoritmo (Recuperação Total, SeRViSO e Adaptativo) foi repetido quatro vezes e a média das execuções foi usada. Uma limitação do protótipo atual é trabalhar apenas com arquivos MPEG-4 prégravados com conteúdo H.264, sem suportar *streams*. Neste caso, por causa do tocador o cabeçalho do arquivo deveria ser transferido confiavelmente. Entretanto, ele foi ignorado nos testes porque as máquinas não têm interface e também para não influenciar nos resultados ao aumentar as retransmissões. O arquivo usado nos testes possuía taxa de *bits* de 249 kb/s e duração de 30 minutos. Todos os nós observaram um limite de emissão durante o experimento, de 2Mbps. Cada nó inicia a exibição 10 segundos após a disponibilidade do primeiro segmento, a fim de obter um pequeno *buffer*.

Comparando a quantidade de dados retransmitidos com a taxa base pode-se verificar qual o nível de esforço que cada algoritmo emprega para obter os pacotes perdidos (figura 7). Enquanto o modo Recuperação Total retransmite sempre que pode, os algoritmos SeRViSO realizam esforço menor, em variados graus. A primeira coluna da tabela 1 resume isso com a média das retransmissões para cada teste. O Adaptativo realiza mais retransmissões que o SeRViSO no nível de perdas testado, mas em cenários com mais perdas isso poderia se inverter, visto que a meta não é fixa. Os algoritmos SeRViSO geraram 14 e 6% de economia em relação ao de Recuperação Total. Os picos mais acentuados dos algoritmos SeRViSO nos gráficos são atribuídos à perdas concentradas (aleatórias) de fatias I, que sempre são escolhidas para retransmissão. Em contrate, quando as perdas recaem em maior número sobre fatias P ou B, os algoritmos são mais tolerantes, gerando mínimos locais nos gráficos. Já o modo Recuperação Total desconhece o tipo de dados perdidos e recupera indistintamente, apresentando valores mais estáveis.

A quantidade de NACKs enviados por segundo (figura 8) é um reflexo da forma como cada algoritmo avalia a retransmissão, visto que todos seguem o mesmo processo para controlar a execução do processo de seleção de elementos e envio dessas mensagens. O modo Recuperação Total envia NACKs (considerando cada segmento) enquanto houver tempo, enquanto os outros param antes, de acordo com suas metas. A segunda coluna da tabela 1 demonstra isso através da média de NACK enviados por segundo em cada teste. O Adaptativo para de enviar NACKs antes dos outros, mesmo realizando mais retransmissões. Como ele se adapta às condições da rede, na primeira vez que ele é executado para um determinado segmento, escolhe todas as perdas para retransmissão, sendo que a meta é diminuída a cada execução. Desta forma, inicialmente a pedida de retransmissão é maior, mas como ela cai o algoritmo para de enviar NACKs mais cedo.

As perdas medidas mostram a tolerância dos algoritmos na seguinte ordem: *Recuperação Total* < *Adaptativo* < *SeR-*

[2] `http://www.planet-lab.org/`

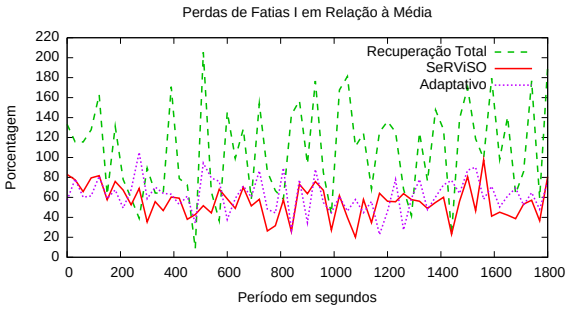

Figura 10: Perda de Quadros Importantes.

ViSO (figura 9). Comparando a taxa geral de perda com a medida para cada tipo de fatia pode-se determinar a 'seletividade' dos algoritmos, ou seja, o quanto eles se focaram nos quadros mais importantes (figura 10). As colunas três e quatro da tabela 1 trazem as médias dessas duas medidas de perda para cada cenário de teste. O algoritmo SeRViSO básico parece ser mais 'focado' em fatias I. Entretanto, deve-se levar em conta que ele apenas está tolerando mais perdas que Adaptativo no cenário testado. Por outro lado, o Adaptativo teria seletividade ainda mais intensa em cenários com maiores perdas. Se outro protocolo para controle de congestionamente mais agressivo que o TFRC for usado, esses cenários poderam ser verificados.

As mensagens de controle não aumentaram significativamente o tráfego, alcançando 3.74, 5.23 e 5.24% em relação à taxa base (das mensagens de dados) nos cenários Recuperação Total, SeRViSO e Adaptativo. Foi verificado o uso de mensagens ***QDATA*** em lugar das ***DATA***, mas foram encontrados valores muito baixos (em média: 0.0, 0.31 e 0.32%), indicando que essa mensagem é necessária em eventos isolados e relativamente raros.

O recebimento de pacotes após seu tempo de exibição é duplamente nocivo. Além de já estarem perdidos para o usuário, sua transmissão ocupou um canal de comunicação já saturado. Assim eles podem acabar impossibilitando o recebimento de outros pacotes posteriores, que poderiam chegar a tempo. Apenas o modo Recuperação Total exibe esse problema, mas muito pouco para ser relevante (0.13% em média). Tal eficiência deve ser atribuída às janelas do Escalonador e do modo "Desespero".

6. CONCLUSÃO

Em virtude de sua versatilidade, as redes sobrepostas oferecem uma interessante alternativa ao *IP Multicast*, sejam as CDNs, redes P2P ou as que implementam *multicast* na camada de aplicação. Critérios para retransmissão já haviam sido aplicados em protocolos de *multicast* semanticamente confiáveis e na retransmissão seletiva de *streaming* cliente-servidor. Agora este trabalho oferece uma nova proposta voltada a redes sobrepostas, baseada nas características do vídeo codificado segundo o padrão H.264.

Foi proposto um novo algoritmo para controle de erro por retransmissão seletiva dos pacotes perdidos. Ele define prioridades baseadas no tipo de codificação: I tem prioridade maior que P, cuja prioridade é maior que B. O algoritmo anterior (SeRViSO) escolhe todas as partes do tipo I, e também entre as outras de forma que obtenha 90% da soma das prioridades de todos as partes do segmento. As prioridades são definidas por: tipo de codificação e tamanho. O novo algoritmo (Adaptativo) desenvolve esse mecanismo, adotando uma meta que se adapte ao estado da rede em vez de fixá-la em 90%. Desse modo, na primeira vez que um segmento tem partes escolhidas para retransmissão, todo ele é requisitado. Já nas vezes seguintes a meta é reduzida a cada execução.

Pelos resultados dos testes, o algoritmo atinge o objetivo de fornecer uma recuperação altamente seletiva. Especificamente para ambientes com taxa de erro imprevisível, ou pequeno, ou grande, o algoritmo Adaptativo oferece benefícios, mas quando a qualidade desejada é fixa o algoritmo anterior (SeRViSO) é uma boa opção, sempre com a possibilidade de adaptar as metas à qualidade desejada. Os algoritmos SeRViSO e Adaptativo economizam respectivamente, num cenário com 20% de perdas introduzidas artificialmente, 13.45 e 5.44% do tráfego base. As perdas foram medidas em 16.1 e 10% do vídeo. As perdas dos dados mais importantes (fatias com codificação I) foram até 35% inferiores às perdas médias, enquanto no modo de recuperação total elas foram 10% superiores à média.

7. REFERENCES

[1] C. Bortoleto, L. Lung, F. Siqueira, A. Bessani, and J. da Silva Fraga. A semi-reliable multicast protocol for distributed multimedia applications in large scale networks. In *Proceedings of the 8th International Conference on Management of Multimedia Networks and Services*, 2005.

[2] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. *Network, IEEE*, 2000.

[3] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control Protocol (RFC 3448). *Internet Engineering Task Force*, 2003.

[4] Á. Huszák and S. Imre. Content-Aware Selective Retransmission Scheme in Heavy Loaded Wireless Networks. *Wireless and Mobile Networking*, 2008.

[5] E. Kohler, M. Handley, S. Floyd, and J. Padhye. Datagram congestion control protocol (DCCP – RFC 4340). *Internet Engineering Task Force*, 2006.

[6] C. E. Lenz, L. C. Lung, and F. A. Siqueira. Serviso: A selective retransmission scheme for video streaming in overlay networks. *ACM Symposium on Applied Computing - SAC*, 2010.

[7] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM*, 2003.

[8] F. Wang, Y. Xiong, and J. Liu. mTreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In *27th International Conference on Distributed Computing Systems*, 2007.

[9] S. Wenger. H. 264/avc over ip. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003.

[10] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, 2003.

[11] M. Zhang, J. Luo, L. Zhao, and S. Yang. A peer-to-peer network for live media streaming using a push-pull approach. In *Proceedings of the 13th ACM international conference on Multimedia*, 2005.

[12] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming / DONet: A data-driven overlay network for efficient live media streaming. In *IEEE Infocom*, 2005.