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

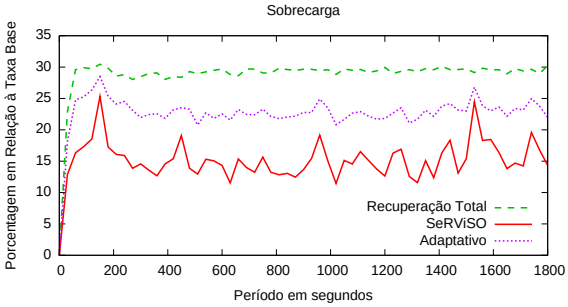

Figura 7: Sobrecarga das Retransmissões.

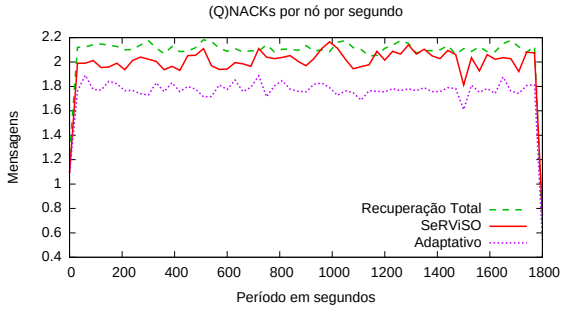

Figura 8: NACKs.

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.

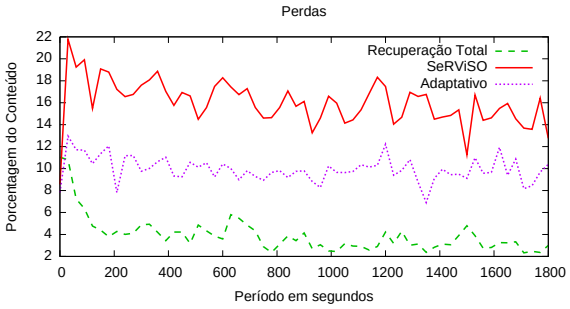

Figura 9: Perda.

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/`

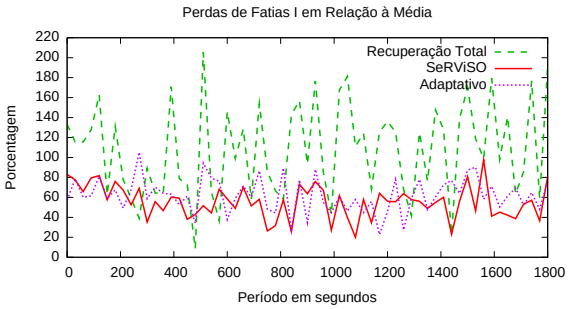

Figura 10: Perda de Quadros Importantes.

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.