Comunicação Multicast Confiável na Implementação de uma Ferramenta Whiteboard

Carlos E. Fisch de Brito * Raphael S. de Moraes
Denise J. de Oliveira †
Edmundo de Souza e Silva ‡

Universidade Federal do Rio de Janeiro, NCE, DCC e COPPE/Sistemas
Cx. P. 2324, Rio de Janeiro, RJ - 20001-970 - Brasil

Resumo

Com o advento das redes de alta velocidade surgiram várias aplicações com os mais variados requisitos de qualidade de serviço. Uma ferramenta whiteboard, usada em teleconferência e trabalho cooperativo é uma dessas ferramentas, com requisitos particulares de QoS. Os requisitos deste aplicativo fazem com que seja necessário o desenvolvimento de algoritmos eficientes para recuperação de pacotes em um ambiente *Multicast* não confiável e ainda garantir a consistência das informações entre os usuários. Este trabalho descreve o desenvolvimento da ferramenta Tangram Whiteboard (TGWB) que possui facilidades não encontradas em outras ferramentas disponíveis atualmente. Descrevemos as soluções adotadas para garantir a consistência das informações em um ambiente multicast não confiável. Também é apresentado um estudo analítico da sensibilidade de medidas de desempenho à variação de parâmetros do algoritmo de recuperação de pacotes.

Abstract

With the advent of high speed computer networks many applications with different quality of service requirements have been developed. A Whiteboard tool, used in teleconferences and in cooperative work is one of these applications with specific QoS requirements. For applications with such characteristics it is necessary the development of efficient algorithms capable of recovering lost packets in an unreliable Multicast environment and also ensuring data consistency between users. This article describes the Tangram Whiteboard (TGWB) developed at UFRJ which has many features not available in other similar tools. We describe the solutions adopted to ensure data consistency in an unreliable Multicast network. An analytical model was constructed to study the sensibility of performance measures with respect to several parameters of the packet recovery algorithm.

*C.E.F. Brito possui bolsa de mestrado da CAPES.
†D.J. Oliveira possui bolsa de iniciação científica do CNPq.
‡Este trabalho conta com o apoio do CNPq/ProTeM e PRONEX.

1 Introdução

A evolução e expansão das redes de computadores, abriram caminho para o desenvolvimento de ferramentas de auxílio a trabalho cooperativo entre pessoas interligadas por uma rede. A finalidade de uma ferramenta Whiteboard é permitir que um grupo de pessoas, em locais distintos, compartilhe uma mesma área de trabalho gráfica. As possibilidades de uso para uma ferramenta como esta são bastante amplas, e vão desde o suporte a teleconferências e ensino a distância, até ao desenvolvimento de sistemas à distância, em equipe.

Um do pontos cruciais para o desenvolvimento de uma ferramenta Whiteboard é a garantia da consistência entre as informações de cada um dos usuários. Isto é, a imagem na tela de um usuário deve ser a mesma daquela na tela de qualquer outro usuário do mesmo grupo. Existem basicamente dois problemas que podem levar a inconsistências entre as telas dos usuários: 1) a perda de pacotes na rede; 2) a execução de comandos gráficos em ordens diferentes nas estações de trabalho do grupo. O primeiro problema é bastante óbvio, pois é impossível manter a consistência entre usuários que não recebem as mesmas informações. Desta forma, o Whiteboard precisa utilizar um mecanismo de comunicação Multicast confiável, isto é, que possa recuperar qualquer pacote eventualmente perdido. O segundo problema é um pouco mais sutil, e envolve o conceito de ordem entre eventos distribuídos. A princípio, não existe uma ordem única e correta para a execução dos comandos gerados pelos diversos usuários de uma sessão. Entretanto, caso usuários diferentes executem o mesmo conjunto de comandos em ordens diferentes, é possível que eles cheguem a resultados distintos. A Figura 1 mostra um exemplo de inconsistência causado pela execução dos comandos em ordens diferentes. Neste exemplo, o usuário A cria os objetos na seguinte ordem: retângulo, circunferência e caixa de texto. Devido a atrasos na rede, o usuário B pode receber e executar os comandos na ordem: circunferência, caixa de texto e retângulo. Supondo que os objetos são opacos, e objetos criados mais recentemente aparecem sobre os outros, o resultado para o usuário B é o exibido na figura. Para resolver este problema, é necessária a implementação de algoritmos que assegurem uma ordem de execução única para todos os usuários.

Existem na literatura diversos trabalhos em que são descritas implementações de Whiteboards. A maioria desses aplicativos disponibilizam recursos limitados para edição gráfica. Além disso, restringem o comportamento do usuário para simplificar problemas relacionados à ordem de execução dos comandos. Em [9] são apresentados sucintamente os problemas básicos que aparecem no desenvolvimento de um whiteboard, sem contudo apresentar soluções definitivas para os problemas. Um trabalho mais recente, derivado deste e bastante conhecido é [7]. Este trabalho apresenta uma solução para recuperação dos pacotes perdidos na rede. Entretanto, o problema de consistência entre os estados dos usuários é simplificado, não permitindo que um usuário modifique os objetos criados por outro. Desta forma, este aplicativo não aborda o problema de ordenação dos comandos.

O objetivo deste trabalho é descrever as principais soluções adotadas no projeto de um Whiteboard (Tangram Whiteboard - TGWB) que possui versatilidade maior que outros Whiteboards descritos na literatura. Fazemos também uma análise do mecanismo de transmissão Multicast confiável utilizado de forma a determinar a sensibilidade de medidas de performance em relação a parâmetros do mecanismo.

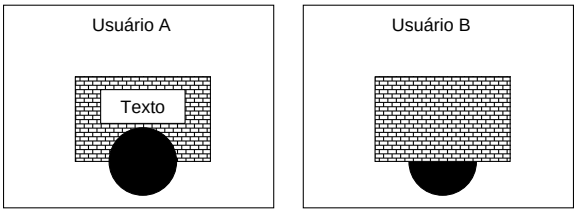

Figura 1: Exemplo de inconsistencia entre usuarios

O aplicativo Whiteboard que projetamos e implementamos, utiliza como interface com o usuário o editor gráfico TGIF (*Tangram Graphic Interface Facility* [4]). O TGIF é uma poderosa ferramenta para edição gráfica baseada em objetos. Neste aplicativo, um desenho é criado a partir de objetos primitivos, tais como linhas, circunferências, polígonos, etc. Os objetos podem possuir texturas, e sofrer diversas transformações como escala e rotação. O TGIF também dá suporte à criação e edição de texto, permitindo a criação de slides e transparências para apresentações. É também sobre este editor gráfico que está sendo implementada a ferramenta TANGRAM-II para modelagem e análise de sistemas [2]. O TGWB está sendo integrado à ferramenta TANGRAM-II que, juntamente com um aplicativo de voz desenvolvido [5] [6] e um servidor de vídeo, fornecerão um conjunto poderoso de ferramentas de teleconferência e ensino à distância.

A seção 2 descreve brevemente a arquitetura da ferramenta e as camados do protocolo desenvolvidas. A seção 3 trata do problema de Multicast confiável, descrevendo a solução adotada. É indicada também a API para a comunicação com a ferramenta gráfica. Na seção 4 apresentamos o modelo adotado para os estudos de confiabilidade Multicast e os resultados obtidos. A seção 5 trata do problema de ordenação de comandos no aplicativo desenvolvido. Finalmente, na seção 6 apresentamos nossas conclusões.

2 O Aplicativo TGWB Desenvolvido

O funcionamento do aplicativo TGWB é baseado em trocas de mensagens. Todo usuário possui uma interface gráfica local, e ao executar um comando, este é codificado e enviado a todos os outros participantes da sessão, através de uma mensagem. Ao receber uma mensagem, o comando é decodificado e executado localmente, como se fosse gerado pelo próprio usuário. Desta maneira, todos os comandos gerados pelos usuários de uma sessão, são executados localmente no aplicativo de cada um.

Estabelecido o funcionamento de cada um dos membros de uma sessão Whiteboard, é necessário definir a estratégia para o gerenciamento da sessão. Neste caso, temos duas opções: uma arquitetura Cliente-Servidor, em que todos os componentes do grupo enviam suas mensagens para o servidor, que pode ser um dos membros do grupo, e este por sua vez re-envia os comandos para os demais participantes da sessão (Figura 2a); ou uma abordagem completamente distribuída em que as mensagens são enviadas diretamente para todos os participantes da sessão (Figura 2b).

A grande vantagem da arquitetura Cliente-Servidor é a sua simplicidade. Ela oferece facilidades para o tratamento de pacotes perdidos e sequenciamento das mensagens, pois é implementada utilizando um conjunto de conexões ponto-a-ponto entre o servidor e cada

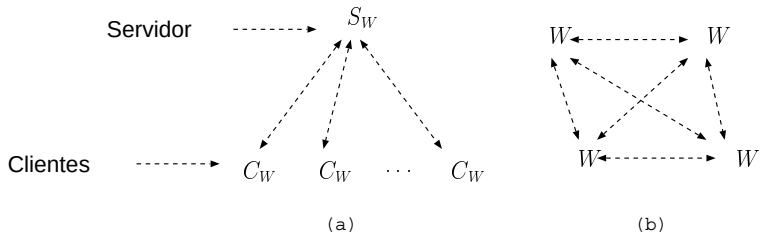

Figura 2: Arquiteturas Cliente-Servidor e Distribuída

um dos clientes. A ordem de execução dos comando é estabelecida pela ordem de chegada das mensagens ao servidor. A arquitetura Cliente-Servidor, no entanto, apresenta a séria desvantagem ao centralizar as mensagens no servidor, e portanto possui as desvantagens de qualquer arquitetura centralizada (ponto único de falha, congestionamento do servidor, aumento do retardo de comunicação entre os membros, etc.).

Uma alternativa à estratégia anterior é a abordagem completamente distribuída. Neste caso são necessários algoritmos mais sofisticados para a recuperação de pacotes perdidos, e definição da ordem de execução de comandos. Por outro lado, ela não apresenta os problemas encontrados na arquitetura Cliente-Servidor, pois todos os membros do grupo exercem o mesmo papel na sessão. Por oferecer uma solução mais elegante, robusta e flexível, foi escolhida a abordagem distribuída para o desenvolvimento do aplicativo TGWB.

Para realizar a troca de mensagens entre os participantes de uma sessão, o Whiteboard utiliza o protocolo IGMP(*Internet Group Management Protocol*). Este protocolo mantém a comunicação entre o grupo Multicast de maneira transparente, isto é, os membros do grupo não precisam conhecer a sua composição e a topologia da rede. Além disso, o protocolo possui facilidades para que os membros possam entrar e sair do grupo a qualquer momento, oferecendo maior flexibilidade para a aplicação.

O protocolo IGMP não garante a confiabilidade da comunicação, nem a ordem de entrega das mensagens. Aplicativos Multicast possuem requisitos de qualidade de serviço bastante diversos. Como o Whiteboard possui requisitos que o protocolo não oferece, devem ser implementados mecanismos para fornecer a qualidade de serviço exigida pelo aplicativo. A figura 3 exibe a arquitetura do aplicativo TGWB desenvolvido, sendo que em negrito estão as camadas implementadas no âmbito deste projeto:

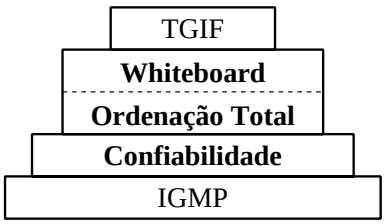

Figura 3: Arquitetura do aplicativo Whiteboard

3 Multicast Confiável: Transmissor *Versus* Receptor

Para a implementação de um protocolo para confiabilidade Multicast, é necessário possuir a capacidade de executar, eficientemente, duas tarefas: 1) detectar a perda de um pacote; 2) recuperar o pacote perdido.

Um dos pontos principais a ser levado em consideração é a definição sobre qual dos agentes da comunicação (o transmissor ou o receptor), será o responsável pela detecção e recuperação dos pacotes perdidos. Em uma comunicação ponto-a-ponto este papel pode ser desempenhado tanto pelo transmissor quanto pelo receptor, funcionando adequadamente em ambos os casos. A comunicação Multicast, no entanto, envolve um transmissor e diversos receptores para cada pacote, portanto apresentando problemas adicionais em comparação à transmissão unicast. Existem basicamente dois métodos para a implementação de uma camada de confiabilidade Multicast [10]: um com recuperação de erro baseada no transmissor; e outro baseada no receptor, sendo que este último possui duas variações.

No método de confiabilidade baseado no transmissor, a responsabilidade pela deteção e recuperação de pacotes perdidos é atribuída ao membro que gerou e enviou o pacote ao grupo. Esta estratégia é caracterizada pelo uso de *positive-acknowledgements* (ACK's). Todo membro do grupo, ao receber um pacote, envia ao transmissor uma mensagem de ACK, informando que o pacote foi recebido com sucesso. O transmissor mantém uma lista de ACK's recebidos para cada pacote. Sempre que um pacote é transmitido, um temporizador é ajustado para um intervalo de espera pelos ACK's do pacote. Se os ACK's de todos os receptores chegaram antes do fim do intervalo, o temporizador é desligado. Quando o temporizador expira, o transmissor considera que o pacote não foi recebido por um ou mais receptores. Neste caso, é realizada a retransmissão do pacote para o grupo, e o temporizador é ajustado para um novo intervalo de espera.

Este método possui alguns problemas que podem desaconselhar a sua utilização, dependendo da aplicação. Em primeiro lugar, ao exigir que um pacote enviado seja confirmado por todos os membros do grupo, temos um aumento considerável no número de mensagens necessárias para se manter a comunicação confiável. Além disso, como as mensagens de ACK também estão sujeitas a perdas, ocorrerão retransmissões desnecessárias de pacotes. Com este esquema as entradas e saídas do grupo devem ser notificadas explicitamente, para evitar que o transmissor retransmita um pacote indefinidamente para um membro que tenha saido do grupo.

A alternativa ao mecanismo anterior é o método de confiabilidade baseado no receptor. Neste método, a responsabilidade pela deteção e recuperação de pacotes perdidos é atribuída aos membros que recebem o pacote do grupo. Ao detetar um pacote perdido, o receptor deve solicitar a sua retransmissão através de um *negative-acknowledgement* (NACK). A deteção de pacotes perdidos é feita através da verificação de um número de sequência do pacote, associado ao transmissor. Caso o último pacote recebido de um determinado membro do grupo tenha sido i, e chegue o pacote $i + n$, todos os pacotes do intervalo $[i + 1, i + n - 1]$, transmitidos por este membro, são considerados perdidos. Também podem ser utilizadas mensagens de controle para detetar perdas de pacote de uma sequência enviada por um transmissor [7].

Existem duas variações deste método para resolver o problema de recuperar os pacotes

perdidos. Na primeira delas, assim que ocorre a deteção da perda do pacote, é enviado um NACK diretamente para o transmissor, através de uma conexão unicast. O transmissor, ao receber o NACK, retransmite o pacote para todo o grupo. O receptor utiliza temporizadores para enviar novas requisições no caso de perda do NACK ou do pacote retransmitido.

A segunda variação procura reduzir o *overhead* do mecanismo através de atrasos aleatórios no envio de NACK's e retransmissões. Nesta versão, quando é detetada a perda de um pacote, o NACK não é enviado imediatamente, mas sim após expirar um temporizador de envio de NACK. Um NACK é enviado para todo o grupo, ao invés de apenas para o trasmissor como com caso anterior. A idéia é que qualquer membro do grupo que tenha recebido o pacote corretamente possa participar da retransmissão, e possivelmente diminuir o atraso de recuperação (por exemplo se o membro que retransmite o pacote estiver mais próximo do requisitante). O fato de vários participantes poderem re-enviar um pacote pode causar um excessivo aumento de tráfego, pelo envio de várias cópias do pacote. Para diminuir o número de cópias, quando um participante pronto para enviar um NACK, recebe outro NACK pelo mesmo pacote, o envio do seu próprio NACK é abortado e acionado um temporizador para espera do pacote. Caso o pacote seja recebido antes do fim do intervalo, o temporizador é desligado. Por sua vez, se este segundo temporizador expirar, um NACK é enviado para todo o grupo, e o temporizador novamente ajustado. Ao receber um NACK, qualquer membro do grupo que possua uma cópia do pacote solicitado pode enviar a retransmissão. É utilizado um esquema análogo para reduzir as retransmissões. É importante notar que o intervalo de tempo dos temporizadores é aleatório de forma a reduzir o número de retransmissões.

A solução baseada no receptor não apresenta vários dos problemas encontrados no primeiro método. O overhead provocado por ela é menor, uma vez que os NACK's são enviados apenas quando é detectada a perda de um pacote. Também não existe o problema de retransmissão desnecessária de um pacote devido a perda de um NACK, o que tende a reduzir o número médio de retransmissões por pacote. O desempenho pode ser potentialmente melhor caso seja utilizada a segunda variação do método. Apesar das vantagens apresentadas, a implementação deste método é mais complexa que o esquema baseado no transmissor. Isso inclui o problema de ajuste dos temporizadores utilizados. É claro que, valores grandes para os temporizadores tendem a reduzir o tráfego mas por outro lado aumentam o intervalo decorrido do momento que um pacote é enviado até ser corretamente recebido por todos os membros do grupo. Na seção 4 estudamos o problema de ajuste dos temporizadores da política mais complexa acima (a última descrita) através de um modelo analítico.

4 Análise do Protocolo para Confiabilidade Multicast

Na seção anterior, foram apresentadas alternativas para confiabilidade Multicast baseadas no transmissor e no receptor. Em [10] e [13] são apresentados estudos análiticos em que são avaliados o *throughput* e o atraso médio para recuperação de pacote, de cada uma destas alternativas. O modelo definido para estes estudos possui diversas simplificações, como por exemplo, assume-se que os grupos são compostos por um grande número de

participantes, todos os receptores experimentam a mesma probabilidade de perda na rede, não considera que os receptores podem compartilhar canais da árvore Multicast, e supõe que a transmissão de ACK's e NACK's é perfeitamente confiável.

Em [7] são apresentados diversos resultados de simulação visando obter o número de retransmissões feitas por pacote e atraso na recuperação de pacotes. A ênfase deste trabalho se encontra na análise do comportamento do protocolo para diversos tipos de topologia, como estrela e árvore, variando o número de participantes do grupo.

Uma série de resultados obtidos a partir de extensivas medições *backbone* Multicast (MBone [3]) são apresentados em [12] . Estes dados foram úteis na definição dos parâmetros do modelo apresentado a seguir.

A seguir apresentamos o modelo utilizado neste trabalho. Os estudos foram realizados com o objetivo de avaliar, a sensibilidade das medidas de interesse (por exemplo o número de retransmissões e retardo para recuperação de pacotes) à variação dos intervalos dos temporizadores mencionados acima. A ferramenta TANGRAM-II [2] foi usada para a construção e solução do modelo.

No modelo desenvolvido, o transmissor envia um novo pacote imdiatamente após todos os membros do grupo terem recebido o pacote anterior. Cada receptor implementa a segunda variação do algoritmo de recuperação de pacotes baseado no receptor, onde dois temporizadores são utilizados. Supomos que a rede introduz um retardo aleatório nos pacotes de informação e NACK's, com a mesma média para todos os membros do grupo. Todos os detalhes do algoritmo são levados em consideração, incluindo as probabilidades de perda dependente da topologia. Do nosso conhecimento, não existe na literatura um modelo analítico deste algoritmo com este nível de detalhe.

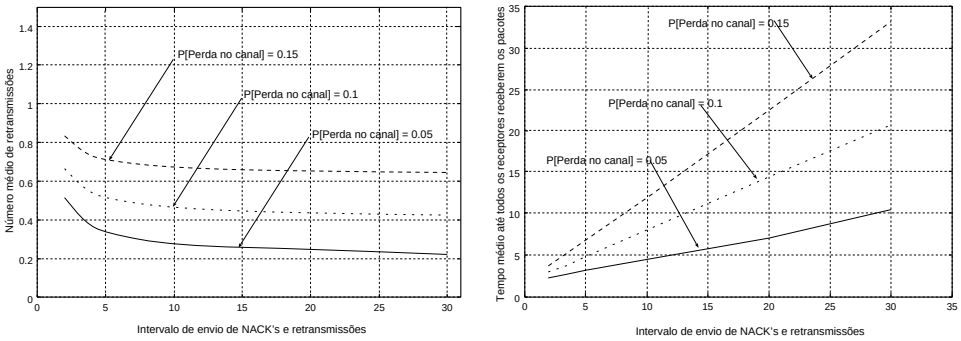

Figura 4: Resultados para atraso na rede de 0.05

Para obter os resultados, foram utilizados modelos com grupos de 4 participantes. As figuras 4, 5 e 6, exibem os valores obtidos para o número médio de retransmissões por pacote, e tempo médio até que todos os membros do grupo recebam um pacote do transmissor. As medidas foram obtidas para várias probabilidades de perda nos canais, e atrasos na rede. Em todos as situações foram utilizados os mesmos valores para os intervalos de envio de NACK e retransmissão, mas isto não é uma restrição do modelo. Todos os tempos exibidos nos gráficos foram normalizados pelo valor do atraso médio fim-a-fim causado pela rede.

Observamos nos gráficos que a medida que aumentamos o valor dos temporizadores de envio de NACK's e retransmissões, o número de médio de retransmissões diminui, e o

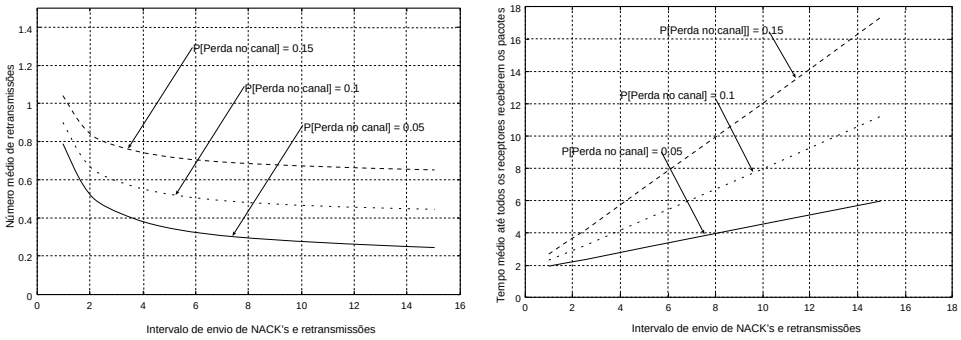

Figura 5: Resultados para atraso na rede de 0.10

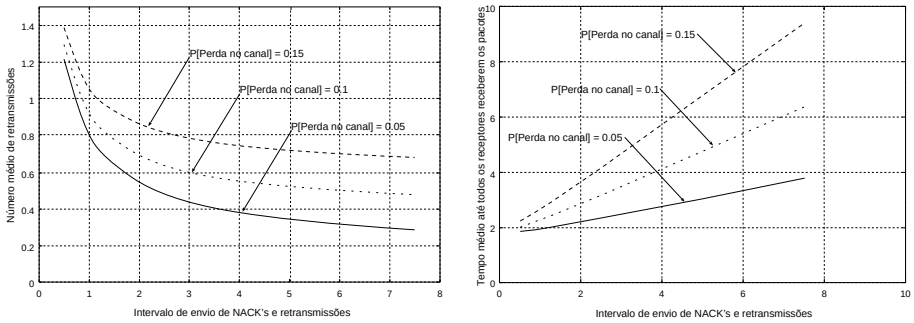

Figura 6: Resultados para atraso na rede de 0.20

tempo médio para recepção do pacote por todos os receptores aumenta. Além disso, para as situações estudadas, o número de retransmissões torna-se menos sensível ao intervalo dos temporizadores à medida que este intervalo aumenta. Isto se explica porque, quando os intervalos são grandes em relação ao atraso da rede, a probabilidade de envio de mais de um NACK ou retransmissão por pacote perdido é baixa.

Por outro lado, o tempo médio decorrido do momenteo que um pacote é enviado pelo transmissor até que todos os receptores o recebam, cresce de maneira linear com o aumento dos intervalos dos temporizadores.

Para determinar o valor para os intervalos de NACK's e retransmissões, o ideal é encontrar um ponto em que o número de retransmissões é baixo, sem que isto acarrete um tempo médio para recebimento do pacote muito alto. Para este fim, foi definida a seguinte função, $f(timer)$, que procura capturar o efeito de ambas as medidas:

$$f(timer) = \frac{1}{E[n^o deRtx(timer)]} * \frac{1}{E[temporec.pacote(timer)]}$$

onde $timer$ é a variável que indica o valor escolhido para os temporizadores.

Esta função assume valores mais altos, quanto menores forem os valores para nossas medidas de interesse. A Figura 7 exibe o comportamento desta função para um atraso de 0.20 na rede, e diversas probabilidades de perda nos canais.

Observamos que, neste exemplo, para a probabilidade de perda igual a 0.15 o valor ótimo dos temporizadores é de duas vezes o atraso médio na rede. Entretanto, à medida que a probabilidade de perda nos canais diminui, a função tem seu máximo deslocado para

maiores valores dos parâmetros dos temporizadores, em relação ao retardo de transmissão. Os exemplos mostram, portanto, que a performance do algoritmo é razoavelmente sensível ao valor destes temporizadores.

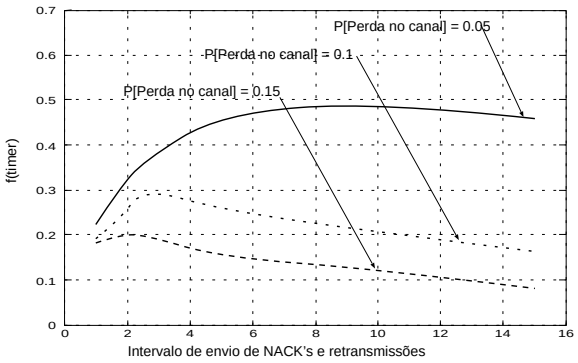

Figura 7: Função de otimização f(timer)

5 Ordenação dos Comandos

Nesta seção, descrevemos como é feita a ordenação dos comandos.Uma vez que a camada de protocolo que implementa a ordenação é feita acima da camada de confiabilidade Multicast, o algortimo de ordenação assume que a comunicação entre os usuários é totalmente confiável, ou seja, não ocorrem perdas de pacotes. Esta suposição é correta, uma vez que esta camada utiliza a camada de confiabilidade Multicast.

A consistência entre os usuários é garantida se todos os usuários executam os comandos na mesma ordem. Para resolver este problema utilizamos a mesma abordagem apresentada em [1], para análise de uma computação distribuída genérica. Os usuários representam os nós desta computação e os comandos são eventos gerados por estes nós. Quando um nó gera um evento, é enviada uma mensagem a todos os outros nós. Desta forma, nosso problema se resume à ordenação de eventos gerados distribuidamente.

5.1 Formalização do Problema

Um evento é uma unidade fundamental de uma computação distribuída, em nosso caso representa um comando gerado na interface. A computação distribuída pode então ser analisada como a execução de um conjunto de eventos, denotado por $\mathcal{E}$. Um evento ϵ é representado pela tupla,

$$\epsilon = \langle n_i, \tau, \Lambda, m \rangle$$

onde

- n_i é o nó em que o evento foi gerado.
- τ é o tempo, dado pelo relógio local de n_i, em que o evento ocorre.

- Λ é o conjunto de mensagens (eventos gerados por outros nós) recebidas por n_i, que influenciam diretamente a geração do evento ϵ.
- m é a mensagem enviada para os outros nós em decorrência do evento ϵ.

A definição de evento apresentada é bastante geral, e deixa em aberto o critério para a definição do conjunto de mensagens Λ. Este critério deve ser definido pela aplicação que está sendo executada de maneira distribuída. No caso do Whiteboard, poderíamos considerar que um comando (evento) tem influência direta na geração de outro se ambos se referem a um mesmo objeto, ou se ambos atuam sobre a mesma área gráfica na interface.

Os eventos em $\mathcal{E}$ estão fortemente inter-relacionados, uma vez que a mensagem que um nó envia ao gerar um evento, pode pertencer ao conjunto de mensagens que influenciam a geração de eventos em outros nós. Definimos então uma relação, denotada por $\rightarrow$, sobre o conjunto de eventos $\mathcal{E}$,

Definição 1 *Se ϵ_1 e ϵ_2 são eventos, então $\epsilon_1 \rightarrow \epsilon_2$ se, e somente se, uma das seguintes condições é satisfeita [1]*

i) ϵ_1 e ϵ_2 são gerados pelo mesmo nó, respectivamente com tempos locais τ_1 e τ_2, tal que $\tau_1 < \tau_2$. Ainda, nenhum outro evento ocorre neste nó no tempo τ, tal que $\tau_1 < \tau < \tau_2$.

ii) ϵ_1 e ϵ_2 são gerados em nós distintos, mas a mensagem m enviada em decorrência de ϵ_1 pertence ao conjunto de mensagens Λ de ϵ_2.

A condição i) expressa o conceito intuitivo de causalidade que existe entre eventos gerados pelo mesmo nó, enquanto que a condição ii) representa a relação de causa-efeito existente entre eventos gerados por nós distintos. A Figura 8 exibe o grafo de precedência $H = (\mathcal{E}, \rightarrow)$, definido pela relação $\rightarrow$, para a execução de uma computação distribuída. O grafo H é um grafo acíclico direcionado, em que o conjunto de nós é definido pelo conjunto de eventos $\mathcal{E}$, e suas arestas são determinadas pelos pares de eventos relacionados por $\rightarrow$.

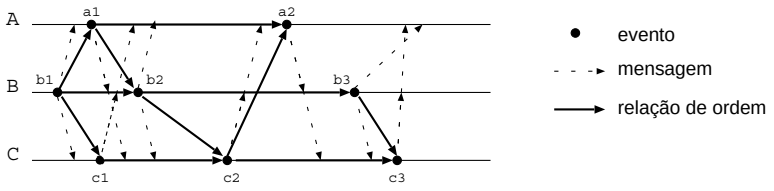

Figura 8: Grafo de precedência para uma computação distribuída

O fechamento transitivo de $\rightarrow$, denotado por $\rightarrow^+$, é irreflexivo e transitivo, portanto estabelece uma ordem parcial para o conjunto de eventos $\mathcal{E}$. Dois eventos ϵ_1 e ϵ_2 que não estejam relacionados por $\rightarrow^+$ são ditos eventos concorrentes. Na Figura 8, os pares de eventos $(a1, c3)$ e $(b1, a2)$ estão relacionados por $\rightarrow^+$, e $a1$ e $c1$ são exemplos de eventos concorrentes. Podemos verificar graficamente se dois eventos estão relacionados por $\rightarrow^+$, se existe um caminho entre eles no grafo de precedência H. A relação $\rightarrow^+$ pode ser usada para definir os conceitos de passado e futuro de um evento. Estes conceitos serão utilizados para explicar o modelo de execução do aplicativo TGWB.

Definição 2 *Para um evento ϵ, definimos [1]*

$$Passado(\epsilon) = \{\epsilon' | \epsilon' \rightarrow^+ \epsilon, \epsilon' \in \mathcal{E}\}$$
$$Futuro(\epsilon) = \{\epsilon' | \epsilon \rightarrow^+ \epsilon', \epsilon' \in \mathcal{E}\}$$

À primeira vista, para garantir a consistência entre os usuários de uma sessão TGWB, bastaria executar os comandos de acordo com os conjuntos definidos acima, utilizando-se a seguinte regra:

Regra 1 *"O comando associado ao evento ϵ deve ser executado após todos os eventos pertencentes ao conjunto $Passado(\epsilon)$, e antes da execução dos eventos em $Futuro(\epsilon)$."*

Entretanto, os conjuntos $Passado(\epsilon)$ e $Futuro(\epsilon)$ foram definidos a partir da relação de ordem parcial $\rightarrow^+$. Portanto, podem existir eventos que não pertençam a nenhum destes conjuntos, ou seja, eventos concorrentes a ϵ. Caso não seja especificada uma ordem de execução única para eventos concorrentes, a ser utilizada em todos os nós, podemos alcançar um estado de inconsistência entre os usuários. O exemplo a seguir, ilustrado pela Figura 9, mostra uma situação em que isto ocorre.

Suponha uma sessão Whiteboard com 3 participantes: A, B e C. Em um determinado instante A desenha um quadrado na sua interface, consequentemente é enviada uma mensagem contendo este comando aos demais participantes. Após receberem e executarem este comando, B e C alteram "simultaneamente" (isto é, cada um dos comandos é gerado antes chegada da mensagem relativa ao outro) a textura da figura para hachurado e preto, respectivamente. Os comandos de B e C geram eventos claramente concorrentes. Caso o critério utilizado para definir a ordem de execução de eventos concorrentes seja a ordem de chegada das mensagens, chegaríamos a um estado de inconsistência em que B possui um quadrado preto, C possui um quadrado hachurado, e a textura do quadrado em A depende da ordem de chegada das mensagesn de B e C.

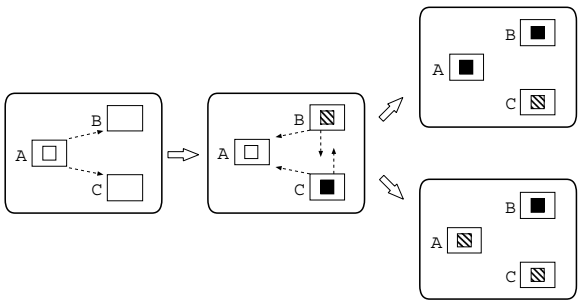

Figura 9: Exemplo de inconsistencia entre usuarios

Para resolver este problema, é necessário estender $\rightarrow^+$ para obter uma relação de ordem total, denotada por $\rightarrow_t^+$. Uma ordem total é uma ordem parcial em que, para todo $\epsilon_1, \epsilon_2 \in \mathcal{E}$, existe uma das relações (ϵ_1, ϵ_2) ou (ϵ_2, ϵ_1). A ordem total não contradiz $\rightarrow^+$, uma vez que ela é obtida a partir de $\rightarrow^+$, pela inclusão dos pares de eventos concorrentes.

Eventos concorrentes não existem quando a ordem total é aplicada. Portanto, os conjuntos $Passado(\epsilon)$ e $Futuro(\epsilon)$ são redefinidos, de acordo com esta nova relação, e a regra 1 passa a garantir a consistência entre os usuários do aplicativo.

5.2 Relógios Lógicos

A implementação da ordenação total de eventos é feita através de relógios lógicos, utilizando o algoritmo apresentado por Lamport em [8]. Um relógio lógico é apenas uma maneira de associar um número a um evento. A definição de relógio lógico deve ser baseada na relação de ordem entre os eventos, assim chegamos à seguinte condição:

Condição 1 *Para quaisquer dois eventos a e b, se $a \rightarrow^+ b$, então $C(a) < C(b)$.*

onde $C(\epsilon)$ é o valor do relógio associado ao evento ϵ.

Para implementar os relógios lógicos é necessário que cada nó n_i mantenha um contador C_i, de tal modo que para um evento a, $C_i(a)$ é dado pelo valor do contador C_i no instante de sua geração. Os contadores C_i são atualizados de acordo com as seguintes regras:

R1. Imediatamente após a geração de um evento, o nó n_i incrementa seu contador de uma unidade

R2. Ao receber uma mensagem, o nó n_i ajusta o seu contador para o valor

$$max\{C_i, C(msg) + 1\}$$

Este mecanismo fornece apenas uma relação de ordem parcial entre os eventos. Como visto anteriormente, esta relação é insuficiente para garantir a consistência entre os usuários. Portanto é necessário estender este mecanismo para que ele suporte a relação de ordem total. Dois eventos a e b são concorrentes se e somente se $C(a) = C(b)$. Isto só é possível se a e b são gerados em nós distintos. Caso cada nó n_i que participa da computação distribuída possua uma identificação única, $Id(n_i)$, sempre podemos definir uma ordem total arbitrária para estes nós, antes do início da computação. Desta forma, definimos

Definição 3 *Sejam a e b são eventos, gerados em n_i e n_j respectivamente. Então $a \rightarrow_t^+ b$ se, e somente se, uma das seguintes condições é satisfeita [8]*

i) $C(a) < C(b)$

ii) $C(a) = C(b)$ *e* $Id(n_i) < Id(n_j)$

Para o aplicativo Whiteboard, a identificação de um nó é construída a partir da concatenação do número IP da máquina em que o aplicativo está executando, com o número do processo PID. A ordem entre os nós é definida pela avaliação lexicográfica do campo IP-PID.

5.3 Algoritmo para Execução dos Comandos

A Regra 1, apresentada na seção 4.1, permite definir um algoritmo simples para a execução dos comandos. De acordo com este algoritmo, uma vez gerado ou recebido um novo comando, ele seria armazenado até que todos os comandos pertencentes ao seu passado tenham sido recebidos e executados, quando então ele poderia ser executado. O esquema assim definido é simples, mas apresenta dois problemas: 1) pouca agilidade para a aplicação; 2) necessidade de detectar se todo o passado de um comando já foi recebido.

O primeiro problema consiste em que um comando, ao ser gerado ou recebido, não pode ser imediatamente executado, mas deve esperar até que todos os comandos em seu passado sejam executados. Isto pode aumentar significativamente o tempo de latência entre a geração do comando por um usuário e a sua execução em todas as interfaces. Além disso, este algoritmo também apresenta uma situação inusitada ao usuário, pois os comandos que ele gera em sua interface local não são imediatamente executados, mas sim armazenados para execução futura. Certamente este não é um efeito desejável para um aplicativo interativo como o Whiteboard.

O segundo problema é um pouco mais complexo, e consiste em determinar para um determinado comando c se todos os comandos em seu passado já foram recebidos. Os comandos gerados localmente no passado de c não apresentam dificuldade pois já se encontram todos executados ou armazenados. A dificuldade está em saber se algum usuário gerou um comando no passado de c que ainda não foi recebido.

Devido aos problemas relacionados acima, buscamos uma solução alternativa para a execução dos comandos. Uma observação importante, é que a relação de ordem total, necessária para garantir a consistência entre os usuários, é muito mais forte que a relação de causa-efeito exigida pela semântica do aplicativo TGWB. Isto significa que, em muitos casos, apesar da relação $\rightarrow_t^+$ determinar que um comando c_1 está no passado de um outro comando c_2, a troca na ordem de execução entre eles não iria afetar o resultado final. Isto ocorre, por exemplo, quando os comandos se referem a objetos diferentes, e atuam sobre áreas diferentes da interface gráfica.

Além disso, o editor gráfico TGIF fornece o recurso de desfazer a execução de um comando, através da função $Undo\_a\_Cmd()$. Desta forma, sempre podemos desfazer a execução de um conjunto de comandos, "voltar ao passado da sessão", e executar os comandos novamente em outra ordem. Esta operação é chamada de *RollBack and Recovery*. Este mecanismo é utilizado em outras aplicações, como por exemplo simulação distribuída [11].

Definimos o algoritmo para execução dos comandos levando em conta as duas observações anteriores. A idéia fundamental deste mecanismo é tentar executar os comandos assim que eles são gerados ou recebidos pela rede. Em alguns casos, a execução de um comando não é possível, por exemplo porque o objeto a que ele se refere não existe na interface local. Quando isto ocorre, o comando é marcado como inconsistente e armazenado para execução futura. Quando chega um novo comando, pode se verificar que a ordem de execução utilizada viola a ordem estabelecida pela relação $\rightarrow_t^+$. Neste caso, é realizada a operação de $RollBack$, e os comandos são novamente executados, desta vez na ordem correta.

Para implementar este mecanismo, cada participante da sessão mantém uma lista com todos os comandos conhecidos por ele. Esta lista é ordenada de acordo com o sistema de relógios lógicos. Cada comando da lista pode se encontrar no estado executado ou inconsistente, fig. 10.

Quando um novo comando, $novo\_cmd$, é recebido pela rede, identifica-se o seu ponto de inserção na lista. Os comandos que se encontram antes deste ponto de inserção pertencem a $Passado(novo\_cmd)$, enquanto que os comando após o ponto de inserção pertencem a $Futuro(novo\_cmd)$, fig. 11. De acordo com a Regra 1, o comando $novo\_cmd$ deve ser executado antes de todos os comandos em seu futuro. Portanto, é necessário realizar a

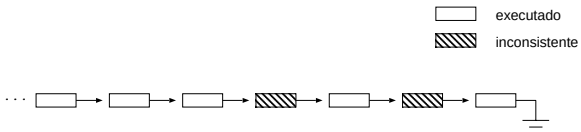

Figura 10: Lista de comandos

operação de *RollBack*. Para fazer isto, é aplicada a função $Undo\_a\_Cmd()$ a todos os comandos em $Futuro(novo\_cmd)$ que se encontram executados. O comando *novo_cmd* é então inserido na lista na posição indicada, e executado. A seguir, os comandos em seu futuro são novamente executados, respeitando-se a ordem da lista (fig. 12).

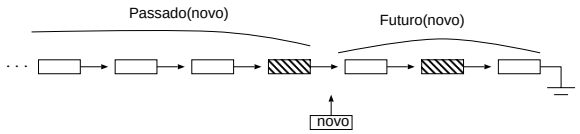

Figura 11: Lista de comandos

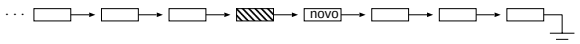

Figura 12: Lista de comandos

Para evitar que a lista de comandos conhecidos se torne muito grande, é necessário implementar algum mecanismo para detectar o ponto a partir do qual os comandos não precisam mais ser armazenados. Para isto, é suficiente que cada membro do grupo envie, periodicamente, uma mensagem informando qual o último comando gerado por ele. Desta forma, cada participante tem a condição de detectar se já recebeu todas as mensagens no passado de um determinado comando, e eliminá-lo da lista.

6 Conclusão

Neste artigo descrevemos os problemas inerentes à implementação de um aplicativo Whiteboard. Esses problemas não são triviais uma vez que o aplicativo demanda a execução de comandos gráficos na mesma ordem por todos os usuários (interligados por rede) participantes do grupo. É necessário então recuperar os pacotes multicast perdidos pela rede e elaborar um algoritmo para garantir que o resultado final (o desenho na tela) seja idêntico para todos os usuários, sem que essa exigência incorra em retardos inaceitáveis para os usuários.

Devido à complexidade destes problemas, os whiteboards existentes em domínio público são limitados. Alguns aplicativos funcionam com um único transmissor enviando desenhos a vários receptores, sendo que só o usuário transmissor pode alterar o desenho. Outros não permitem que usuários distintos possam operar sobre um desenho criado por outro usuário. Outros simplesmente não garantem a consistência do desenho. O aplicativo que desenvolvemos não possue estas limitações, e permite que o mesmo desenho possa ser completamente e simultaneamente modificado por qualquer um dos usuários do grupo, garantindo também, de forma eficiente, a confiabilidade das informações trocadas.

Estudamos o desempenho do algoritmo de confiabilidade multicast empregado em relação aos valores dos temporizadores usados. O modelo desenvolvido é complexo e inclui detalhes importantes não levados em consideração por outros modelos. Os resultados mostram a importância de se fazer o *tunning* dos temporizadores usados. Está em andamento no momento a comparação dos resultados apontados pelo modelo com medições feitas na nossa rede experimental.

Atualmente o aplicativo encontra-se com todas as camadas de protocolo funcionando e será em breve disponibilizado em domínio público, após o término do período de testes da versão presente. No futuro este aplicativo será totalmente integrado com a ferramenta TANGRAM-II de modelagem e análise, usada no desenvolvimento do modelo desenvolvido neste trabalho.

7 Agradecimentos

Gostariamos de agradecer a William C. Cheng, projetista do TGIF, pela implementação de recursos adicionais no TGIF que permitiram o desenvolvimento do TGWB.

Referências

[1] Valmir C. Barbosa. *An Introduction to Distributed Algorithms*. The MIT Press, 1996.

[2] R.M.L.R. Carmo, L.R. de Carvalho, E. de Souza e Silva, M.C. Diniz, and R.R. Muntz. Performance/Availability Modeling with the TANGRAM-II Modeling Environment. *Performance Evaluation*, 33:45–65, 1998.

[3] S. Casner. Frequently Asked Questions (FAQ) on the Multicast Backbone (MBONE).

[4] William C. Cheng. URL http://bourbon.cs.umd.edu:8001/tgif/. TANGRAM Graphic Interface Facility.

[5] Daniel Ratton Figueiredo, Flavio Pimentel Duarte, and Edmundo de Souza e Silva. Implementação de um aplicativo de voz com ánalise e caracterização de seu tráfego. In *XXIV Seminário Integrado de Software e Hardware*, pages 531–540, Aug. 1997.

[6] D.R. Figueiredo, B.R. Brandi, and E. de Souza e Silva. Mecanismos para Recuperação de Perdas de Pacotes de Voz na Internet. In $16^{\underline{th}}$ *Brazilian Computer Networks Symposium*, pages 52–67, Rio de Janeiro, Brazil, May 1998.

[7] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framming. *Proc. ACM Sigcomm95*, pages 342–356, 1995.

[8] L. Lamport. Time, Clocks, and Ordering of Events in a Distributed System. *Commun. ACM 21, 7*, pages 558–565, 1978.

[9] S. McCanne. A distributed whiteboard for network conferencing. *UC Berkeley CS 268 Computer Networks term project*, May 1992.

[10] S. Pingali, D. Towsley, and J. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *Proc. 1994 ACM Sigmetrics Conf.*, May 1994.

[11] R. Rigther and J.C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, Jan 1989.

[12] Maya Yajnik, Jim Kurose, and Don Towsley. Packet Loss Correlation in the MBone Multicast Network. *IEEE Global Internet Conference*.

[13] Miki Yamamoto, James F. Kurose, Donald F. Towsley, and Hiromasa Ikeda. A Delay Analysis of Sender-Initiated Reliable Multicast Protocols. *Proc. of ACM SIGCOMM Conf*, 25(4):342–356, 1995.