SMTA: Um Sistema para Monitoramento de Tráfego em Aplicações Multimídia

M. C. M. A. Pinheiro marcos@info.ufrn.br
T. Batista thais@ufrnet.br
G. L. Souza Filho guido@dimap.ufrn.br

Grupo NatalNet - Departamento de Informática e Matemática Aplicada – DIMAp
Universidade Federal do Rio Grande do Norte – UFRN
Campus Universitário – Lagoa Nova – 59072-970, Natal - RN

Resumo

A utilização de aplicações multimídia inerentemente dinâmicas no ambiente da Internet gera alto tráfego na rede podendo levar à uma situação de grande congestionamento. Para evitar que o congestionamento comprometa as aplicações em operação, é interessante monitorar dinamicamente o tráfego na rede e evitar o envio das mesmas informações para uma mesma localização. Neste trabalho apresentamos o SMTA, um sistema de monitoramento de tráfego por aplicação que identifica dinamicamente os pontos com sobrecarga e exibe um grafo da topologia de distribuição do tráfego da aplicação sendo monitorada destacando, no grafo, os pontos de sobrecarga.

Palavras Chave: Monitoramento, Reconfiguração, Congestionamento, Multimídia.

1. Introdução

O grande crescimento da Internet nos últimos anos se deve principalmente a possibilidade de trabalhar com outras mídias, além da texto, como por exemplo: imagens e áudio. Existe atualmente uma necessidade crescente de transmissão de vídeo pela Internet para dar suporte a aplicações como Vídeo sob Demanda, Transmissão de TV ao vivo, etc. Portanto, se hoje a velocidade de transmissão é um grande problema da Internet, com a utilização cada vez mais frequente de dados multimídia, o problema de velocidade tende a se agravar bastante.

Este problema se torna cada vez mais crítico face às aplicações multimídia atuais, como é o caso de Televisão interativa. Este tipo de aplicação possui uma grande quantidade de usuários e o fluxo de usuários fazendo acesso ao sistema é bastante dinâmico: constantemente novos usuários estão tentando conectar-se ao sistema e outros usuários estão desconectando-se. Portanto, é necessário um sistema que monitore continuamente o acesso ao sistema e realize uma alocação de recursos que melhor utilize a capacidade de transmissão da rede.

Para fazer uma alocação prévia realmente eficiente seria necessário conhecer todos os atuais e futuros clientes da aplicação e onde estes clientes estão localizados. Com a abrangência mundial da Internet é praticamente impossível colocar em prática esta abordagem.

Consequentemente a melhor solução para fazer um uso eficiente da estrutura de transmissão é distribuir inicialmente os recursos segundo algum estudo preliminar e monitorar o tráfego na rede identificando dinamicamente pontos com sobrecarga para a partir daí fazer as realocações necessárias. A idéia dessas realocações é utilizar uma solução semelhante ao *MBONE* [1] (para transmitir tráfego *multicast* [2]), onde procura-se evitar o envio de vários fluxos para uma mesma localização identificando pontos de interseção nos caminhos para os vários destinos e transmitindo apenas um fluxo ao longo de segmentos comuns.

O monitoramento de tráfego pode ser classificado como genérico ou específico de uma aplicação. No monitoramento genérico todo o tráfego na rede é monitorado enquanto que no monitoramento específico por aplicação apenas o tráfego gerado por uma dada aplicação é considerado. Enquanto o monitoramento genérico permite uma visão mais realista do desempenho da rede o monitoramento específico permite uma melhor configuração das aplicações. Existem vários sistemas de monitoramento de tráfego, mas a maioria deles não se preocupa com um serviço específico mas sim com a rede como um todo. Neste trabalho apresentamos um sistema de monitoramento do tráfego de aplicações multimídia (SMTA) que identifica dinamicamente os pontos com sobrecarga e exibe um grafo [3] com a topologia de distribuição do tráfego de uma aplicação. Neste grafo são destacados os pontos de sobrecarga. A particularidade deste sistema está no tratamento diferenciado do monitoramento para diferentes classes de aplicações. Por exemplo, para uma aplicação de distribuição de televisão na Internet é fundamental se ter um monitoramento específico da aplicação por dois motivos. Primeiro porque esta aplicação pode passar de situações de pouca carga para sobrecarga da rede e dos servidores em intervalos de tempo muito curtos, bastando para tal que se inicie a exibição de um programa de grande interesse para os usuários. O outro motivo é comercial. É fundamental para as emissoras de TV saber quantos, quem são e onde estão seus clientes, disto depende o valor que elas cobram pelo espaço publicitário. Apenas um sistema de monitoramento específico para a aplicação permite obter esta informação.

O desenvolvimento do sistema de monitoramento de tráfego de aplicações segue uma abordagem baseada em configuração [4], onde a programação do sistema é dissociada da programação dos módulos que o compõem. Nesta abordagem, a funcionalidade do sistema é implementada por componentes - módulos de software compilados separadamente que interagem através de interfaces [5]. A nível de configuração são definidos os componentes que compõem o sistema e estabelecidas as interações entre eles. Reuso e reconfiguração dinâmica são aspectos beneficiados pelo desenvolvimento orientado à configuração uma vez que, partes do sistema podem ser modificadas e substituídas durante a execução. A necessidade de reconfiguração dinâmica é uma realidade em diversas aplicações cuja disponibilidade é um fator essencial. Neste caso, a execução da aplicação não deve ser interrompida para alteração em algumas partes. Para este propósito, na configuração do SMTA está sendo usada atualmente a linguagem Java.

Este trabalho está estruturado da seguinte forma. A seção 2 apresenta a arquitetura do sistema. A seção 3 mostra detalhes sobre a implementação dos módulos de monitoramento de tráfego e de desenho do grafo da rede. A seção 4 comenta alguns trabalhos que endereçam o problema de monitoramento de tráfego. Na seção 5 são delineadas as conclusões e comentados a utilização do SMTA no contexto de um projeto mais amplo.

2. Arquitetura do Sistema

O sistema de monitoramento de tráfego de aplicações multimídia tem como finalidade identificar congestionamento em ligações entre elementos de uma rede de computadores (roteadores, clientes, servidores, etc) e exibir os resultados em um grafo que ilustra a topologia da rede de distribuição de tráfego, enfatizando os pontos de congestionamento.

O conceito de *congestionamento* de uma rede está intrinsecamente relacionado com as aplicações em operação na rede. Aplicações multimídia, por exemplo, utilizam mais banda passante que aplicações puramente textuais, portanto, acarretam um maior congestionamento na rede. Desta forma, um sistema de monitoramento de tráfego deve considerar parâmetros diferentes para diferentes classes de aplicações em operação.

Neste trabalho, esta flexibilidade é conferida pelo desenvolvimento baseado em configuração. O monitoramento é realizado por um componente que recebe como parâmetro o limite de conexões que cada ligação pode ter de forma a não comprometer a aplicação em operação. Este parâmetro é definido à nível de configuração, portanto pode ser facilmente modificado para aplicações de classes diferentes.

A Figura 1 ilustra a arquitetura do SMTA. Os elementos são representados usando a linguagem gráfica UML (Unified Modelling Language) [6].

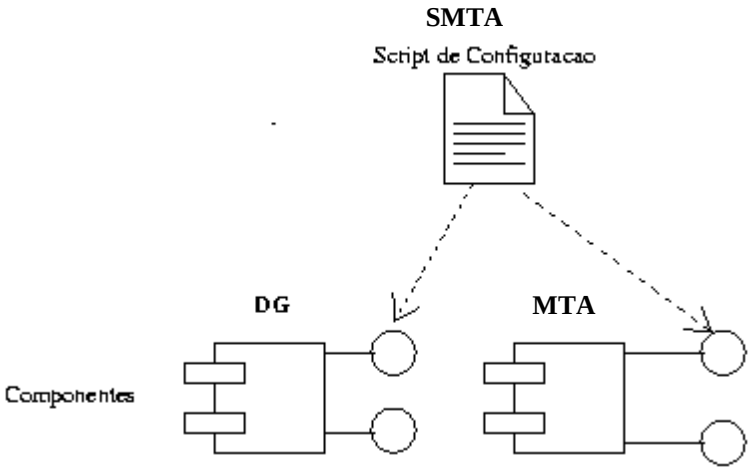

Figura 1: Arquitetura do Sistema de Monitoramento de Tráfego de Aplicação

O módulo de Monitoramento de Tráfego de Aplicação (MTA) verifica continuamente a quantidade de conexões em cada ligação da rede, denominada de *carga da aplicação,* de forma a identificar pontos com excesso de conexões, ou seja, os pontos de congestionamento. Este módulo recebe como parâmetro o limite de conexões que não é significante para acarretar problemas na rede. A idéia é limitar a quantidade de banda passante [7] dos enlaces e equipamentos da rede que podem ser usados por uma aplicação. Por exemplo, uma aplicação que envolve transmissão de Vídeo MPEG-2 (perfil principal, nível principal) compromete em média 6Mbps da rede por fluxo transmitido.

O módulo de Desenho do Grafo da Rede (DG) apresenta uma interface gráfica que exibe o grafo da topologia da rede, ou seja, todos os elementos da rede, as ligações entre eles e o número de conexões abertas em cada ligação. Este grafo apresenta para o administrador da aplicação quantos são e onde estão os clientes da aplicação. O grafo mostra também os pontos de rede com excesso de carga.

3. Implementação

Os módulos de Monitoramento de Tráfego e Desenho do Grafo [8] foram implementados em Java. A interface dos módulos foi descrita na linguagem de Descrição de Interfaces do CORBA (IDL) [9] , de forma a permitir a interoperabilidade e o reuso destes módulos em diversos contextos de uso.

3.1 Monitoramento de Tráfego de Aplicação (MTA)

Ao iniciar, o módulo de *monitortrafego* (MTA) cria uma *rede* e insere nesta um *elementorede* que identifica a máquina onde a aplicação a ser monitorada está executando (*localhost).* Além disso, o MTA cria um objeto *traceroute* [10] passando para este o nome do programa externo utilizado para descobrir rotas. A partir daí, ele fica aguardando por avisos de conexões (e desconexões) no servidor da

aplicação que lhe serão enviados pelo *script de configuração* (SMTA). Para fazer tal sinalização o SMTA chama o método c*onexao* (ou *desconexão*) do módulo MTA passando como parâmetro o endereço IP do cliente que conectou (ou desconectou) na aplicação. A função de cada um desses métodos, e o conseqüente fluxo de execução do programa, esta descrita nas próximas duas sessões.
A Figura 2 descreve a interface IDL do módulo de Monitoramento de Tráfego de aplicação (MTA).

```
struct info_carga{
        ligacao l;
        int carga;
};
typedef sequence<info_carga> lista_congestionados;

interface elementorede{
        attribute string IP;
        void elementorede( in string ender );
}
interface ligacao{
        attibute elementorede obj1;
        attibute elementorede obj2;
        attribute int conexoes;
        void ligacao( in elementorede obj1, in elementorede obj2);
        void incrementa_conexao( );
        void decrementa_conexao( );
        int numero_conexoes( );
}
typedef  sequence<elementorede> lista_elementos;
typedef  sequence<ligacoes> lista_ligacoes;

interface rede{
        attribute lista_elementos lelem;
        attibute  lista_ligacoes ll;
        elementorede obter_elementorede( in string ender );
        ligacao obter_ligacao( in elementorede obj1, in elementorede obj2 );
        boolean sendo_utilizado( in elementorede obj );
        void insere_rota( in rota r );
        void remove_rota( in rota r );
        void insere_elementorede( in elementorede er );
        void remove_elementorede( in elementorede er );
        void insere_ligacao( in ligacao l );
        int remove_ligacao( in ligacao l );
}
typedef sequence<string> rota;
typedef sequence<rotas> rotas;

interface monitortrafego{
        attibute rede r;
        attibute rotas cache_rotas;
        void conexao( in string obj1 );
        void desconexao( in string obj1 );
        lista_congestionados verificaCarga( in int limite );
}
interface traceroute {
        attibute string programa_traceroute;
        void traceroute( in string programa );
        rota obter_rota( in string ender );
}
```

Figura 2: Interfaces IDL do Monitoramento de Tráfego

3.1.1 Conexões

O método *conexão* primeiro se encarrega de descobrir a rota até o endereço IP que lhe foi passado. Para este propósito, é feita uma consulta a tabela *cache_rotas* que mantém em cache as rotas para todos os clientes conectados. Caso esta tabela contenha o registro procurado, um contador que armazena o número de clientes que está utilizando esta entrada é incrementado. Caso contrário, é feita uma chamada ao método *obter_rota*, da interface *traceroute* passando o endereço IP [11] do cliente. Esse método, por sua vez, faz uma chamada ao sistema operacional para executar o programa especificado em *programa_traceroute* (que é quem realmente descobre a rota). Em seguida, captura a saída do *programa_traceroute*, cria um objeto *rota* contendo a rota para o cliente e o retorna para o processo que o chamou (*conexao,* que vai inseri-lo em *cache_rotas)*. Neste ponto, seja através da cache ou da execução de um *traceroute*, o método *conexao* possui um objeto *rota* que contém a rota para o cliente. O próximo passo é fazer uma chamada ao método *insere_rota* da interface *rede* passando *rota* como parâmetro. É este método que efetivamente cria a estrutura de dados que representa o grafo da rede, ou seja as máquinas, as ligações entre elas, e o número de conexões em cada ligação. Para isto o método primeiro obtém, através do método *obter_elementorede* da interface *rede* (este método retorna *null* caso o IP não exista ou o *elementorede* que representa a máquina com o IP dado caso contrário), o *elementorede* que representa a máquina onde está rodando a aplicação sendo monitorada(*localhost*). Depois ele analisa sequencialmente os IPs contidos em *rota* e para cada um deles realiza as seguintes operações: verifica através do método *obter_elementorede* se o IP já existe na *rede*.

A partir desse ponto há duas possibilidades:

Se o IP existir na *rede* verificamos, através do método *obter_ligacao* da interface *rede,* se existe uma ligação entre este IP e o IP anterior na *rota*. (mesmo o primeiro IP de *rota* tem um anterior que é sempre *localhost).* O método *obter_ligacao* retorna *null* caso não exista uma ligação entre os dois IPs ou a *ligacao* entre eles, caso contrário. Caso exista a *ligacao* o valor do atributo *conexoes* desta *ligacao* é incrementado através do método *incrementa_conexao*. Caso contrário, uma *ligacao* entre os dois *elementorede* que representam os IPs é criada (o valor do atributo *conexoes* desta *ligacao é* definido como 1) e esta *ligacao* é inserida na *rede* através do método *insere_ligacao*.

Se o IP não existir na *rede* criamos um *elementorede (*que representa este IP) e o inserimos na *rede* através do método *insere_elementorede*. Criamos também uma *ligacao* entre este *elementorede* e o *elementorede* que representa o IP anterior (o valor do atributo *conexoes* desta *ligacao é* definido como 1) e a inserimos na *rede* através do método *insere_ligacao*.

Após realizarmos esses passos para cada um dos IPs de *rota,* teremos inserido na *rede* informações que dizem qual o caminho do servidor da aplicação até o cliente que o está utilizando e quantas conexões existem entre cada par de máquinas neste trajeto.

A seguir descrevemos graficamente um exemplo de conexão:

Suponhamos que em um determinado instante temos a configuração de uma aplicação na *rede* conforme mostrado na Figura 3a, onde os círculos representam as máquinas, as linhas as ligações, e os números nas linhas a quantidade de conexões abertas naquele segmento.

Se um cliente, por exemplo 10.1.5.1, conectar no servidor da aplicação sendo monitorada e a rota para este, descoberta através do *traceroute*, for: 10.1.1.1, 10.1.2.1, 10.1.3.1, 10.1.5.1 teremos a seguinte sequência de passos:

1. Como já existe uma *ligacao* entre 10.1.1.1 e 10.1.2.1 basta incrementar o número de conexões. (Figura 3b)

2. Como já existe uma *ligacao* entre 10.1.2.1 e 10.1.3.1 basta incrementar o número de conexões. (Figura 3c)

3. Como 10.1.5.1 não existia ele foi criado e foi criada também uma *ligacao* entre ele e o anterior (10.1.3.1). Além disso, essa *ligacao* terá a principio apenas uma *conexao* aberta.

Portanto essa é a nova configuração da rede após a conexão do cliente 10.1.5.1 (Figura 3d).

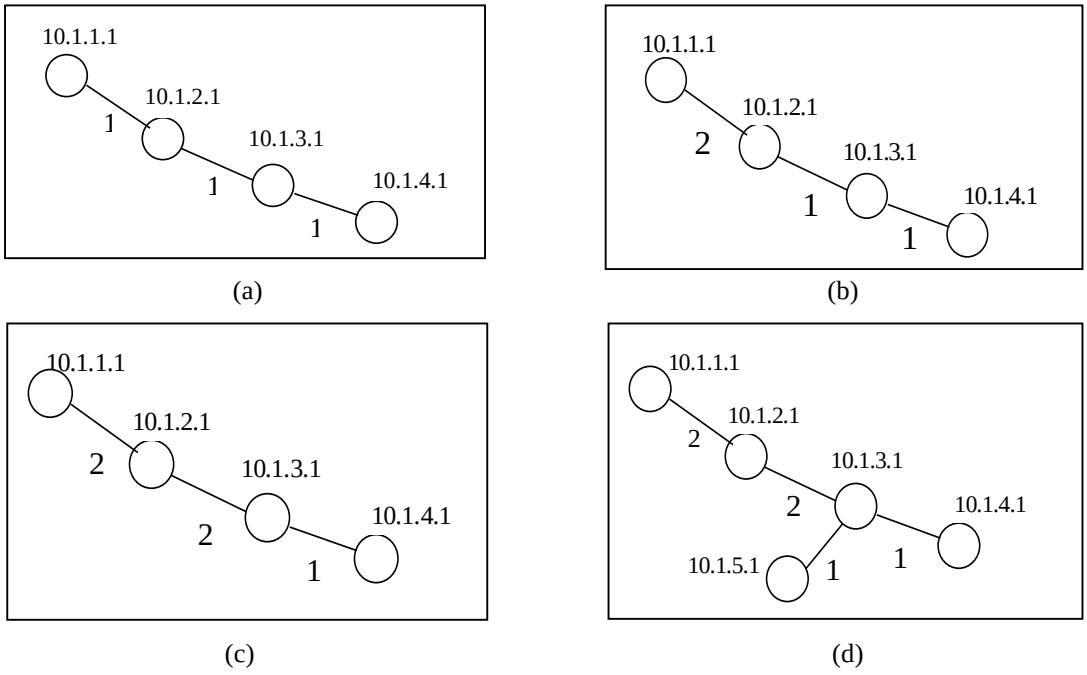

Figura 3 – Processo de conexão de um cliente

3.1.2 Desconexões

Conforme citamos anteriormente, além de receber avisos de conexões o MTA também deve ser informado pelo SMTA quando um cliente desconecta da aplicação sendo monitorada. Vimos também que para fazer tal sinalização o SMTA chama o método *desconexao* do módulo MTA passando como parâmetro o endereço IP do cliente que desconectou da aplicação. O objetivo desse método é retirar da *rede* toda informação que diz respeito ao referido cliente. Isso equivale a decrementar o número de *conexoes* em cada *ligacao* utilizada para acesso ao cliente e eventualmente excluir ligações e máquinas quando a única conexão aberta numa *ligacao* for a do cliente que desconectou. A primeira coisa que o método *desconexao* deve fazer é obter a *rota* até o cliente para depois retirar essas informações da *rede*. Conforme citado anteriormente, para obtermos a *rota* para um determinado cliente devemos primeiro realizar uma busca em *cache_rotas* e, caso essa busca seja mal sucedida, devemos chamar o método *obter_rota* da interface traceroute. Acontece que neste caso a busca nunca vai ser mal sucedida pois para um cliente desconectar ele primeiro tem de ter conectado, e portanto vai estar cadastrado em *cache_rotas* já que as entradas nesta tabela só são removidas no processo de desconexão.

Neste ponto temos uma consideração importante a fazer. A Internet é uma rede dinâmica, e normalmente possui caminhos alternativos entre dois pontos, que são selecionados com base nos critérios dos protocolos de roteamento. Assim sendo, pode acontecer que a *rota* entre o cliente e o servidor seja diferente nos momentos da conexão e da desconexão. Portanto, sempre que um cliente

desconecta, devemos utilizar a *rota* que está em *cache_rotas,* ao invés de usarmos *obter_rota* novamente. Apesar de num primeiro momento essa simplificação que adotamos aparentar que a nossa modelagem da *rede* não corresponde a realidade, devemos considerar que como o tempo em que um cliente permanece conectado é muito pequeno em relação ao intervalo em que essas mudanças de rota ocorrem, portanto, na prática elas não tem grande impacto.

Uma vez que o método *desconexao* tenha obtido a *rota* de *cache_rotas* ele deve passá-la para o método *remove_rota* da interface *rede*. Este método vai remover da *rede* toda informação relativa a conexão desse cliente. Para isso ele primeiro insere no início de *rota* o IP da máquina onde está rodando a aplicação sendo monitorada(*localhost*). Depois ele inverte *rota* para que a pesquisa seja feita do último para o primeiro e passa a analisar sequencialmente os IPs contidos em *rota,* realizando para cada um deles as seguintes operações: obtém através do método *obter_elementorede* o *elementorede* que representa o IP sendo analisado e obtém também, através do método *obter_ligacao* da interface *rede*, a *ligacao* entre o *elementorede* sendo analisado e o *elementorede* que corresponde ao IP anterior na rota. Feito isso, decrementamos o número de conexões nesta *ligacao* através da chamada ao método *decrementa_conexao* da interface *ligacao*. Este método retorna o novo número de conexões na *ligacao* após o decremento. Se o número retornado for zero, excluímos a ligação chamando o método *remove_ligacao* da interface rede e verificamos se é necessário excluir também o *elementorede* anterior através de uma chamada a *sendo_utilizado* da interface *rede*. Este método retorna verdadeiro, indicando que *elementorede* não deve ser excluído, se existe alguma *ligacao* que o referencie e caso contrário, retorna falso, indicando que ele não deve ser excluído.

Neste ponto as alterações na *rede* já foram executadas, restando apenas atualizar *cache_rotas*. Decrementa-se então o contador de clientes utilizando esta *rota*, e se este contador chegar a zero esta entrada é removida da tabela.

A seguir descrevemos graficamente um exemplo de desconexão:

Suponhamos que em um determinado instante temos a configuração da aplicação na *rede* conforme mostrado na Figura 4a, onde os círculos representam as máquinas, as linhas as ligações, e os números nas linhas a quantidade de conexões abertas naquele segmento.

Se o cliente 10.1.4.1, desconectar do servidor da aplicação sendo monitorada, devemos primeiramente obter a *rota* para ele de *cache_rotas.* Esta rota será 10.1.1.1, 10.1.2.1, 10.1.3.1, 10.1.4.1. A partir daí teremos a seguinte sequência de passos:

1.Decrementamos o número de *conexoes* na *ligacao* entre 10.1.1.1 e 10.1.2.1. Como o novo valor é maior que zero nada mais deve ser feito para esse segmento. (Figura 4b)

2. Decrementamos o número de *conexões* na *ligacao* entre 10.1.2.1 e 10.1.3.1. Como o novo valor é maior que zero nada mais deve ser feito para esse segmento. (Figura 4c)

3. Decrementamos o número de *conexoes* na *ligacao* entre 10.1.3.1 e 10.1.4.1. Como o novo valor é igual a zero excluímos a ligação. Além disso devemos verificar se 10.1.4.1 ainda é referenciado por alguma outra *ligacao*. Como isso não ocorre ele é excluído da *rede*. (Figura 4d)

Portanto essa é a nova configuração da rede após a desconexão do cliente 10.1.4.1

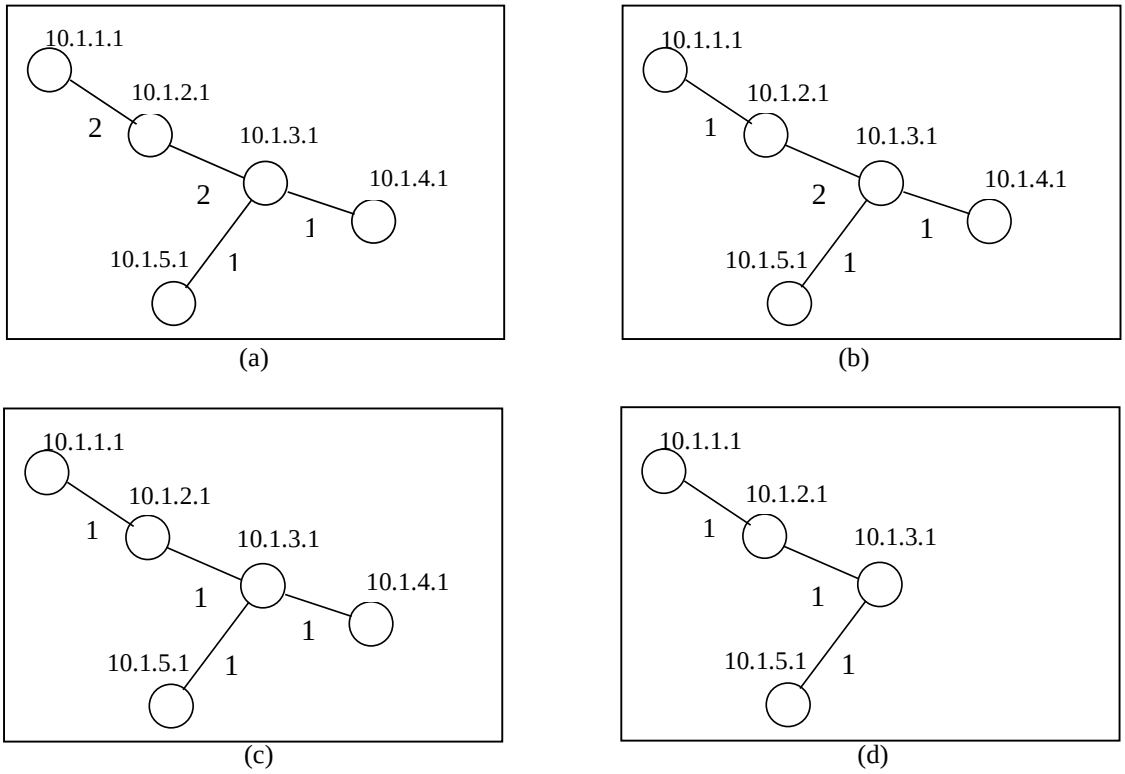

Figura 4 – Processo de desconexão de um cliente

3.2 Desenho do Grafo (DG)

O módulo de Desenho do Grafo é o responsável por criar uma representação gráfica para a estrutura de dados [12] que corresponde à topologia da rede. Essa representação gráfica mostra os elementos que compõem a rede, as ligações entre eles e o número de conexões abertas em cada ligação. Para facilitar a identificação dos pontos de congestionamento o desenho das ligações é feito obedecendo um esquema de cores definido pelo SMTA, que utiliza cores diferentes de acordo com do número de conexões. Esse esquema trabalha com faixas de valores. Por exemplo, se existem de 0 a 10 conexões abertas numa ligação esta deve ser desenhada utilizando a cor verde, se existem de 11 a 50 utilizando laranja e acima de 50 utilizando vermelho.

A interface *grafo* possui basicamente dois métodos, *set_cores* que é utilizado pelo SMTA para a definição do esquema de cores utilizado no desenho da rede, e *desenharede* que é chamado também pelo SMTA sempre que for necessário desenhar / redesenhar a rede.

O método *set_cores* tipicamente só é chamado no inicio do monitoramento, mas pode ser chamado outras vezes, caso se queira trocar o esquema de cores em algum outro instante. Isso pode ser útil em diversas situações. Por exemplo, quando se tem muitos pontos de congestionamento é interessante definir uma escala mais refinada, ou seja com faixas menores.

A Figura 5 apresenta a interface IDL do módulo de Desenho do Grafo (DG).

```
struct esquema_cores {
        int inferior;
        int superior;
        string nome_cor;
}
typedef <sequence> esquema_cores esqcores;

interface graforede{
        void desenharede( in rede r );
        void set_cores( in esqcores );
}
```

Figura 5: Interface IDL do módulo de Desenho do Grafo (DG)

O desenho de estruturas de rede pode seguir duas abordagens: uma que distribui os elementos baseado em suas localizações geográficas, e outra que não leva isto em conta, se preocupando principalmente em minimizar a sobreposição de elementos e o cruzamento de ligações.

Preferimos adotar esta última, pois apesar da abordagem geográfica ser bastante interessante, na prática ela é limitada por diversos fatores, comentados a seguir:

- Uma das técnicas utilizadas para obter a localização geográfica de um elemento de rede é através do DNS reverso [13]. O problema é que os elementos de rede principais que compõem nossa aplicação são roteadores e normalmente os roteadores usam endereços privados e não tem DNS reverso.
- Mesmo quando se tem sucesso com o DNS o resultado é muito genérico, normalmente agrupando-se todos os elementos por país. Para melhorar o resultado uma outra técnica é tentar descobrir a latitude e longitude usando as informações do endereço de quem registrou o domínio DNS [14], que podem ser obtidas através de pesquisas com *whois*. Essa é uma técnica bastante interessante e que obtém resultados bem mais específicos, mas que podem no entanto não corresponder a realidade uma vez que alguém pode registrar um domínio em um local e instalar os equipamentos em outro.

A figura a seguir ilustra um exemplo de rede desenhada pelo módulo DG que não usa informações geográficas.

Na Figura 6 podemos identificar que os clientes 10.9.96.16, 10.248.0.16, 200.17.143.130 e 200.17.143.31 estão acessando o servidor (aqui identificado por 127.0.0.1), sendo que o cliente 200.17.143.31 tem duas conexões abertas com este.

Podemos observar também que entre cada par de roteadores está indicado o número de conexões que está passando na ligação, por exemplo, no link entre os roteadores 192.168.1.11 e 192.168.1.1 estão passando três conexões. Além disso, as ligações foram desenhadas de acordo com um esquema de cores predefinido.

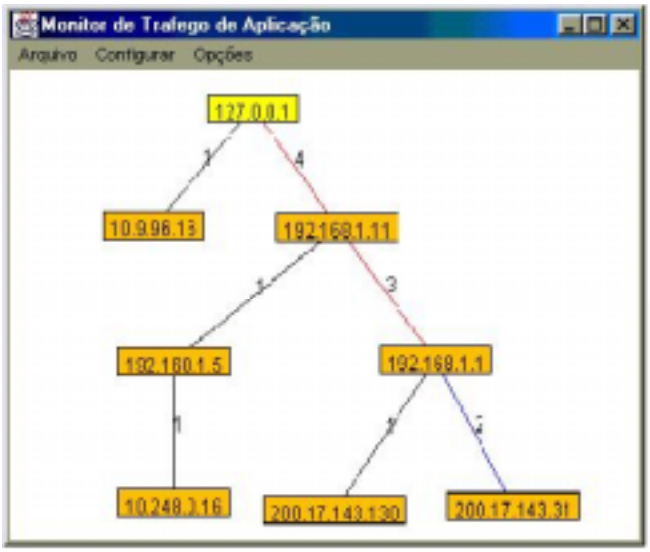

Figura 6 – Exemplo de saída do módulo DG

4. Trabalhos Correlatos

Os sistemas discutidos nesta sessão, ou tratam diretamente sobre monitoramento de tráfego ou serviram como base para idéias utilizadas em etapas da solução.

O Graphical TraceRoute [15] é um sistema cujo objetivo é construir um banco de dados de roteadores dos Estados Unidos (EUA) e permitir o acesso a estas informações via web. Este sistema é utilizado através de um applet Java iterativo que permite ao usuário visualizar a rota de um pacote para um destino particular dentro dos EUA. O applet serve como um cliente, uma vez que ele conecta em um servidor Java que é o responsável por executar o *traceroute* e repassar a resposta para o cliente. O servidor Java é encarregado de, constantemente, construir um banco de dados, usando o programa *traceroute*, para mapear a topologia da rede. O problema desta abordagem é que, quando um cliente solicita a descoberta de uma rota, o *traceroute* é executado nessa base de informação e não na rede. Um mapa dos Estados Unidos é utilizado no applet para que se possa desenhar os roteadores levando-se em conta a localização física e um esquema de cores que demonstra como é a ligação entre eles.

O Gtrace [16] é uma ferrementa desenvolvida pela *Cooperative Association for Internet Data Analysis (CAIDA)* [17] que funciona como um visualizador gráfico das saídas do *traceroute*. Ele usa um conjunto de heurísticas para determinar a localização de um elemento de rede baseado na idéia de que normalmente o nome de um nó contém informações geográficas, como por exemplo, o nome de uma cidade ou o código de algum aeroporto. O Gtrace possui um banco de dados que mapeia endereços IP, códigos de cidades e códigos de aeroporto para latitude e longitude. Em suma, este sistema usa as informações geográficas obtidas do nome de um nó para realizar pesquisas no seu banco de dados e desenhar os nós em um mapa de acordo com sua localização física real. (Além da sua base de dados ele consulta também outros serviços na Internet como *NetGeo*[18] e *Whois*.) O grande problema dessa ferramenta é não conseguir obter um bom desempenho quando os nós analisados estão fora dos Estados Unidos, pois as informações na sua base de dados estão concentradas nos EUA. Para o Brasil, por exemplo só existem informações sobre poucas cidades e apenas um ou dois pontos por cidade. Apesar disso, esse sistema pode ser bastante proveitoso se customizado para uma determinada região, uma vez

que ele permite a adição de novos mapas e informações na sua base de dados possibilitando uma visão detalhada e realista da rede.

O InternetTrafficReport [19] monitora o fluxo de tráfego na Internet mostrando valores que determinam o estado dos links, onde valores mais altos representam conexões mais rápidas e mais estáveis. O sistema é composto por diversos servidores espalhados pelo mundo que executam *pings* simultâneos para o ponto sendo analisado. Cada servidor compara o resultado obtido com os últimos resultados deste mesmo servidor e por fim, os resultados de todos os servidores são combinados em um único valor. O inconveniente deste sistema é que ele fornece informações sobre como está o acesso a um determinado destino, mas não detalha informações sobre os pontos intermediários para se atingir esse destino.

5. Conclusões

Uma das vantagens do sistema apresentado neste trabalho é oferecer um monitoramento de tráfego por aplicação, permitindo que a aplicação sendo monitorada utilize a rede de forma mais eficiente. Este aspecto é especialmente interessante para aplicações multimídia que geram um alto tráfego na rede. O sistema aqui apresentado foi desenvolvido no contexto do projeto RMAV ProTeM-RNP NatalNet. O foco principal deste projeto é o estudo de aplicações multimídia distribuídas em redes IP de alta velocidade. Dentre as aplicações estudadas destacam-se Transmissão de TV, Vídeo sob Demanda e Vídeo Conferência. Estas aplicações consomem muita banda passante e tem um potencial de demanda explosivo, justificando o estudo e desenvolvimento de sistemas de monitoramento específicos, como desenvolvemos. O sistema cuja primeira versão esta implementada, permite ao administrador de uma aplicação acompanhar em tempo real a distribuição do tráfego gerado pela mesma.

O uso de uma abordagem baseada em configuração flexibiliza o SMTA uma vez que o módulo de monitoramento de tráfego pode receber valores diferentes para o parâmetro crítico, conforme a aplicação em operação na rede. Este aspecto promove o reuso do sistema em diferentes contextos. Além disso, no programa de configuração pode-se estabelecer diferentes tratamentos para diferentes níveis de congestionamento na rede.

Várias possibilidades de trabalhos futuros emergem do sistema descrito neste artigo. De imediato, sugerimos o desenvolvimento de um módulo de otimização da rede (MO) que recebe a lista de ligações congestionadas e descobre as possíveis realocações de ligações e de conexões na rede, aplicando um algoritmo de otimização. Este módulo deve retornar as ações que devem ser realizadas para otimizar as ligações. No programa de configuração, o administrador da rede pode estabelecer quais ações indicadas pelo OR deverão ser realizadas. Possíveis ações que este módulo pode retornar são: alocação de um servidor secundário em determinado ponto de congestionamento, organização de clientes em grupo caso o roteador origem da ligação congestionada permita multicast.

O MTA realiza o monitoramento de uma aplicação, sendo necessário uma instância dele para cada aplicação a ser monitorada. Uma possibilidade a ser investigada é expandir esse módulo transformando-o numa espécie de framework de gerenciamento com capacidade para monitorar diversas aplicações. Utilizando este framework, cada cliente registraria uma aplicação e o MTA criaria uma *rede* de distribuição de tráfego para cada uma delas.

Um outra necessidade eminente é utilizar um ambiente de configuração com capacidade de reconfiguração dinâmica que permita inclusive adicionar nova funcionalidade no SMTA no decorrer da execução. Neste sentido estamos estudando o ambiente LuaSpace que dispõe de uma linguagem

interpretada e dinamicamente tipada para configuração de aplicações - a linguagem Lua [20] - e um conjunto de ferramentas baseadas nesta linguagem. Acreditamos que a integração do SMTA neste ambiente tornaria o sistema mais flexível pois, dentre outras facilidades, seria possível adicionar novos módulos no SMTA sem necessidade de interrupção da sua execução.

Por fim, várias melhorias no DG podem ser feitas, como por exemplo: permitir consultas SNMP [21] aos elementos de rede; permitir visões diferentes da rede baseadas em critérios como domínios DNS; guardar as informações de acesso dos clientes para analisar o comportamento da aplicação em períodos como horas, dias, etc e por fim criar um banco de dados com informações geográficas para permitir o desenho dos elementos de acordo com sua latitude e longitude, pelo menos para regiões específicas, como uma cidade por exemplo.

Referências

[1] Kevin Savetz, Neil Randall, Yves Lepage. Mbone: Multicasting Tomorrow's Internet. Ed. IDG. 1996

[2] Deering, S., Cheriton, D., "Multcast Routing in Datagram Internetworks ans Extended LANs", em ACM Transactions on Computer Systems, Vol. 8, No 2. Maio de 1990.

[3] Furtado, Antonio Luz. Teoria dos grafos: algoritmos. Livros Técnicos e Científicos.

[4] Kramer, J. & Magee, J. Configuration Programming - A Framework for the Development of Distributable Systems. Proceedings of IEEE International Conference on Computer Systems and Software Engineering (COMPEURO 90). 1990. Tel-Aviv, Israel. IEEE.

[5] Batista, T. & Rodriguez, N. Dynamic Reconfiguration of Component-Based Applications. International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000), June 2000, Limerick, Ireland. To be published.

[6] Booch, G. Jarcobson, I. Runbaugh, J. The Unified Modeling Language User's Guide. Addson-wesley, 1999.

[7] Soares, L.F.G; Lemos, G. e Colcher, S. Redes de Computadores: das LANs MANs e WANs ás Redes ATM. Segunda Edição. Editora Campus. Rio de Janeiro, 1995.

[8] Algorithms for Drawing Graphs: an Annotated Bibliography. Giuseppe Di Battista, Peter Eades. Junho 1994.

[9] Siegel, J. CORBA: Fundamentals and Programming. John Wiley & Sons, 1996.

[10] Jack Rickard. Mapping The Internet With Traceroute.

[11] Commer, Douglas E. & Stevens, David L. Internetworking With TCP/IP. Prentice-Hall International Editions.

[12] E. Horowilz,S. Sahni, Fundamentals of Data Structures, Computer Science Press, 1983.

[13] RFC 1034 - Domain Names - Concepts and Facilities

[14] RFC 1876 - A means of expressing location info in DNS

[16] Periakaruppan R., Nemeth E., GTrace - A Graphical Traceroute Tool
http://www.caida.org/tools/visualization/gtrace/paper/Gtrace.ps

[17] Cooperative Association for Internet Data Analysis (CAIDA).

[18] The Internet Geographic Database, http://www.caida.org/tools/NetGeo

[19] http://www.internettrafficreport.com/

[20] Ierusalimschy, R.; Figueiredo, L. H. & Celes, W. Lua - an extensible extension language. Software: Practice and Experience, 26(6), 1996.

[21] A Simple Network Management Protocol (SNMP). RFC1157. 1990.