Uma Ferramenta para Geração de Tráfego e Medição em Ambiente de Alto Desempenho

Márcio Augusto Parente Leocádio*
Programa de Pós-graduação DCC/IM e NCE
Universidade Federal do Rio de Janeiro

Paulo Henrique de Aguiar Rodrigues**
DCC/IM e NCE
Universidade Federal do Rio de Janeiro

Resumo

Este artigo descreve a implementação e uso de programas de medição e geração de tráfego para realização de medidas precisas em ambientes TCP/IP de alto desempenho e em redes ATM. Problemas críticos de projeto da ferramenta para este ambiente são apresentados e medições da latência de um comutador e do desempenho de um ambiente IP clássico demonstram a utilidade da ferramenta. A ferramenta, desenvolvida dentro do projeto Almadem, tem distribuição pública através de http://www.land.ufrj.br.

Abstract

This article describes the implementation of a tool devised for traffic generation and precise measurements in high performance TCP/IP and ATM environments. Criticai design issues are discussed and measurements of switch latency and CLIP environment performance demonstrate the usefulness of the tool. The tool, developed under the Almadem project, has a public distribution from http://www.land.ufri.br.

1 Introdução

Este artigo descreve a implementação e uso de programas de medição e geração de tráfego em ambiente TCP/IP e ATM. Estes programas, desenvolvidos dentro do projeto Almadem, visam, sobretudo, possibilitar a coleta precisa e confiável de estatísticas para arquiteturas de alto desempenho, como Fast ou Giga Ethernet comutados e ATM.

Num ambiente de rede de alto desempenho, podemos estar interessados em determinar o desempenho de equipamentos isolados, como, por exemplo, a latência de comutadores, ou determinar o desempenho de serviços de rede, como o tempo de processamento de um servidor em um ambiente LANE (emulação de rede local) [lane], ou o tempo de resposta de um servidor em um ambiente CLIP (IP clássico) [clip]. O tempo de resposta de um serviço pode ser inferido a partir do atraso ida e volta das requisições/respostas e das latências específicas dos equipamentos intermediários, existentes no trajeto dos pacotes. Em ambos os casos, medidas de baixa ordem de grandeza e com precisão devem ser feitas.

Trabalho executado dentro do Projeto Almadem, CNPq Protem-III.
' Bolsa Capes, ** Bolsa CNPq EV, RNP.

6 Conclusões

Este artigo descreveu a implementação e uso de programas de medição e geração de tráfego para realização de medidas precisas em ambientes TCP/IP de alto desempenho e em redes ATM, desenvolvidos no âmbito do Projeto Almadem. Problemas críticos de projeto e detalhes da implementação da ferramenta foram apresentados. A utilização prática da ferramenta foi demonstrada na obtenção de medidas de latência de comutadores FORE ASX-1000 e IBM 8265 e na avaliação de desempenho dos servidores ATMARP da Fore e do protótipo do projeto Almadem. Estas medições comprovam a eficácia e adequação da ferramenta na obtenção de medidas de alta precisão em ambiente de rede local de alto desempenho.

Entretanto, a ferramenta, pela precisão que apresenta e facilidade de uso, deverá ser bastante útil em medidas de desempenho de redes metropolitanas e de longa distância, baseadas em ATM ou Frame Relay. Este é o caso das REMAV's e da RNP2. Enviar tráfego no limite da banda contratada, constatando ausência de perda, e medir com precisão o retardo ida e volta nas conexões virtuais estabelecidas será tarefa simples com o uso de monitor_pvc2, no caso de testarmos diretamente as conexões ATM, e monitor_udp2, no caso de testarmos o Frame Relay/ATM, usando a pilha UDP/IP. O uso da variante 2 dos programas permite fixar a duração da sessão e obter estatísticas em arquivos, facilitando a automação dos testes.

A ferramenta atualmente encontra-se na versão 1.2.1. Modificaçõess foram introduzidas, visando nao apenas pequenas correções, como também otimizaçõess de caráter geral. Dentre as principais modificações podemos citar o desacoplamento do cálculo da taxa de bits gerada pela sessão, que antes era apresentado de forma única com base no montante transmitido. Na versão atual são apresentadas as taxas efetivas e teóricas de recepção e de transmissão.

A nova versão conta também com uma nova ferramenta para geração de tráfego de fundo em ATM nativo utilizando PVC's. Todos os geradores de tráfego disponíveis foram otimizados para redução do consumo de CPU e alcance de maior precisão na taxa efetivamente gerada.

Todos os programas foram reestruturados visando torná-los mais robustos, amigáveis e com melhor desempenho. Além disso, novos parâmetros foram acrescentados à linha de saída dos programas, aumentando assim o retorno de informação dos programas para os usuários.

Algumas das facilidades incorporadas na ferramenta foram direcionadas para solucionar problemas práticos a serem vencidos pelo experimento piloto de implantação do backbone ATM da RNP2. Medidas precisas de desempenho são importantes não apenas na caracterização da performance de equipamentos e serviços em si, mas também para viabilizar a especificação de parâmetros para a contratação de serviços de comunicação de alto desempenho. A ferramenta Almadem pode ser obtida em http://www.land.ufrj.br.

7 Referências

[apilinux] Almesberger, Werner; *Linux ATM API Draft*; version 0.4, Julho 1996.
[clip98] Laubach, M., Halpern J., *Classcial IP and ARP over ATM, RFC2225*, Abril 1998.
[ibm98] IBM, *8265 BoilerPlate Release 4.0*, 8265-BP01-00, 1998.
[inarp] Bradley, T., Brown, C., Malis, A., *Inverse ARP Resolution Protocol*, RFC-2390, Agosto 1998.
[lane97] The ATM Fórum, *Lan Emulation Over ATM Version 2 - LUNISpecification*, Julho 1997.
[mars] Armitage, G., *Support fo Multicast over UNI 3.0/3.1 based ATM Networks*, RFC-2022, Nov 1996.
[netp] http://www.netperf.org
[patch1] http://www.nl.linuxfocus.org/English/articles/article33.html.
[patch2] http://hegel.ittc.ukans.edu/projects/kurt, e http://hegel.ittc.ukans.edu/Droiects/utime.
[RFC1483] Heinanen, J., *Multiprotocol Encapsulation Over ATM Adaptation Layer* 5, RFC1483, Julho 1993.

destino, que confirma a aceitação da conexão, retornando um CONNECT. Ao receber o CONNECT, a estação remetente obtém a identificação do SVC aberto e pode então usar este SVC para a transferência de dados IP com o destino. O destino, por sua vez, pode também aproveitar este SVC aberto para enviar dados para a estação remetente.

As estações mantém os mapeamentos mais recentes em cache, sendo que o tempo de envelhecimento é de vários minutos. Após este tempo, as entradas e/ou SVC's correspondentes, se ainda não definitivamente descartados, terão que ser revalidados antes de serem novamente usados, implicando, eventualmente, numa nova consulta ao servidor ou no envio de InATMARP [inarp] no SVC em questão.

Gostaríamos de obter o tempo de resolução do ambiente CLIP, medido desde o instante do envio do ATMARP_request, até o estabelecimento do SVC, com o recebimento do CONNECT. Usaremos duas estações SUN conectadas ao comutador FORE. Configuraremos um servidor ATMARP no próprio FORE, e rodaremos o programa sendwait_udp na estação remetente inicial, com o correspondente refletor_udp na outra estação.

Num primeiro instante, faremos um PING entre as duas estações para que o SVC entre elas seja estabelecido e os caches de ambas sejam preenchidos. Após constatar a atualização de ambos os caches, disparamos o sendwait_udp com L=4 (uma célula por PDU), N=50 e E = 8s. Todas as PDU's enviadas deverão seguir e retornar pelo SVC já aberto. O retardo médio fornecido pela ferramenta é o retardo de referência.

Num segundo instante, zeraremos os caches das estações e dispararemos o mesmo programa sendwait_udp. A primeira PDU transmitida irá disparar o processo de consulta ao servidor ATMARP, estabelecer o SVC e transmitir a única célula desta PDU pelo SVC aberto. A estação refletora imediatamente retorna a PDU pelo mesmo SVC. Antes que uma nova PDU seja enviada (após o tempo de 8s), iremos zerar manualmente os caches das estações, forçando o retardo extra de consulta ao servidor e estabelecimento do SVC a cada PDU enviada. Após termos a estatística deste segundo experimento, a diferença dos tempos entre os dois experimentos nos fornece o tempo de resolução do ambiente CLIP utilizado.

Para efeito comparativo, medimos o desempenho de um servidor ATMARP desenvolvido dentro do próprio projeto Almadem. Trata-se, na realidade, de um servidor MARS [mars], *Multicast Address Resolution Server*, operando como servidor ATMARP. Este servidor foi desenvolvido em linguagem C para Sun Solaris e encontra-se em fase de testes e otimização. Para estas medidas, conectamos uma terceira estação Sun Ultra 170 Mhz ao comutador FORE, e configuramos o ambiente IP Clássico com o servidor nesta terceira estação. Os procedimentos foram exatamente os mesmos. Todas as estações foram conectadas via OC-3.

A tabela abaixo mostra os resultados obtidos:

Ambiente	Retardo médio	Latência CLIP
Com cache ativo	0,623 +/- 0,016 ms	0
Servidor ATMARP FORE	162,206+/0,781 ms	161,583 ms
servidor ATMARP Almadem	150,376 +/- 0,655 ms	149,753 ms

Podemos observar que a consulta inicial ao servidor ATMARP causa um atraso substancial na comunicação em rede de alto desempenho. Por outro lado, observamos que a implementação do servidor Almadem tem um desempenho superior (cerca de 12 ms mais rápido), demonstrando que nem sempre os serviços disponibilizados internamente em comutadores são necessariamente de maior desempenho.

Podemos ver que Atraso B - Atraso A = Tcel + Tcom. Como Tcel = 53x8/155,52 = 2,726 µs, podemos obter imediatamente o tempo de latência dos equipamentos.

Uma outra maneira de determinar a latência dos comutadores é obter Tsun e deduzir Tcom diretamente de Atraso A ou de Atraso B. Para obter Tsun, executamos o programa sendwait com cabo externo fazendo loopback nas portas da Sun. Neste caso o atraso medido é equivalente a Tsun + Tcel, já que a célula é transmitida uma única vez nesta topologia. Realizando as cinco execuções e selecionando o menor valor obtivemos: Atraso C = (0.230220 +/- 0.000575) ms.

A tabela abaixo mostra as três estimativas obtidas para a latência de cada comutador:

Comutador	Fórmula de cálculo de **Tcomutação**	Latência (µs)	Média das **latências** (µs)	Fornecido pelo fabricante
fore ASX-1000	Atraso B - Atraso A - Tcélula	14,29	13,59	**+/- 12**
	(Atraso A - Tsun - 3 Tcélula)/2	13,03		
	(Atraso B - Tsun - 4 Tcélula)/3	13,45		
IBM 8265 (CPSW1)	Atraso B - Atraso A - Tcélula	16,45	18,43	25,7
	(Atraso A - Tsun - 3 Tcélula)/2	20,02		
	(Atraso B - Tsun - 4 Tcélula)/3	18,83		

Como podemos verificar, os valores medidos estão extremamente próximos, dentro de 1,5 vezes o tempo de transmissão de uma célula, no pior caso. Para o Fore, o primeiro cálculo forneceu o pior valor. Já para o IBM, este mesmo cálculo fornece o valor mais baixo. Estas medidas indicam que um único procedimento não pode ser escolhido de forma absoluta Por outro lado, a especificação técnica dos fabricantes é coerente com os valores medidos, sendo a IBM [ibm98] até conservadora em relação a performance de comutação.

Esta mesma metodologia poderia ser usada para medir, por exemplo, latências de comutadores Ethernet ou Fast Ethernet. Nesse caso, usaríamos o sendwait_udp e uma estação adicional rodando o programa refletor. Tsun, neste caso, englobaria todos os atrasos no processamento das duas estações. Tcel seria substituído por Tquadro, que representaria o tempo de transmissão de um quadro no Ethernet.

5,2 Medição de performance de ambiente IP Clássico (CLIP)

Num CLIP, a comunicação entre duas estações numa mesma rede lógica IP (LIS) se faz via SVC's (ou PVC's, num ambiente mais restrito), estabelecidos diretamente entre as estações. Para que o SVC seja estabelecido, a estação remetente tem que descobrir o endereço ATM correspondente ao IP destino. Esta descoberta é feita através do envio de uma primitiva ATMARP_request (semelhante ao ARP convencional de rede local) ao servidor ATMARP.

O servidor ATMARP possui os mapeamentos IP/ATM de todas as estações ativas da LIS. Uma estação ao se tornar ativa, registra-se no servidor ATMARP através de um SVC de controle. Para abrir este SVC, o endereço ATM do servidor tem que estar configurado na estação ou é obtido por meio de informação de gerência ILMI. Em geral, uma estação troca informação ILMI com o comutador ATM ao qual está conectada (sinalização UNI tem que estar habilitada), obtendo a parte alta do seu próprio endereço ATM e endereços de serviços disponíveis na rede, como o do servidor ATMARP.

Após receber o endereço ATM do destino, a estação remetente dispara um SETUP, que é a primitiva de abertura de conexão virtual da UNI. Este SETUP é rateado pela rede ATM até o

Os outros programas da ferramenta apresentam saídas semelhantes. Abaixo, mostraremos dois usos da ferramenta: cálculo da latência de dois modelos de comutadores, Fore ASX-1000 e IBM 8265, e avaliação de desempenho do servidor ATMARP Fore e do servidor ATMARP do projeto Almadem. O equipamento IBM é o comutador utilizado nos projetos REMAV (Redes Metropolitanas de Alta Velocidade) e também o equipamento a ser utilizado na implantação do backbone ATM da RNP2.

5.1 Medição de latência dos comutadores FORE ASX-1000 e IBM 8265

Para medir a latência dos comutadores, iremos utilizar o programa sendwait_pvc, gerando uma única célula por PDU em grandes intervalos (E=500 us). Esta configuração ajuda a diminuir a variância das medidas de latência de cada PDU, pois evita problemas de sincronização ou armazenamento de seqüências de células na estação, que poderiam causar retardos bastantes variáveis. Por outro lado, o tempo maior entre envio de células desacopla os processos transmissor e receptor na estação, ganhando precisão.

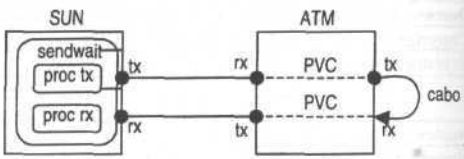
Convém ressaltar que células sempre terão alguma variância no atraso, pois os enlaces são síncronos e células não podem ser transmitidas a qualquer instante. Da mesma forma, comutadores operam sincronamente em sua arquitetura interna, o que pode causar jitter no tempo de comutação. A opção por enviar um número elevado de células nas medições visa obter uma média mais imune a estes fatores imponderáveis.

Uma estação Sun Ultra rodará sendwait_pvc e medirá o atraso ida-e-volta das PDU's. Nos experimentos de medida de latência usaremos L=32 (gerando apenas uma célula por PDU), E=500 e N=50000. Para cada comutador e para cada um dos esquemas indicados, rodaremos cinco vezes o sendwait_pvc, escolhendo o menor atraso médio por pacote obtido. A escolha de cinco rodadas serve com uma garantia adicional, já que medidas da ordem de microssegundos podem ser afetadas por execuções adversas dos sistemas operacionais dos equipamentos. Chamaremos Tsun o atraso devido ao processamento do programa sendwait_pvc na Sun, a cada pacote. Tcel é o tempo de transmissão de uma célula. Tcom é o tempo de comutação que queremos estimar. Nos esquemas abaixo, as portas são todas OC-3 multimodo e pertencem sempre a um mesmo módulo de interface. A estação Sun Ultra usa placa ATM Fore SBA-200.E

- Esquema A

Atraso A = Tsun + 3 Tcel + 2 Tcom

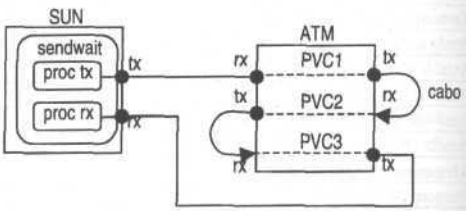
[Text Box1] Estatísticas do Esquema A	
Comutador	Atraso médio por pacote
IBM 8265	(0.278438 +/- 0.000735) ms
Fore ASX-1000	(0.264457 +/- 0.000925) ms



- Esquema B

Atraso B = Tsun + 4 Tcel + 3 Tcom

Estatísticas do Esquema B	
Comutador	Atraso médio por pacote
IBM 8265	(0.297615 +/- 0.000190) ms
Fore ASX-1000	(0.281473 +/- 0.000926) ms



Os programas monitores e refletores são interoperáveis a nível de sistema operacional, ou seja, podemos, por exemplo, utilizar uma variante de programa monitor em uma estação rodando Linux, com seu refletor equivalente em uma estação rodando Solaris. Isto é verdadeiro também para os programas em ATM nativo. Quando tratamos de ATM nativo, no entanto, o processo de reflexão pode ser alcançado pela configuração de um *loopback* no próprio *switch* ATM (dispensando assim o uso de um refletor).

Os programas geradores operam somente sobre a pilha UDP/IP. Como estes programas se prestam apenas à geração de tráfego de fundo, como perturbação ao ambiente sendo testado, a geração de tráfego em ATM pode ser alcançada indiretamente através de um ambiente de IP clássico ou LANE, utilizando-se os próprios geradores UDP. Uma alternativa, caso tenhamos apenas PVC's puros, é o uso dos programas monitor/refletor nativos para geração de tráfego.

Quando utilizamos IP sobre PVC CBR nas estações, a taxa de pico a ser suportada pelo PVC é fornecida ao se configurar a interface lógica IP/ATM na estação. Observe que se o PVC for configurado como CBR com determinada taxa de pico PCR, o driver IP da estação limitará o tráfego a ser gerado nesta interface lógica IP/ATM a este valor PCR, ainda que a aplicação, usando programação *socket,* tente gerar um tráfego acima de PCR.

S Exemplos de utilização da ferramenta

Para exemplificar, a tabela abaixo mostra a saída típica da ferramenta sendwait_pvc. Podemos observar que temos todas as informações necessárias para uma análise e avaliação consistente dos resultados. Em especial, os intervalos de confiança medidos fornecem o grau de precisão da estatística coletada. No exemplo, como L=32 e o overhead no ATM é de 16 bytes (8 de AAL5 + 8 de LLC/SNAP), temos 48 bytes exatos e uma única célula será transmitida por PDU. Podemos checar este comportamento, consultando as estatísticas de células recebidas e transmitidas no comutador. A saída indica ausência de perda nas 50.000 células transmitidas.

Exemplo de Saída de SendWait_PVC	
Conectado ao PVC (0.116)...	
Enviadas 50000 PDU's AAL5.	
Sessao concluida com sucesso.	
./sendwait_pvc: versao 1.1	
Programa desenvolvido pelo Laboratorio de Modelagem/Analise e	
Desenvolvimento de Sistemas de Computacao e Comunicacao (LAND).	
Copyright (c) 1999 Nucleo de Computacao Eletronica (NCE/UFRJ).	
Sugestoes/Perguntas/Bugs: mldio@land.ufrj.br	
Estatisticas (intervalo de confianca de 90%):	
Numero de pacotes gerados	50000
Numero de pacotes recebidos	50000
Tamanho de pacote utilizado	32 bytes
Perda total	0.00 %
Atraso total da transmissao	25186.280 ms
Volume total recebido	12800000 bits
Intervalo de geracao de pacotes	500 us
Banda Util	508.21 kbps
Atraso medio por pacote	**(0.264457 +/- 0.000925) ms**
Numero de amostras	50000
Atraso medio entre pacotes	(0.503728 +/- 0.001151) ms
Numero de amostras	49999

socket é finalmente associado com a chamada bind(). O ponto de acesso de serviço (SAP) é registrado na entidade de sinalização local com a chamada **listen**().

Ao retornar da chamada **listen**(), o processo pode ficar bloqueado na chamada accept(), até que uma solicitação de conexão seja recebida, ou pode realizar interrogações regulares através de chamadas accept() não-bloqueantes ou utilizando a chamada **select**(). Se uma solicitação de conexão atende à especificação do SAP, a chamada accept() é bem sucedida (ou o *socket* se torna legível de forma que a chamada select() retorna e a chamada accept() pode ser invocada). Sendo a chamada accept() bem sucedida, novo *socket é* criado para a conexão e a troca de dados pode ter início.

O *socket* utilizado para receber conexões e o(s) *socket(s)* utilizados para troca de dados podem ser individualmente fechados com a chamada close(). O fechamento de um *socket* utilizado para recebimento de conexões apenas remove o registro SAP e não afeta quaisquer conexões que já tenham sido estabelecidas.

O ponto terminal local pode ser associado a um endereço específico com a chamada **bind**(). Se uma chamada **connect**() não bloqueante for disparada, o processo continua a executar, enquanto a conexão é estabelecida, e pode também decidir fechar o *socket,* prematuramente.

Por ser a API baseada no conceito de programação *socket,* as chamadas citadas são as mesmas utilizadas em programação TCP/IP tradicional. A instalação da API acrescenta ao sistema duas novas famílias de protocolos e suas estruturas: PF_ATMSVC e PF_ATMPVC.

A fase de troca de dados, tanto em PVC's como em SVC's, é realizada através das chamadas **write**() e read() habituais.

As opções de QoS são passadas ao sistema operacional na chamada **setsockopt**, utilizando o nível SOL_ATM e a opção SO_ATMQOS, através da estrutura **atm_qos** especificada em "/usr/include/linux/atm.h":

```
struct atm_qos {
    struct atm_trafprm txtp;
    struct atm_trafprm rxtp;
    unsigned char aal;
};
```

onde txtp e rxtp são os parâmetros de tráfego nas direções de transmissão e recepção, respectivamente. A estrutura **traf_prm**está especificada em "/usr/include/linux/atm.h":

```
struct atm_trafprm {
                    unsigned char       traffic_class;
                    int         max_pcr;
                    int             pcr;
                    int         min_pcr;
                    int         max_cdv;
                    int             max_sdu;
};
```

4.3 Comentários sobre a implementação

Na versão 1.1 da ferramenta, a única informação relativa à qualidade de serviço disponível ao usuário é a PCR *(Peak Cell Rate)* desejada. Se o valor especificado for igual a zero, o descritor será associado a um PVC do tipo UBR (*UnspecifiedBit Rate).* Caso o valor especificado seja positivo, o descritor será associado a um PVC CBR, com taxa de pico igual a PCR solicitada. Atualmente a ferramenta encontra-se disponível para uso apenas sobre PVC's, sendo toda a execução feita em modo síncrono (bloqueante). O tráfego gerado pela ferramenta é estritamente aquele especificado pelo usuário.

com a chamada **socket()**. Eventualmente, opções de QoS poderão ser configuradas nesta fase, através da chamada **setsockopt()**. Feito isso, o *socket* estará apto para originar uma conexão (lado ativo) ou ficar a espera por conexões (lado passivo). Nesta fase, as chamadas **bind() ou connect(),** que possuem funcionalidade idêntica quando utilizadas com PVC's, são utilizadas tanto do lado originador como do lado terminador.

Os trechos de código, a seguir, foram extraídos da implementação e ilustram como é feita a abertura de uma conexão PVC, onde o tipo de PVC é determinado pelo parâmetro <pcr> fornecido pelo usuário.

```
raemset(&qos,O,sizeof(qos));
if (pcr > 0) {
  qos.txtp.traffic_class = ATM_CBR;
  qos.txtp.min_pcr       = pcr;
}
else
  qos.txtp.traffic_class = ATM_UBR;
qos.txtp.max_sdu = 8192;
qos.rxtp                 = qos.txtp;
qos.aal                  = ATM_AAL5;

if (setsockopt(fd,SOL_ATM,SO_ATMQOS,&qos,sizeof(qos) ) < 0) {
  perror("setsockopt SO_ATMQOS");
  return -1;
}



memset(&addr,0,sizeof(addr));
addr.sap_family = AF_ATMPVC;

addr.sap_addr.vpi = vpi;
addr.sap_addr.vci = vci;
if (connect(fd,(struct sockaddr *) &addr,sizeof(addr)) < 0) {
  perror("connect");
  return -1;
}
```

Um detalhe interessante nesta API é que PVC's podem ser abertos em dois passos: primeiro, apenas uma parte do descritor de endereço é fornecida, de forma que os recursos necessários à conexão possam ser alocados; segundo, o descritor de endereço completo é fornecido. O endereço utilizado no primeiro passo é denominado um *endereço não completamente especificado.* Concluída a fase de conexão, dados podem ser trocados até que o *socket* seja fechado com a chamada **close()**. Um *socket* pode ser fechado em qualquer estado.

O estabelecimento de conexões do tipo SVC na rede é controlado pelas partes comunicantes. Um sistema final pode tanto aceitar uma conexão de entrada (*passive open*), como tentar estabelecer uma conexão de saída *(active open).*

A maneira usual de se estabelecer uma conexão é criar um socket SVC com socket(), especificar os requesitos de QoS (**setsockopt()**), preparar o descritor de endereço, requisitar o estabelecimento da conexão com um **connect()** bloqueante, bloquear até que a rede aceite ou rejeite a conexão, trocar dados, e eventualmente fechar o *socket* com a chamada **close()**.

O ponto terminal local pode ser associado a um endereço específico com a chamada bind(). Se na conexão um connect() não bloqueante for utilizado, o processo continua a executar, enquanto a conexão é estabelecida, e pode também decidir fechar o *socket* prematuramente.

Em uma abertura passiva, um *socket é* primeiramente criado com a chamada socket() e os parâmetros de QoS são determinados com **setsockopt()**. Então um descritor de endereço *é* configurado para especificar o tipo de conexões que devem ser atribuídas a este *socket,* e o

```
    return -1;
}
else
  options = (char *)NULL;

addr_req.addressType=AF_ATM_PVC;
addr_req.vpi = vpi;
addr_req.vci = vci;

memset(&call_req, 0, sizeof(call_req));
call_req.addr.len = sizeof(addr_req);
call_req.addr.buf = (char *) &addr_req;

if ((options != (char *) NULL) && (optlen > 0)) {
    call_req.opt.len = optlen;
    call_req.opt.buf = options;
}

if (t_connect(fd, &call_req, (struct t_call *) NULL) < 0) {
  t_error("t_connect");
  return -1;
}
```

Uma conexão de transporte (PVC ou SVC) pode ser liberada a qualquer momento através de uma chamada a **t_snddis().** O usuário pode então cancelar a inicialização do ponto final de transporte através do fechamento do descritor, ou especificar um novo endereço local (após ter desassociado o antigo endereço) ou ainda simplesmente reutilizar o antigo endereço e estabelecer uma nova conexão de transporte.

Maiores informações sobre a especificação XTI podem ser obtidos em: http://www.opengroup.org/public/pubs/catalog/c523.htm.

Na versão disponibilizada com a distribuição corrente do *driver* (5.0.2), a API oferece suporte a tráfegos do tipo UBR, CBR, VBR e ABR, sendo o último apenas sobre PVC's. As camadas de adaptação disponíveis na API são a AAL5 e a AAL0, também conhecida como AAL nula, significando ausência de camada de adaptação.

4.2.2 API ATM Linux

Os programas em ATM nativo para Linux foram implementados com a API desenvolvida por Werner Almesberger [apilinux], que pode ser obtida em http://icawww1.epfl.ch/linux-atm. A API é baseada em programação *socket*, sendo sua utilização semelhante à programação TCP/IP tradicional. A versão corrente da API é a 0.4. Uma vantagem significativa da implementação sobre Linux em relação à implementação sobre Solaris é o fato da sua API ser independente do fabricante da interface ATM, o que aumenta a versatilidade da ferramenta.

Nesta API, uma conexão típica atravessa quatro fases: preparação, estabelecimento, troca de dados e encerramento. Durante a fase de preparação, parâmetros são determinados *e* recursos locais gerais (descritores de *socket,* por exemplo) são alocados. Nenhum recurso de rede é alocado nesta fase. Durante a fase de estabelecimento, recursos locais de rede (identificadores de conexão, buffers, etc.) são alocados. A alocação de recursos na rede pode ser efetuada manualmente (PVC's) ou pode ser feita como parte da fase de estabelecimento (SVC's). Na fase de troca de dados, informações são trocadas sobre uma conexão previamente estabelecida. Ao encerrar uma conexão, os recursos alocados são liberados.

Há um conjunto de chamadas correspondente a cada fase de uma conexão. Chamadas efetuadas em fases inadequadas retornam erro.

Há algumas diferenças entre conexões PVC e SVC. Abrir um socket PVC significa associar um ponto final a uma conexão que normalmente já existe na rede. Socket's PVC são criados

modos de serviço e execução. No caso do provedor ATM, apenas o modo de serviço orientado a conexão é encontrado, sendo os modos de execução síncrono e assíncrono suportados.

Durante sua existência, uma conexão ATM XTI típica atravessa quatro fases distintas: inicialização, estabelecimento de conexão, transferência de dados e liberação de conexão.

O primeiro passo do processo de conexão em XTI diz respeito à criação de um ponto final de transporte. Um ponto final de transporte especifica um caminho de comunicação entre um usuário de transporte e um provedor de transporte específico. Quando um usuário cria um ponto final de transporte, um descritor é retornado servindo doravante como identificador único do ponto final de transporte aberto. A chamada t_open() é utilizada para criação de um ponto final de transporte (todas as chamadas XTI são precedidas de "t_"). O provedor de transporte desejado deve ser passado como parâmetro na chamada t_open(). No caso do ATM, há dois provedores disponíveis: um para PVC's e outro para SVC's.

Para conexões do tipo SVC, o passo seguinte à criação do ponto final de transporte é a associação de um endereço local ao descritor obtido. Isto é alcançado através da chamada **t_bind()**. O usuário originador (ativo) pode optar por não especificar um endereço local na chamada a t_bind(). Caso opte pela não especificação, o provedor selecionará um endereço *default*. O usuário terminador (passivo) deve ao menos especificar um seletor (último *byte* de um endereço ATM NSAP) para o endereço onde deseja ficar à espera por conexões, o restante do endereço sendo preenchido pelo provedor. Esta funcionalidade é utilizada na ferramenta.

Para PVC's, a chamada t_bind() serve apenas para mover ponto final de transporte do estado **T_IDLE** (inicial) para o estado **T_BOUND** (associado).

Para estabelecimento da conexão de transporte do tipo SVC, as chamadas **t_connect()** (modo ativo), ou **t_listen()** e **t_accept()** (modo passivo) são utilizadas. Quando realizadas em modo síncrono, as chamadas ficarão bloqueadas, até que uma resposta ou uma solicitação de conexão tenha sido recebida. Em modo assíncrono o controle é retornado ao usuário, imediatamente após a realização da chamada. Um evento ocorrerá no descritor associado, quando uma resposta a uma solicitação de conexão ou uma *indicação de solicitação de* conexão for recebida. O usuário pode monitorar a ocorrência de eventos através das chamadas poll() e **select()**.

A fase de conexão também se faz presente em PVC's. Quando tratamos de PVCs, é nesta fase que o descritor obtido com t_open() é associado a um par VPI/VCI e opções são configuradas (QoS, por exemplo). Em PVCs, a fase de conexão implica em uma chamada a t_connect() tanto do lado originador como do lado terminador. A chamada **t_connect()**, quando utilizada com PVC's, não gerará nenhuma indicação para o lado remoto, tendo assim significado local apenas (associação ao par VPI/VCI e determinação das opções desejadas).

Dentre as opções passíveis de configuração estão o tipo de conexão (ponto-a-ponto ou ponto-multiponto), a QoS desejada, parâmetros de tráfego, etc. Cada opção possui um nome e diz respeito a um nível. Em ATM, as opções se encontram ao nível **T_ATM_SIGNALLING**. As opções desejadas devem configuradas e armazenadas em um *buffer* contíguo na memória. O conteúdo desse buffer será posteriormente utilizado na chamada **t_connect()**.

O trecho de código a seguir foi extraído da implementação e ilustra parte do processo de abertura de uma conexão PVC em XTI:

```
if (pcr != 0) {
  options = construct_pvc_options(fd,pcr,&optlen);
  if (options == (char *) NULL)
```

Obs: Valores de E iguais ou inferiores ao retardo de ida e volta de um pacote não acarretam diferenças no resultado final. Se destino e porta não forem especificados será utilizado o endereço de *loopback* IP.

- *refletor_udp*
- *monitor_pvc* <VCI> <pcr> **<N>** <L> <E>
  Obs.: Somente tráfegos CBR e UBR são suportados no momento.
- *monitor_pvc2* <VCI> <pcr> <L> <E> <D> [arquivo].
- *sendwait_pvc* <VCI> <pcr> <N> <L> <E>
- *refletor_pvc* <VCI> <pcr>
- *monitor_tcp* <*destino*> <*porta*> <*N*> <*L*> <*E*>
- *refletor_tcp*

Exemplos de saída dos programas monitor e sendwait são mostrados na seção 5, que aborda alguns exemplos de testes de laboratório realizados com a ferramenta.

4 Implementação

Para implementação da ferramenta, além da programação baseada no conceito de *sockets* tradicional para BSD UNIX, foram utilizadas APFs disponíveis para programação em ATM nativo para os sistemas operacionais Solaris e Linux. Excetuando-se aqueles programas que fazem uso das APFs para ATM nativo, os restantes estão também disponíveis para FreeBSD.

4.1 Pilhas TCP/IP e UDP/IP

A ferramenta disponibiliza programas de monitoração para utilização sobre as pilhas TCP/IP e UDP/IP. A versão para TCP apresenta estatísticas referentes aos pacotes transmitidos em uma sessão. Convém lembrar que o TCP opera com recuperação de erros, de modo que pacotes perdidos nas camadas inferiores são sempre retransmitidos e recuperados pelo TCP. A nível de aplicação nunca teremos perdas. A versão para UDP, além das medidas de retardo e tempo entre pacotes, apresenta também a porcentagem de perda de pacotes, e deve ser usada sempre que a estatística de perda for necessária. Ambas versões são acompanhadas de seus respectivos programas refletores. É válido mencionar que estes programas podem também ser utilizados para testes sobre ATM, desde que um ambiente CLIP ou LANE esteja disponível. A existência do ATM fica transparente, nestes casos.

4.2 ATM Nativo

Os programas desenvolvidos em ATM nativo fazem uso de duas APFs distintas, uma para Linux e outra para Solaris. Como poderemos constatar a seguir, a forma geral de utilização das API's é bastante semelhante. Ambas, no entanto, apresentam algumas peculiaridades.

4.2.1 API XTI Fore

A API utilizada na implementação em ATM nativo para Solaris é disponibilizada pela FORE Systems Inc. como parte integrante do *driver* para seus adaptadores de rede. A API em questão é baseada na especificação XTI (X/Open Group Interface) [xti], que define uma interface de serviço de transporte independente com o intuito de permitir a comunicação de múltiplos usuários ao nível de transporte do modelo de referência OSI. A especificação descreve características de camada de transporte que são suportadas por uma ampla variedade de protocolos de transporte e apresenta ainda o conceito da camada de transporte como sendo composta por um ou mais provedores de transporte. Cada provedor pode oferecer diferentes

determina a geração de tráfego em intervalos fixos de tempo, sendo portanto do tipo determinístico (a nível de aplicação). O volume de tráfego gerado a cada transmissão é função da taxa solicitada pelo usuário. Dependendo da variante do programa, o usuário tem também acesso ao tamanho da unidade de transmissão a ser utilizada.

Os programas sendwait foram realizados com o intuito de permitir medida precisa de latência bastante pequena. Uma vez que nos programas monitores temos dois processos rodando simultaneamente, transmissor e coletor, há uma competição pelo processador da estação, gerando imprecisão. Essa competição poderá afetar as medições da ferramenta em função, principalmente, da capacidade de processamento da estação, do tamanho de pacote utilizado, e do intervalo de geração de pacotes, especificado pelo usuário. Para contornar o problema, o programa só envia um novo pacote após o recebimento do anterior ou na ocorrência de um time-out. Esta melhoria é mais significativa em ambiente de rede local, onde o atraso de propagação é desprezível ou muito pequeno e latências são apenas de alguns microssegundos.

3.1 Chamadas da Ferramenta Almadem

A ferramenta Almadem versão 1.1 utiliza os seguintes parâmetros em suas chamadas:

Parâmetro	Descrição
Destino	End IP válido (p/ gerador) ou estação refletora rodando refletor (p/ monitor).
W	Banda a ser *gerada* pela sessão em kbps.
L	Tamanho de pacote a ser utilizado em bytes.
D	Duração da sessão em Segundos (igual, maior ou menor que zero).
Porta	Valor inteiro fornecido na inicialização do programa refletor.
N	Número de pacotes a serem gerados.
L	Tamanho do pacote em bytes (valor mínimo é função da chamada)..
E	Intervalo entre transmissões consecutivas de pacotes, em microssegundos.
MF	Meio físico: ATM, 802.3, Ev2, F802.3 (Fast 802.3), FEv2 (Fast Ethernet v2)
Arquivo	Arquivo para log dos resultados (opcional)
VCI	Identificador do circuito virtual a ser utilizado
pcr	Taxa de pico em kbps (CBR), O (UBR)

Lista de comandos (todos possuem ajuda, se chamados com parâmetros incompletos):

- *gerador_udp* <destino> <W> <L> <D>
  Obs.: D <0, causa uma sessão infinita. D=0, transmite uma única PDU com W bytes.
- *gerador_udp2* <destino> <W> <D>
  Obs.: A cada 10 ms são geradas *i* PDUs de aplicação, sendo *i* calculado a partir da banda W e do intervalo de transmissão de 10 ms. O programa determina o tamanho da PDU, levando a uma precisão maior na quantidade de bytes enviada a cada intervalo, e resultando em uma banda efetiva mais próxima da solicitada. *gerador_udp2* tem por objetivo apresentar uma melhor precisão na banda gerada que *gerador_udp*.
- *monitor_udp* <destino> <porta> <N> <L> <E>
- *monitor_udp2* <destino> <porta> <L> <E> <D> <MF> [arquivo]
  Obs.: O parâmetro <MF> especifica o meio físico a ser utilizado nos testes, sendo utilizado para cálculo das taxas efetivas geradas a nível físico. Uma especificação incorreta desse parâmetro não causa qualquer dano aos dados computados, somente as taxas teóricas exibidas não deverão ser consideradas.
- *sendwait_udp* [destino] [porta] <N> <L> <E>.

do *heartbeat* do sistema, não tendo sido inferior a 10 ms nos testes realizados, tornando as opções inadequadas.

Diante do que foi dito, fica claro que a única alternativa que permite o alcance de uma precisão adequada ao propósito do programa é a espera ocupada. Nesse caso, o processo gerador entra em *loop* após a geração de um pacote, monitorando o relógio do sistema, até que o intervalo E tenha transcorrido, para então gerar o pacote seguinte. Para sessões de pequena duração, essa solução não apresenta maiores problemas. Entretanto, quando tratamos de sessões de maior duração, uma queda significativa no desempenho do processamento poderá vir a ser notada. De qualquer forma, é aconselhável que a estação utilizada para execução das ferramentas fique inteiramente dedicada aos testes durante sua realização.

3 Descrição Geral da Ferramenta

A ferramenta é composta basicamente de quatro programas principais, cada qual possuindo suas variantes (UDP, TCP, ATM, etc): Gerador, Monitor, Refletor e SendWait. Cada programa tem sua finalidade própria, podendo ou devendo ser utilizado de forma conjunta com algum outro. O uso dos programas refletores, por exemplo, está diretamente vinculado ao uso dos programas monitores, com exceção do ATM nativo, onde a reflexão das informações pode ser feita passivamente, através de configuração do próprio comutador ATM ou cabo externo entre interfaces, dispensando a utilização de um programa refletor.

Os programas monitores são constituídos por dois processos: um dedicado à geração de pacotes em instantes de tempo pré-determinados, outro dedicado à recepção dos pacotes previamente gerados e cômputo de suas estatísticas. Cada variante do programa possui seus parâmetros de entrada que, entre outras coisas, permitem a determinação da duração da sessão e do intervalo de geração de pacotes. O processo transmissor *é* o processo pai e dá origem ao processo coletor (filho), imediatamente antes de iniciar a transmissão de pacotes. A partir desse momento, o processo transmissor gerará um pacote contendo, em geral, número de seqüência e *timestamp* de transmissão a cada E µs, onde E é um parâmetro fornecido pelo usuário. O processo coletor fará a leitura dos pacotes refletidos, computando as estatísticas. Terminada a transmissão, o processo pai fica à espera do término do processo filho, que, após o recebimento do último pacote ou time-out, fará o processamento final das estatísticas coletadas. Os resultados finais são impressos na tela ou gravados em arquivo.

Cada variante do programa monitor, exceto a variante TCP, apresenta, dois tipos de chamada: *monitor* e *monitor2*. A primeira chamada é mais adequada a sessões de pequena duração, quando desejamos uma caracterização rápida dos parâmetros de latência do ambiente de rede sendo avaliado. A segunda chamada é otimizada para sessões de maior duração, ideal para situações onde se deseja determinar possíveis variações estatísticas no comportamento da rede, durante um período mais longo de observação.

Os programas refletores têm como única função a leitura e subsequente retransmissão das PDU's enviadas pelos programas monitores equivalentes, sem qualquer processamento extra de informação. Ao contrário dos programas monitores, os refletores são constituídos por apenas um processo, que realiza tanto leitura como escrita.

Os programas geradores têm por objetivo a geração de tráfego de fundo durante a realização dos experimentos. Estes programas permitem a geração de grandes volumes de tráfego (dependendo da capacidade da estação sendo utilizada), sendo capazes de congestionar ambientes de rede local como Fast Ethernet. Em nossos experimentos, uma estação Sun UltraSparc 170 Mhz foi capaz de gerar 120 Mbps. A implementação desses programas

interface de rede ATM, podendo este ser obtido diretamente no site da empresa http://www.fore.com . Os programas em ATM nativo para Linux foram implementados com a API desenvolvida por Werner Almesberger, que pode ser obtida em *http://icawww1.epfl.ch/linux-atm* sendo independente de fabricantes. Na versão corrente, os programas em ATM nativo não foram disponibilizados para o sistema operacional FreeBSD.

Os programas desenvolvidos para as pilhas UDP/IP e TCP/IP fazem uso de programação *socket* BSD padrão, não possuindo maiores requisitos quanto a sua utilização. Esses programas encontram-se disponíveis para os sistemas operacionais Solaris, Linux e FreeBSD. Todos os programas possuem ajuda na linha de comando, facilitando enormemente a especificação dos parâmetros necessários.

Na seção *2,* apresentamos uma discussão das alternativas de como alcançar melhor precisão na implementação das ferramentas e impacto no processamento local das estações. Alternativas de projeto são discutidas e analisadas. Na seção 3, apresentamos a descrição concisa dos programas implementados e linhas de chamada. Na seção 4, apresentamos detalhes da implementação dos programas. Na seção 5, apresentamos exemplos de utilização da ferramenta e, na seção 6, apresentamos as conclusões.

2 Precisão de Medidas

Para alcançar medidas estatísticas do tempo de retardo entre pacotes, a geração de pacotes é feita em intervalos fixos. Um problema crítico de implementação refere-se à geração correta dos instantes de transmissão e determinação precisa dos instantes de recepção dos pacotes. A interferência entre as execuções do código de transmissão e de recepção pode também gerar imprecisões. Estas questões são abordadas a seguir.

O intervalo entre gerações consecutivas de pacotes é definido pelo usuário através do parâmetro E de entrada dos programas. Na versão atual, a resolução é da ordem de µs.

Já que o processo responsável pelo envio de pacotes fica ocioso entre uma geração e outra (basicamente incrementa o contador de pacotes), a ação mais imediata seria simplesmente a colocação do processo para dormir com uma chamada à função *usleep().* A precisão da função *usleep(),* no entanto, é função direta do *heartbeat* do sistema operacional sendo utilizado. Todo sistema operacional mantém uma noção própria de tempo, através de interrupções geradas periodicamente por um *chip* específico. Esse esquema de interrupções é conhecido como *heartbeat.* Por exemplo, o *heartbeat* do Linux é de 10 ms para a plataforma i386 e de 1 ms para a DEC Alpha. Há *patches* para Linux [patch1, patch2] que permitem a reprogramação do *chip* gerador de interrupções, permitindo a obtenção de melhor granularidade. O tratamento de interrupções, em si, consome tempo de processamento da CPU, o que vem a impor um limite mínimo ao *heartbeat* do sistema. Mesmo usando esse recurso, não poderíamos simplesmente reduzir o *heartbeat* para a casa dos microssegundos, pois acarretaria uma queda acentuada de desempenho geral do sistema, tornando a operação inviável. Nos testes realizados, a melhor resolução alcançada com a função *usleepO* ficou em torno de 20 ms. Essa resolução é insuficiente aos propósitos da ferramenta.

Outras alternativas consideradas foram a interface de sinais Spec 1170, padrão do sistema UNIX, e a interface de sinais disponibilizada pela especificação POSIX.4. Ambas permitem que um processo seja acordado por um sinal específico a intervalos configuráveis. A interface Spec 1170 oferece precisão nominal em microssegundos, enquanto que a interface POSIX.4 oferece resolução de nanossegundos. Na prática, entretanto, a granularidade obtida é função

Estamos particularmente interessados em medir atrasos ida-e-volta de pacotes, bem como detectar possíveis variações estatísticas do retardo entre pacotes. Medidas de latência de comutadores são da ordem de poucos μs e o atraso pode também ser desta ordem, se o ambiente for de rede local ou metropolitana. Obter estatísticas fim-a-fim (em uma única direção) com este nível de precisão requer uma sincronização perfeita dos relógios dos equipamentos envolvidos, na transmissão e na recepção. Uma forma possível de atingir essa sincronização é o uso de uma interface especial de relógio baseada em GPS, que forneceria um padrão de tempo global único, com precisão. Este tipo de hardware não é comum e envolve custos adicionais. Entretanto, no cálculo de estatísticas ida-e-volta, podemos adotar o esquema monitor/refletor, onde o tráfego é gerado em uma estação, enviado até a estação refletora, refletido e retornado à estação origem. A comparação entre as marcações de tempo *(timestamps)* nos pacotes entre a transmissão e a recepção permite o cálculo do atraso ida-e-volta de cada pacote, bem como a medida do atraso entre pacotes. No caso do ATM, a estação refletora pode ser dispensada e a reflexão pode ser feita, passivamente, estabelecendo uma conexão virtual em loop, ou pelo uso de cabos externos, interconectando portas de transmissão e recepção. No caso de geração de tráfego, como usamos apenas uma estação transmissora, este problema não se aplica.

Seguindo este modelo monitor/refletor, outras ferramentas existem notadamente TTCP [ttcp] e NetPerf [netp]. Estas ferramentas não foram otimizadas para medidas de desempenho de alta precisão, e, em especial, a saída de estatísticas destas ferramentas não apresenta os resultados com intervalos de confiança para os valores medidos. Em ambientes de alta velocidade, onde a precisão dos resultados é crítica e a ordem de grandeza das medidas é muito pequena, a apresentação de resultados com intervalo de confiança é fundamental. A ferramenta Almadem apresenta o intervalo de confiança de 90% para todas as estatísticas pertinentes e segue o modelo monitor/refletor.

Com relação à geração de tráfego, as ferramentas existentes permitem a especificação de uma taxa a ser gerada ao nível de aplicação, mas não fornecem a taxa efetivamente gerada no meio físico, que, em geral, é o que desejamos determinar. Por exemplo, se contratamos ao serviço ATM de uma operadora um caminho virtual permanente com banda constante (PVP CBR) e igual a 20 Mbps, e sabendo que a operadora não admite tráfego além do contratado, então, se queremos comprovar a garantia de banda do serviço contratado, temos que enviar no meio físico o equivalente a 20 Mbps e verificar descarte zero de células. Se especificarmos simplesmente 20 Mbps ao nível de aplicação, a consideração de todos os overheads envolvidos irá gerar uma taxa acima de 20 Mbps no meio físico e descartes ocorrerão. Redes locais de alto desempenho trabalhando com Ethernet comutado full duplex, permitem o cálculo preciso do overhead no meio físico, pela ausência de colisões. Evidentemente, em redes compartilhadas como Ethernet normal, o tráfego gerado no meio físico por uma determinada aplicação é acrescido de colisões, fazendo com que a taxa efetivamente transmitida seja maior. Neste caso, somente podemos determinar o nível mínimo da taxa no meio físico. A ferramenta Almadem, operando com TCP/IP, UDP/IP ou nativamente sobre ATM, inclui o cálculo das taxas efetivas de transmissão e recepção no meio físico, o que facilita a programação e configuração de testes.

Os programas geradores de tráfego utilizam a pilha UDP/IP. Nos programas de monitoração, o usuário pode optar entre pilhas UDP/IP, TCP/IP ou ATM nativo para execução. No caso específico do ATM nativo para o sistema operacional Solaris, a API de programação utilizada foi a API/XTI [xti] disponibilizada pela FORE Systems Inc., não sendo, portanto, o programa compatível com outras APIs ATM. A API citada é disponibilizada juntamente com o driver da

[ttcp] http://www.ccci.com/tools/ttcp/.
[xti] The Open Group, *Networking Services (XNS) Issue 5*, Copyright 1997, .http://www.opengroup.org/public/pubs/catalog/c523.htm
[xtifore] http://www.fore.com/support/manuals/lib/techpubs/docs/manuals/adapters/solaris52/0450.htm e ch091.htm

Apêndice

Cálculo de Taxa Efetiva e Taxa Teórica da Aplicação Monitor

Seja to, a taxa teórica a ser medida pela ferramenta (em bps), caso a transmissão ocorresse em um tempo infinito e sem perdas. Seja *tm*, a taxa gerada para a camada física (em bps), nestas mesmas condições. O tráfego a ser gerado é caracterizado pelo parâmetro L (em bytes), número de bytes da PDU, e pelo parâmetro E (em μs), que especifica o intervalo de envio entre transmissões consecutivas.

A) Transmissões em ATM

Vamos analisar o cálculo de to e *tm* para o programa monitor_pvc, que opera nativamente sobre PVC multiplexados e AAL5. Os overheads envolvidos são:

- AAL5 - 8 bytes
- PAD AAL5 - variável, entre O e 47 bytes (completar múltiplo de 48 bytes)
- LLC/SNAP - 8 bytes (RFC-1483, identifica o protocolo de nível superior no VC)

O número N de células enviadas pode ser calculado como: N = (L+16+PAD)/48. Então, to = L/E (em Mbps) e *tm* = (N*53*8)/E (em Mbps) = (N*8)/E (em células/s).

Para transmissões usando IP sobre ATM, os overheads de nível superior (cabeçalhos) podem ser identificados como: UDP - 8 bytes, TCP - 20 bytes, e IP - 20 bytes. Usando SVC ou PVC, o conteúdo a ser repassado para a AAL5 será um pacote IP, de tamanho L+28 (pilha UDP/IP) ou L+40 (pilha TCP/IP). Para o cálculo de to e *tm* podemos usar o mesmo raciocínio desenvolvido para a transmissão nativa em ATM, apenas substituindo naquelas fórmulas L pelo tamanho do pacote IP correspondente, L + 28 ou L + 40.

B) Transmissões usando pilhas UDP/IP ou TCP/IP em Ethernet

A taxa to será sempre L/E (em Mbps). No campo de informação do quadro Ethernet, os protocolos superiores (IP, UDP/TCP, aplicação) transmitirão (L+28) bytes, para UDP, ou (L+40) bytes, para TCP. O quadro MAC Ethernet v2 tem o seguinte overhead (em bytes): 8 (preâmbulo + marca) + 12 (endereços) + 2 (tipo) + 4 (CRC)= 26 bytes, no total. O quadro MAC IEEE 802.3 tem 8 bytes adicionais de overhead LLC/SNAP na área do campo de informação (limitando a 1492 o número máximo de bytes de protocolos superiores).

A ferramenta Almadem limita o tamanho de L para estas pilhas em 1472 bytes para UDP e 1460 bytes para TCP, permitindo a transmissão de uma PDU em apenas um quadro Ethernet. Temos então a tabela:

	Tm (Mbps)	
Protocolo	Pilha UDP/IP	Pilha TCP/IP
Ethernet v2	(L+54)/E	(L+66)/E
IEE802.3	(L+62)/E	(L+74)/E

No caso de Fast Ethernet, a comunicação é síncrona e não temos o overhead de preâmbulo, mantendo o restante inalterado.