UNIVERSIDADE SALVADOR – UNIFACS

DEPARTAMENTO DE ENGENHARIA E ARQUITETURA (DEAR)

CURSO DE ENGENHARIA ELÉTRICA

Ênfase em Eletrônica

A ferramenta de Simulação NS (Network Simulator) – Estudos de Caso.

Aluno: Rafael Gonçalves Bezerra de Araújo

Professor Orientador: Sergio de Figueirêdo Brito

SALVADOR – BA

2003

UNIVERSIDADE SALVADOR – UNIFACS

DEPARTAMENTO DE ENGENHARIA E ARQUITETURA (DEAR)

CURSO DE ENGENHARIA ELÉTRICA

Ênfase em Eletrônica

A ferramenta de Simulação NS (Network Simulator) – Estudos de Caso.

"Escolha um trabalho que você ame e não terá que trabalhar um único dia de sua vida."

(Confúcio – Filósofo Chinês)

SALVADOR – BA

2003

ÍNDICE

Índice de Figuras

Resumo

O trabalho tem como objetivo estudar o simulador de redes de computadores Network Simulator – NS, fazendo um detalhamento de suas características e funcionalidades, através da implementação de modelos de sistemas de Redes de Computadores.

A teoria do modelo de fila M/M/1, o protocolo de comunicação de redes ethernet cdma/cd e o tipo de controle de congestionamento do protocolo TCP “stop and wait” são empregados nos modelos descritos através dos scripts no decorrer do presente trabalho. As características e funcionalidades do Network Simulator (NS), pertinentes aos modelos descritos, serão utilizadas e detalhadas para melhor entendimento da ferramenta.

Introdução

Uma Rede de Computadores é basicamente um conjunto de dispositivos ligados entre si de forma a possibilitar o armazenamento, recuperação, e compartilhamento de informações e recursos pelos seus utilizadores [KUROSE 2003]. É um conjunto de pontos ou nós interligados entre si e a dispositivos da rede por meios de comunicação, os dispositivos envolvidos são computadores, servidores, impressoras, dispositivos de armazenamento de dados, entre outros.

As Redes de Computadores trouxeram novas facilidades de processamento de informação, permitindo a utilização das potencialidades de diversos equipamentos, assim como das capacidades dos seus utilizadores independentemente da sua localização geográfica. As Redes de Computadores permitem o processamento de informação de uma forma mais rápida e econômica que anteriormente. É possível reduzir o investimento em hardware, pois o compartilhamento de recursos possibilita ter menos e melhores equipamentos, desta forma o tempo necessário à gestão e manutenção do sistema também é reduzido. Uma rede bem concebida permite controlar o acesso aos seus recursos, assim é possível definir níveis de acesso para os diversos recursos. As redes podem classificar-se quanto à topologia de organização que apresentam, podendo ter uma topologia em barramento, anel, ou estrela. As Redes de Computadores são também caracterizadas pela tecnologia que utilizam na transmissão física dos dados, podendo utilizar a tecnologia Ethernet, FDDI, TokenRing, etc. Pode-se ainda caracterizar uma rede pelo tipo de dados que transporta (voz, dados, ou ambos), daí as diferentes ligações (fibra óptica, cabo coaxial, fio de cobre).

A diversidade e complexidade da tecnologia das Redes de Computadores e seu constante desenvolvimento conduzem pesquisadores e administradores de sistemas, freqüentemente, a depararem-se com propostas para melhoria de processos vigentes ou para soluções de problemas. Estas propostas precisam ser testadas quanto a sua eficácia e abrangência. A avaliação de desempenho de sistemas [JAIN 1991] proporciona técnicas para a abordagem dessas questões, sendo que, a modelagem e simulação de sistemas é uma delas.

O sistema quando tratado pelo método da simulação, permite a confecção de modelos mais complexos e soluções com menor desenvolvimento matemático, empregando-se a computação para a realização das iterações numéricas necessárias. Simulações permitem controlar as condições experimentais, economizando tempo e dinheiro. Usar simulação permite que se analise sistemas com comportamento dinâmico através do tempo.

Construir um modelo ajuda a entender a estrutura do sistema e a identificar relações entre os seus componentes. Em um modelo dinâmico pode-se observar esses componentes e suas relações e ainda o seu comportamento sob eventos aleatórios. A Simulação de eventos discretos possibilita o uso de valores aleatórios que representem o desempenho do sistema, condições do ambiente, eventos inesperados etc. Ao mesmo tempo, esses valores são próximos aos reais na medida em que são usados métodos estatísticos para coleta de dados, representação, e análise.

O Network Simulator [NS 2003] é um simulador de Redes de Computadores amplamente utilizado pelas comunidades cientificas espalhadas por universidades em todo mundo. O presente trabalho tem como objetivo documentar a utilização do simulador de Redes de Computadores Network Simulator – NS, através de exemplos descritos ao longo do trabalho.

O Simulador Network Simulator (NS)

O Network Simulator – NS [NS 2003] é um software de simulação de redes que permite desde a simulação de sistemas de redes de filas simples (M/M/1) até sistemas de Redes de Computadores complexas. É um simulador de eventos discretos, direcionado para pesquisas em redes de computadores. Ele trabalha na simulação de protocolos como o TCP e variantes (Tahoe, Reno, New Reno, Vegas, etc.), multicast, redes com ou sem fio (*wireless*), roteamento de pacotes, satélites entre outras coisas. Implementa filas de roteamento tipo droptail, Diffserv RED, fair queueing (FQ), stochastic fair queueing (SFQ), class-based queueing (CBQ), e vários tipos de geradores de tráfego. Possui algumas ferramentas matemáticas como gerador de números aleatórios e integrais para cálculos estatísticos.

A primeira versão do NS surgiu em 1989, versão oriunda do software REAL Network Simulator. A partir de 1995 este software passou a ter o apoio do DARPA através do projeto VINT na LBL, Xerox PARC, UCB, e USC/ISI. Atualmente o desenvolvimento do NS é apoiado pelo DARPA com o SAMAN e pelo NSF com o CONSER, ambos em colaboração com outros pesquisadores incluindo o ACIRI. O NS também inclui contribuições substanciais de outros pesquisadores, incluindo códigos de wireless do UCB Daedelus e dos projetos do CMU Monarch e da Sun Microsystems.

O Network Simulator se encontra atualmente em sua versão 2. Este é um software desenvolvido na Universidade de Berkeley em C++ e OTcl (linguagem de script TCL com extensão orientada a objeto) [OTCL 2003], sendo um software orientado a objeto. No *apendice A* pode-se obter informações sobre a instalação do

Network Simulator (NS), o texto é parte do trabalho publicado no SEPA (Seminário Estudantil de Produção Acadêmica) da UNIFACS [ARAÚJO 2002].

O NS é um simulador de redes dirigido a eventos que simula vários tipos de redes IP. Ele pode trabalhar simulando protocolos de rede como o TCP e o UDP, comportamento de tráfego em FTP, Telnet, Web, CBR e VBR, mecanismos de gerenciamento de filas de roteadores como DropTail, RED e CBQ, algoritmos de roteamento como o Dijkstra e outros. O NS também implementa multicasting e alguns dos protocolos da camada MAC para simulação de LANs. O projeto NS é agora parte do projeto VINT que é um projeto de pesquisa financiado pelo DARPA cujo objetivo é construir um simulador de rede que permita o estudo de protocolos de rede atuais e futuros. Em [VINT 2003] e [SAMAN 2003] pode-se obter mais informações sobre o VINT e o SAMAN respectivamente.

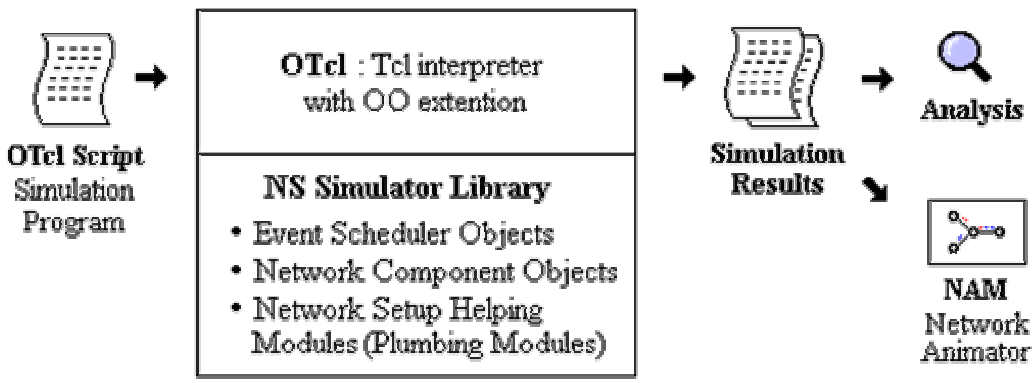

figura 1: Visão simplificada do usuário do NS

A figura 1 [CHUNG 2003], demonstra uma visão simplificada do usuário, o NS é um interpretador de script Tcl orientado a objeto (OTcl). Esta biblioteca contém objetos de escalonamento de eventos, objetos de componentes de rede e módulos de ajuda de configuração de rede. Em outras palavras, para utilizar o NS, se programa em OTcl. Para configurar e rodar um simulador de rede, o usuário deve escrever um script

OTcl que inicia um escalonador de eventos, configura a topologia da rede utilizando os objetos de rede e funções das bibliotecas, e informam as fontes de tráfego quando iniciar e parar a transmissão de pacotes, através do escalonador de eventos. Quando um usuário quer construir um novo objeto de rede, este pode facilmente fazê-lo construindo um novo objeto ou utilizando um objeto de uma biblioteca, e ligando o caminho de dados pelo objeto.

O NS não é apenas escrito em OTcl, mas também em C++. Os objetos compilados são disponibilizados para o interpretador de OTcl por uma ligação OTcl que cria uma correspondência do objeto OTcl para cada um dos objetos C++. Também fazem com que as funções de controle e as variáveis de configuração especificadas pelo objeto C++ ajam como funções de membros e variáveis de membros de objetos OTcl correspondentes. Desta forma, os controles de objetos C++ são dados pelo OTcl. Isto também é possível para adicionar funções de membros e variáveis para o C++ ligados aos objetos OTcl. O objeto em C++ que não é necessário ser controlado na simulação ou internamente utilizado por um outro objeto, não precisa ser ligado ao OTcl.

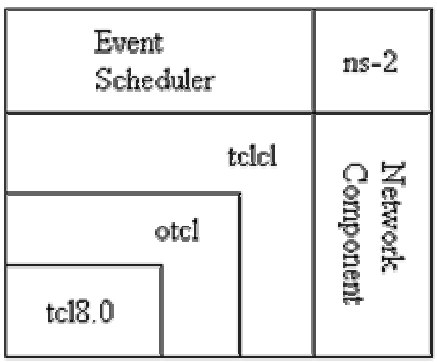

figura 2: Arquitetura geral do NS

A figura 2 [CHUNG 2003], mostra a arquitetura geral do NS. Nesta figura o usuário do NS está situado no canto mais à esquerda e abaixo, planejando e rodando simulações em Tcl utilizando os objetos de simulação das bibliotecas OTcl. Os

escalonadores de evento e a maioria das componentes de rede são implementados em C++ e disponíveis em OTcl pela ligação que é implementada utilizando tclcl.

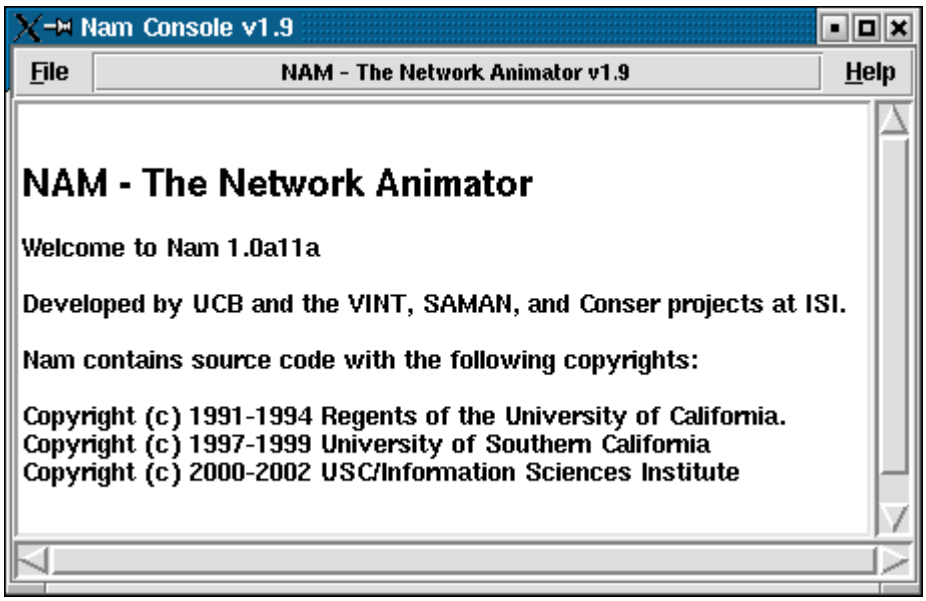

figura 3: Network Animator (NAM)

Quando uma simulação é feita, o NS produz um ou mais arquivos de saída baseados em texto que contém dados da simulação detalhados. Os dados podem ser utilizados para análise de simulações ou como entrada para uma ferramenta de simulação gráfica chamada *Network Animator* (NAM), figura 3, que é desenvolvido com uma parte do projeto VINT. O NAM tem uma interface de usuário gráfica bastante amigável e tem um controlador de velocidade. Pode ainda apresentar, graficamente, informações presentes como vazão e números de pacotes emitidos para cada link. Contudo as informações gráficas não podem ser utilizadas para análise profundas de simulações. O *Network Animator* (NAM) pode ser utilizado como gerador de scripts, através de uma interface gráfica, figura 4. Tipos básicos de fila, geradores de trafego e links podem ser utilizados na geração dos scripts, através da NAM. Maiores informações sobre o *Network Animator* em [NAM 2003].

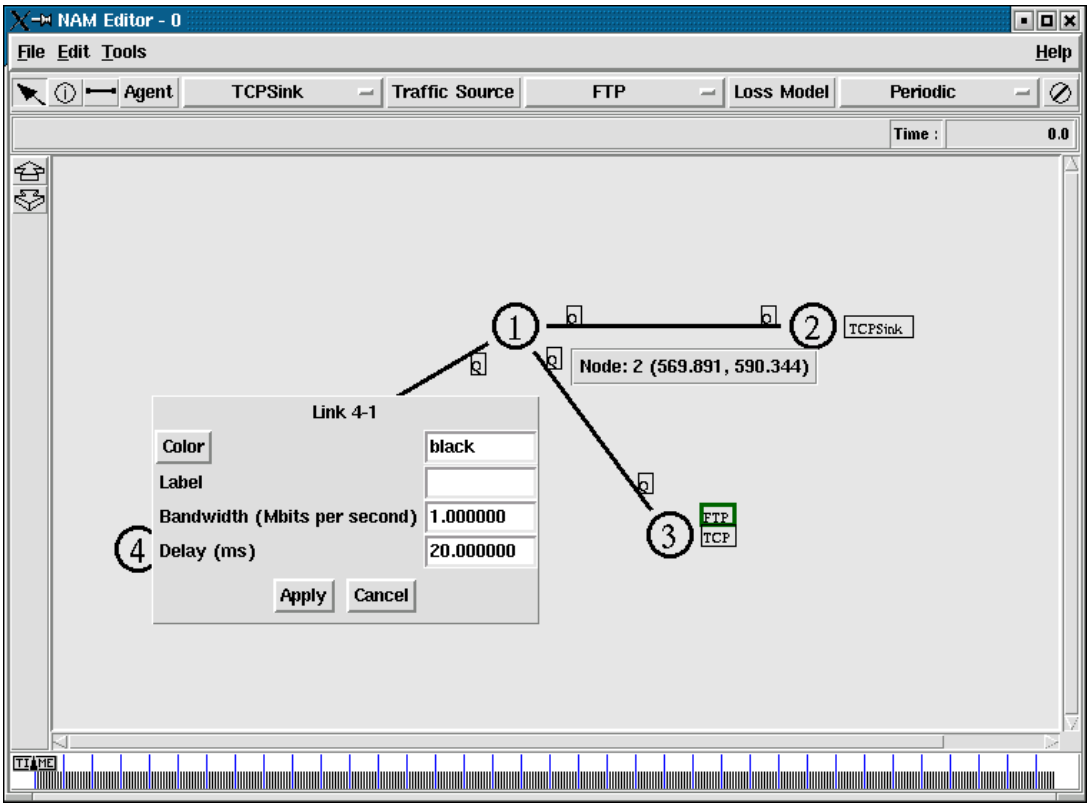

figura 4: Gerador de scripts (NAM)

Para essa finalidade uma ferramenta, denominada *TraceGraph*, pode ser utilizada. Ela utiliza-se dos arquivos de saídas, em formato texto, para obtenção de inúmeras informações, tais como atraso, jitter, tempo de processamento, numero de nos intermediários e estatísticas. O *tracegraph* é um trabalho de graduação em ciências da computação e matemática, de um estudante polonês, na universidade de Wroclaw (Wroclaw University of Technology) [TRACEGRAPH 2002].

Linguagen TCL

As linguagens tradicionais surgiram no final da década de 1950 como uma opção para a programação assembler. Estas linguagens conseguiram otimizar o trabalho dos programadores, pois abstraíam o processo de alocação de registradores e de

memória, as chamadas a procedimentos e as iterações e as condições com comandos tipo while e if. O conceito de tipagem surgiu na mesma época. Este conceito trouxe a necessidade de que todas as variáveis de um programa deveriam ser declaradas e a elas deveria se atribuir um tipo que concordasse com a função desta variável no programa. Também surgiu a regra de que cada variável só pode ser operada por um operador específico ao seu tipo. As conseqüências destas inovações foi a diminuição da complexidade das soluções, a diminuição da quantidade de código necessário para implementação dos programas e maior documentação dos códigos. Por outro lado, também houve uma diminuição do desempenho dos programas, pois foram adotadas soluções genéricas para problemas de baixo nível (endereçamento,...) as quais, muitas vezes, não são as soluções mais adequadas. A figura 5 [SCRIPTING 2003], faz uma comparação gráfica entre a linguagem assembler e diversas linguagens de programação.

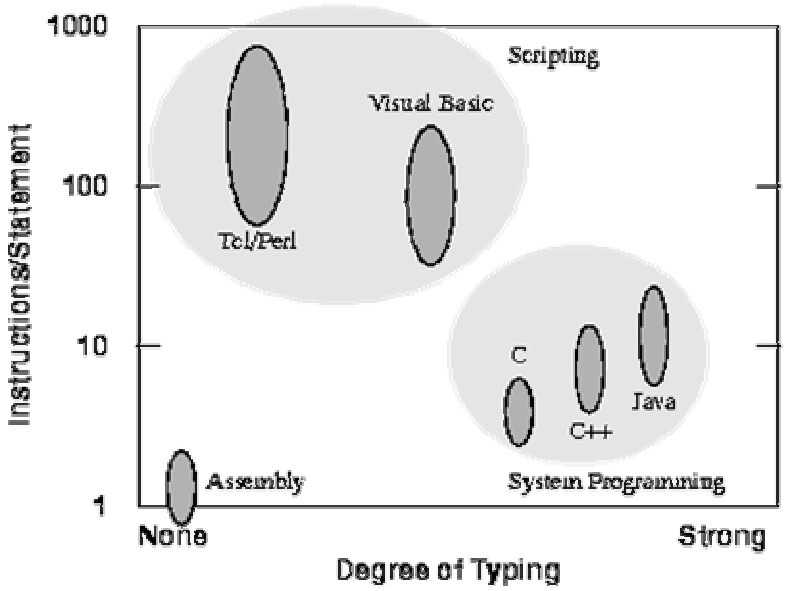

figura 5: Comparação entre diversas linguagens de programação

A figura 5 [SCRIPTING 2003], demonstra uma comparação entre diversas linguagens de programação baseado no seu nível (linguagens de alto nível executam mais instruções de maquina por indicação) e seu grau de “tipagem”. Linguagens de

programação, como C, tendem a ser altamente “tipadas” e possuir médio nível de abstração (5 – 10 Instruções / Indicação). Linguagens de script, como Tcl, tendem a ser fracamente “tipadas” e possuir alto nível de abstração (100 – 1000 Instruções / Indicação).

As linguagens de Script [SCRIPTING 2003] representam uma alternativa para sistemas onde o objetivo é o de aglutinar módulos computacionais já construídos criando sistemas muito complexos em um nível de abstração muito mais alto do que as linguagens convencionais. Estas linguagens consideram a existência de um conjunto de componentes suficientemente completos, escritos em linguagens de alto nível tradicionais, que lhes permita criar sistemas simplesmente aglutinando estes módulos em um sistema maior. Estas linguagens não são feitas para se desenvolver programas partindo do zero e sim para arranjar componentes.

As linguagens de Script não costumam ser “tipadas”, ou seja, uma variável pode assumir o valor de uma string em um momento, um inteiro no momento seguinte e ainda uma função ou alguma estrutura de dados mais complexa. A verificação da validação de um tipo é feita em tempo de execução e as variáveis, normalmente, são representadas por strings de tamanho variável. Esta filosofia faz com que os dados possam ser trocados entre os componentes dinamicamente, facilitando a filosofia do aglutinamento de componentes, já que abstrai boa parte do processo de normalização de dados entre os componentes. Uma grande característica das linguagens de script é o fato de estas serem interpretadas enquanto as linguagens tradicionais são normalmente compiladas. Em 0 pode-se obter mais informações sobre as linguagens de scripts.

A linguagem Tcl/Tk [OTCL 2003] foi desenvolvida pelo Dr. John K. Ousterhout e sua equipe da Universidade da Califórnia, sendo posteriormente mantida pela Sun Microsystems e pela Scriptics.

A Tcl foi desenvolvida para ser essencialmente uma linguagem de script usada, sobretudo, como uma ferramenta no desenvolvimento de comandos para o ambiente UNIX. Contudo acabou mostrando-se excelente, também, para o desenvolvimento de aplicações gráficas avançadas e scripts CGI. Ela possui uma sintaxe muito simples, tornando-a ideal para aglutinar componentes de maneira simples.

A extensão Tk é o Tool Kit gráfico de Tcl e lhe fornece todo o suporte para o rápido desenvolvimento de complexas aplicações gráficas, independentes de plataforma.

Tcl é uma linguagem poderosa, aliando um eficiente compilador a uma sintaxe extremamente simples. O compilador Tcl é tão eficiente, que há pouca diferença, em velocidade, entre um programa Tcl interpretado e sua versão compilada. A linguagem Tcl possui código-fonte aberto e é distribuída gratuitamente sob licença BSD.

Modelos

Modelos serão utilizados para demonstrar a utilização do NS (Network Simulator). Os scripts foram extraídos da Internet e serão explicados no decorrer do trabalho. A descrição destes scripts proporciona a criação de uma fonte de pesquisa sobre o NS e suas funcionalidades. Os modelos serão utilizados para explicar de forma didática o uso do simulador.

O modelo de fila M/M/1, o protocolo de comunicação de redes ethernet cdma/cd e o tipo de controle de congestionamento do protocolo TCP “stop and wait” serão os modelos descritos através dos scripts no decorrer do presente trabalho.

O modelo M/M/1

Um modelo de fila do tipo M/M/1 pode ser utilizado para modelar sistemas de único processamento ou dispositivos individuais em sistemas computacionais, para, por exemplo, dimensionar o processador, memória, disco, ou ainda, verificar a utilização do sistema. Pode-se exemplificar sistemas do tipo trafego rodoviário, barbeiro ou atendimento numa agencia bancaria [JAIN 1991].

O tempo de interchegada e o tempo de serviço são distribuídos exponencialmente e existe apenas um servidor. Não existe limitação para o tamanho da fila e sua disciplina é do tipo primeiro que entra, primeiro que sai. Para analisar este tipo de fila, precisa-se saber apenas dois parâmetros, o tempo médio de interchegada λ e o tempo médio de serviço μ, utilizando-se da terminologia de [JAIN 1991].

Como indicadores de Qualidade de Serviço (QoS) pode-se obter varias estatísticas, tais como: o tamanho médio da fila, vazão, tempo médio de serviço e a utilização do sistema.

figura 6: Visão da topologia de modelo MM1

A figura 6 demonstra a topologia de um modelo M/M/1. Onde ***FT*** é a fonte geradora de trafego, ***FL*** é a fila que será servida pelo servidor ***SV*** e ***SORV*** é o sorvedouro, destino dos pacotes servidos pelo servidor ***SV***.

O script M/M/1

Este script foi retirado de [NS BEGINNERS 2003]. Nas tabelas, de cor cinza, o código do script é mostrado, seguido de sua explicação. O código de cada script, em sua integra, poderá ser visualizado no anexo A.

O primeiro passo é a criação do objeto simulador, que é feita da seguinte maneira:

```
set ns [new Simulator]
```

Após a criação do objeto simulador, abre-se os arquivos *tracing* para posterior análise:

```
set tf [open out.tr w]
$ns trace-all $tf
```

O arquivo out.tr conterá os registros dos eventos de maneira mais completa, permitindo análises profundas da simulação.

No modelo proposto foram estipulados o valor de *lambda* e *mu*, que correspondem às taxas médias de chegada e serviço de clientes, respectivamente. Estes valores serão utilizados nos cálculos durante a simulação.

```
set lambda 30.0
set mu 33.0
```

A topologia da rede é criada da seguinte maneira:

figura 7: topologia inicialmente criada pelo script

```
set n1 [$ns node]
set n2 [$ns node]
set link [$ns simplex-link $n1 $n2 100kb 0ms DropTail]
$ns queue-limit $n1 $n2 100000
```

Nas duas primeiras linhas do código acima são criados os nós. Configura-se então a ligação entre estes nós, com link do tipo simplex, ou seja, existindo apenas um sentido de comunicação. Largura de banda de 100 Kbps com atraso de propagação nulo. A fila do tipo DropTail implementa o modelo FIFO (First in First out) e descarte de pacotes a partir do estouro do tamanho da fila, que por padrão no NS pode possuir qualquer valor. O tamanho da fila foi estipulado com um valor elevado, a última linha do código acima provê os artifícios necessários, configurando um limite para o tamanho máximo da fila entre os nós participantes da comunicação, neste caso os nós 1 e 2. Em [NS 2003] pode-se obter maiores informações sobre a criação e configuração de topologias.

No código abaixo são criados os tempos de interchegada e o tamanho dos pacotes randomicamente, através das variáveis *InterArrivalTime* e *pktSize*, respectivamente. Para os cálculos do tempo de interchegada e do tamanho dos pacotes, utiliza-se a classe OTcl *RandomVariable*. Esta classe provê a geração de números randômicos de acordo com uma infinidade de distribuições, referindo-se a um valor médio. Para o modelo *MM1* a distribuição utilizada é a exponencial. Maiores informações sobre o gerador de números randômicos e distribuição exponencial em [NS 2003] e [JAIN 1991].

```
set InterArrivalTime [new RandomVariable/Exponential]
$InterArrivalTime set avg_ [expr 1/$lambda]
set pktSize [new RandomVariable/Exponential]
$pktSize set avg_ [expr 100000.0/(8*$mu)]
```

Um agente do tipo UDP é criado e anexado ao nó 1. No NS um agente UDP aceita dados em pedaços variáveis ou segmentos de dados da aplicação. O tamanho máximo padrão do pacote (MMS) UDP é de 1000 bytes.

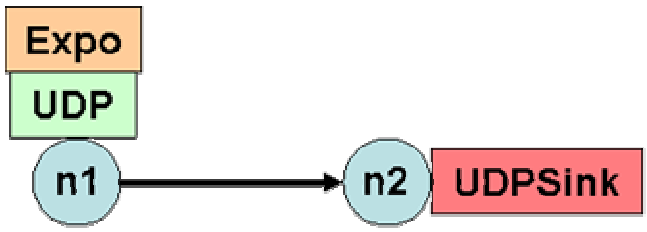

figura 8: topologia completa do script M/M/1

```
set src [new Agent/UDP]
$ns attach-agent $n1 $src
```

Para posterior análise a monitoração da fila é definida, com a criação de outro arquivo trace, *qm.out.*

```
set qmon [$ns monitor-queue $n1 $n2 [open qm.out w] 0.1]
$link queue-sample-timeout
```

O código acima descreve que o monitor irá escrever o número de pacotes que chegam na fila, através da função queue-sample-timeout, no arquivo trace *qm.out*, para posterior análise. Esta analise pode ser feita através de softwares como o *TraceGraph* [TRACEGRAPH 2002] ou mesmo o *Xgraph* que acompanha o pacote *ns-alinone* [NS 2003].

Na *procedure* abaixo é definido o processo de finalização da simulação, chamado pelo escalonador de eventos. Os arquivos de trace são encerrados.

```
proc finish {} {
    global ns tf
    $ns flush-trace
    close $tf
    exit 0
}
```

Na *procedure* abaixo é definido o processo de envio e geração do tamanho dos pacotes, através do agente UDP. Os procedimentos definem que o tempo de envio e o tamanho dos pacotes, dar-se-á de forma aleatória, de acordo com os cálculos anteriormente efetuados.

```
proc sendpacket {} {
   global ns src InterArrivalTime pktSize
   set time [$ns now]
   $ns at [expr $time + [$InterArrivalTime value]] "sendpacket"
   set bytes [expr round ([$pktSize value])]
   $src send $bytes
}
```

No código abaixo o sorvedouro é criado e anexado ao nó 2, figura 8. A conexão entre os agentes anexados aos nós 1 e 2 também é definida. Em seguida a programação do escalonador de eventos é criada, indicando o tempo de inicio e termino da simulação.

```
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $src $sink
$ns at 0.0001 "sendpacket"
$ns at 1000.0 "finish"
```

A última linha do script, finalmente, inicia a execução do simulador.

```
$ns run
```

Este script não possui implementação de um arquivo trace para ser utilizado pelo *Network Animator* (NAM). As estatísticas desta simulação podem ser obtidas através das ferramentas *Tracegraph* e *Xgraph*.

Resultados do script MM1

Após a simulação o arquivo *trace*, out.tr, gerado foi analisado com o auxilio do tracegraph [TRACEGRAPH 2002]. A figura 9 demonstra a tela principal do tracegraph, utilizado para analisar o script M/M/1.

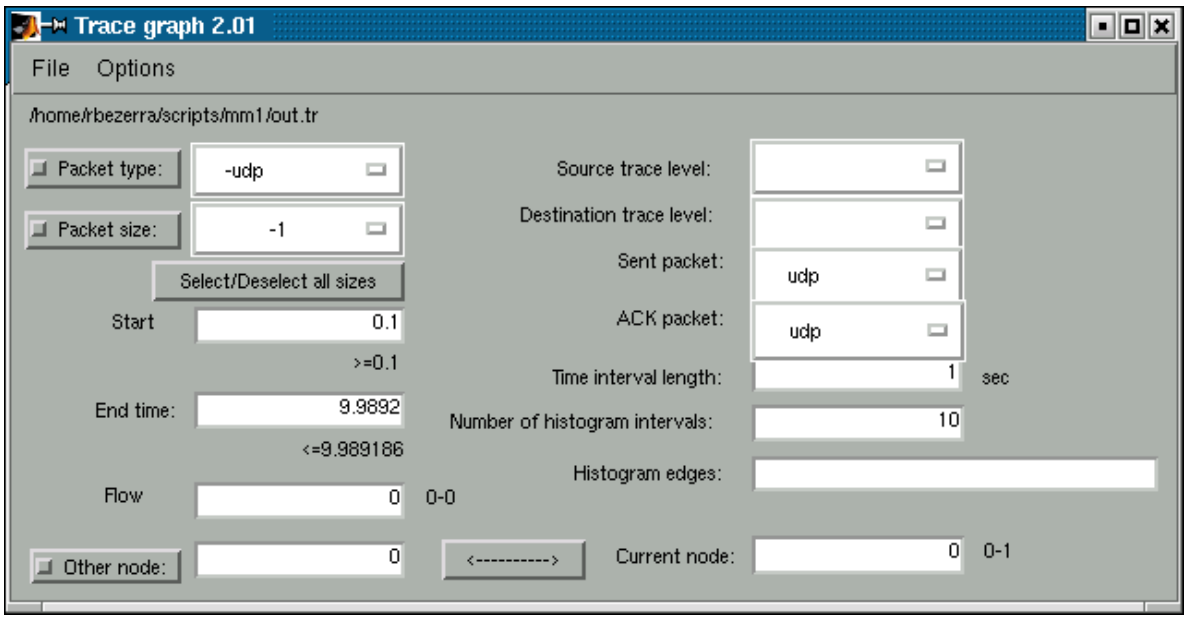

figura 9: Tela principal do TraceGraph para o script M/M/1

Utilizando o TraceGraph alguns dados da simulação puderam ser coletados, como mostra a figura 10. Simulando o script M/M/1, com um tempo de simulação igual a 10, dados como: numero de pacotes gerados, pacotes transmitidos, pacotes perdidos e descartados, tamanho máximo e mínimo do pacote gerado, numero de bytes enviados, entre outros, puderam ser observados. A Tabela 1 demonstra alguns dados extraídos da simulação do script M/M/1.

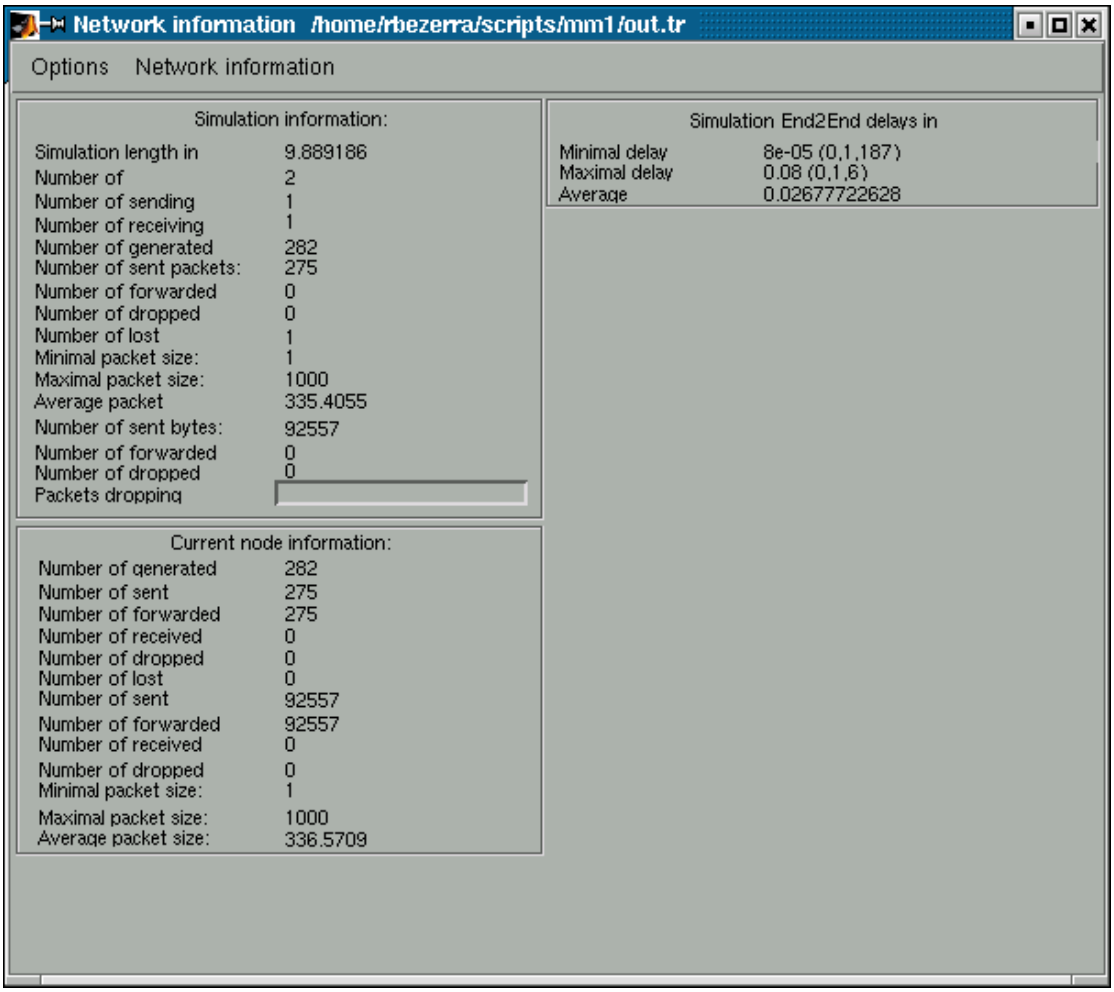

figura 10: Dados obtidos com a simulação do script M/M/1

Parâmetro	Valor
Numero de pacotes gerados	282
Numero de pacotes enviados	275
Numero de pacotes perdidos	1
Tamanho maximo dos pacotes gerados	1000

Tabela 1: Dados extraidos do script M/M/1

Através do TraceGraph pode-se obter gráficos de parâmetros da simulação. A figura 11 demonstra o gráfico da soma acumulativa dos pacotes gerados pelo no 0.

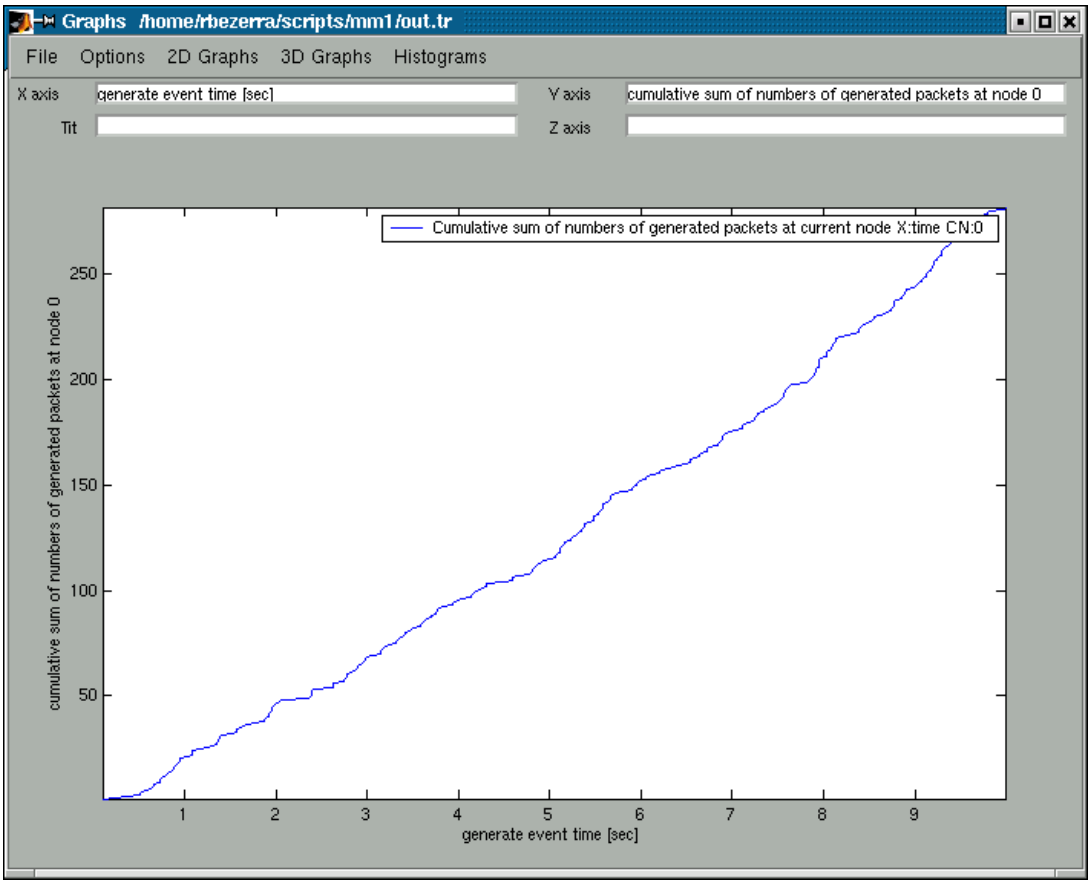

figura 11: Gráfico, gerado pelo TraceGraph, da soma acumulativa dos pacotes gerados no nó 0

O modelo CSMA/CD

O protocolo que utiliza detecção de portadora (carrier sense protocols) e detecção de erros (Collision Detection) é o CSMA/CD (Carrier Sense Multiple Access with collision detection). Segundo este protocolo, quando uma estação tem dados a transmitir, primeiro ela escuta o canal para ver se mais alguém está transmitindo no momento. Quando detectar um canal desocupado, a estação transmitirá um quadro. Se ocorrer uma colisão, a estação perceberá, e em vez de terminar de transmitir seus quadros que, de qualquer forma, já estão deturpados, ela interrompe a transmissão

abruptamente. As colisões podem ser detectadas verificando-se a potência e a largura do pulso do sinal recebido e comparando-o com o sinal transmitido. Este protocolo economiza tempo e largura de banda. O retardo de propagação tem um efeito importante sobre o desempenho do protocolo.

Após detectar uma colisão, as estações transmissoras cancelam suas transmissões, esperam um intervalo de tempo aleatório e, em seguida, tentam novamente quando nenhuma outra estação tenha começado a transmitir nesse meio tempo. Maiores informações sobre o protocolo CSMA/CD em [TANENBAUM 1997].

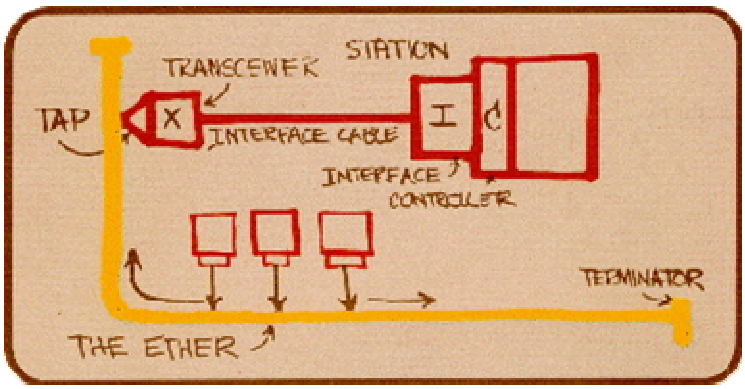

figura 12: Desenho da ethernet

Uma LAN CSMA/Cd 1 – persistente é descrita pelo padrão IEEE 802.3. Este sistema foi chamado de Ethernet, como referencia ao éter luminifero através do qual se pensou, em determinada época, que a radiação eletromagnética se propagava. Quando o físico britânico do século XIX James Clerk Maxwell descobriu que a radiação eletromagnética poderia ser descrita como uma equação de onda, os cientistas presumiram que o espaço deveria ser preenchido com algum meio etéreo no qual a radiação se propagava. Somente depois da famosa experiência de Michelson – Morley, em 1887, os físicos descobriram que a radiação eletromagnética poderia se propagar no vácuo.

O script CSMA/CD

Este script foi retirado de [NS BEGINNERS 2003]. Nas tabelas, de cor cinza, o código do script é mostrado, seguido de sua explicação. O código de cada script, em sua integra, poderá ser visualizado no anexo A.

O primeiro passo é a criação do objeto simulador, que é feita da seguinte maneira:

```
set ns [new Simulator]
```

São criadas diferentes cores para o fluxo de dados na visualização do animador, nam. As cores azuis e vermelhas serão utilizadas.

```
$ns color 1 Blue
$ns color 2 Red
```

Após a criação do objeto simulador, abre-se os arquivos *tracing* para posterior análise, contendo os registros dos eventos de forma mais completa.

```
set file1 [open out.tr w]
set winfile [open WinFile w]
$ns trace-all $file1
```

Para a visualização da simulação, utilizando-se o *Network Animator* (NAM), abre-se outro arquivo onde os registros de eventos simulados serão escritos. Maiores informações sobre o *Network Animator* (NAM) em [NAM 2003].

```
set file2 [open out.nam w]
$ns namtrace-all $file2
```

O processo de finalização da simulação é definido no código abaixo. Fecham-se os arquivos de trace e o *Network Animator* (NAM) é executado.

```
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam out.nam &
        exit 0
}
```

Seis nós são criados, de acordo com o código abaixo. Ao fim o formato e a cor do nó um é modificada em relação ao padrão, configurando-se este nó como uma caixa e de cor vermelha.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n1 color red
$n1 shape box
```

A ligação entre os nos é definida nas linhas de código abaixo, com *links* do tipo duplex, com largura de banda de 2 Mbps e atraso de propagação de 10 *milisegundos* entre os nos 0, 1 e 2. O *link* entre os nos dois e três é configurado como *simplex* para os dois sentidos de comunicação, com largura de banda de 300 Kbps e atraso de propagação de 100 *milisegundos*. A fila do tipo *DropTail* implementa o modelo FIFO (*First in First out*) e descarte de pacotes a partir do estouro do tamanho da fila. No NS o tamanho da fila pode possuir qualquer valor, não havendo restrições. O objeto tipo fila (*queue*) é uma classe de objeto geral capaz de guardar e possivelmente marcar ou descartar pacotes enquanto os mesmos viajam pela topologia simulada, figura 13.

```
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
```

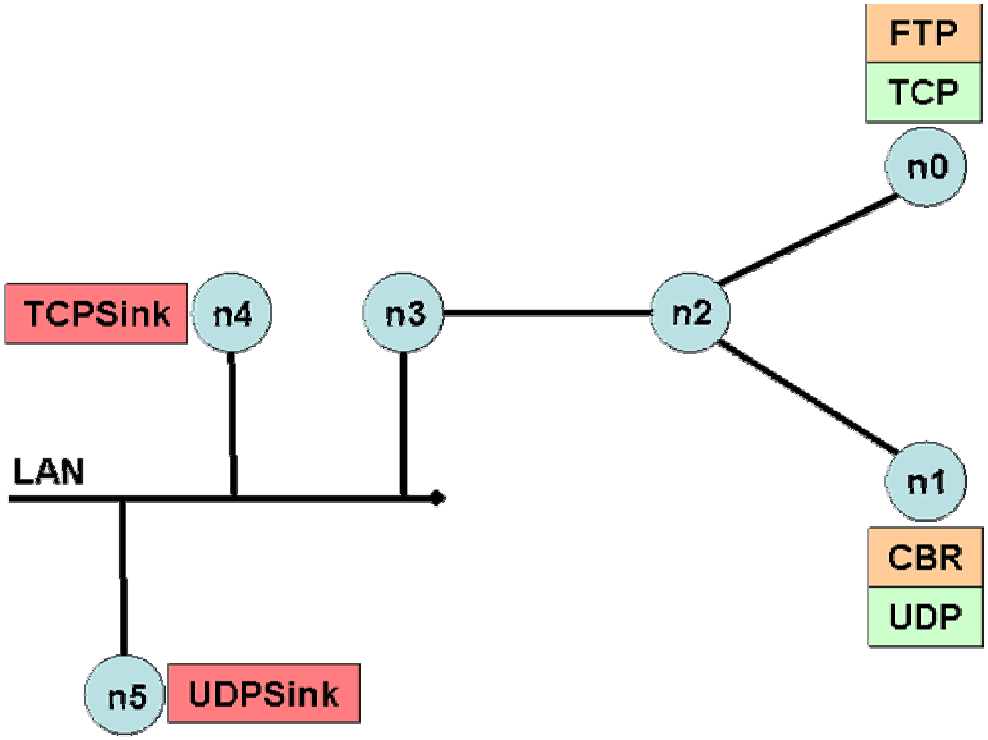

figura 13: Visão simplificada da topologia CSMA/CD

As características de redes locais (LAN) e rede wireless são inerentemente diferentes das redes ponto a ponto. A rede consiste em múltiplos links ponto a ponto que não podem capturar trafego compartilhado e não possuem propriedade de disputa de uma LAN. Para simular essas propriedades, o NS possui um tipo de nó, chamado *LanNode*. A criação e configuração de uma LAN, no NS, diferem ligeiramente daquelas do tipo link ponto a ponto. Os parâmetros para este método são similares ao *duplex-link*.

Os parâmetros opcionais ao *make-lan* especificam o tipo de objetos a serem criados, tais como: camada de ligação (*LL*), tipo de fila, a camada MAC (*Mac*), e a camada física (*Channel*).

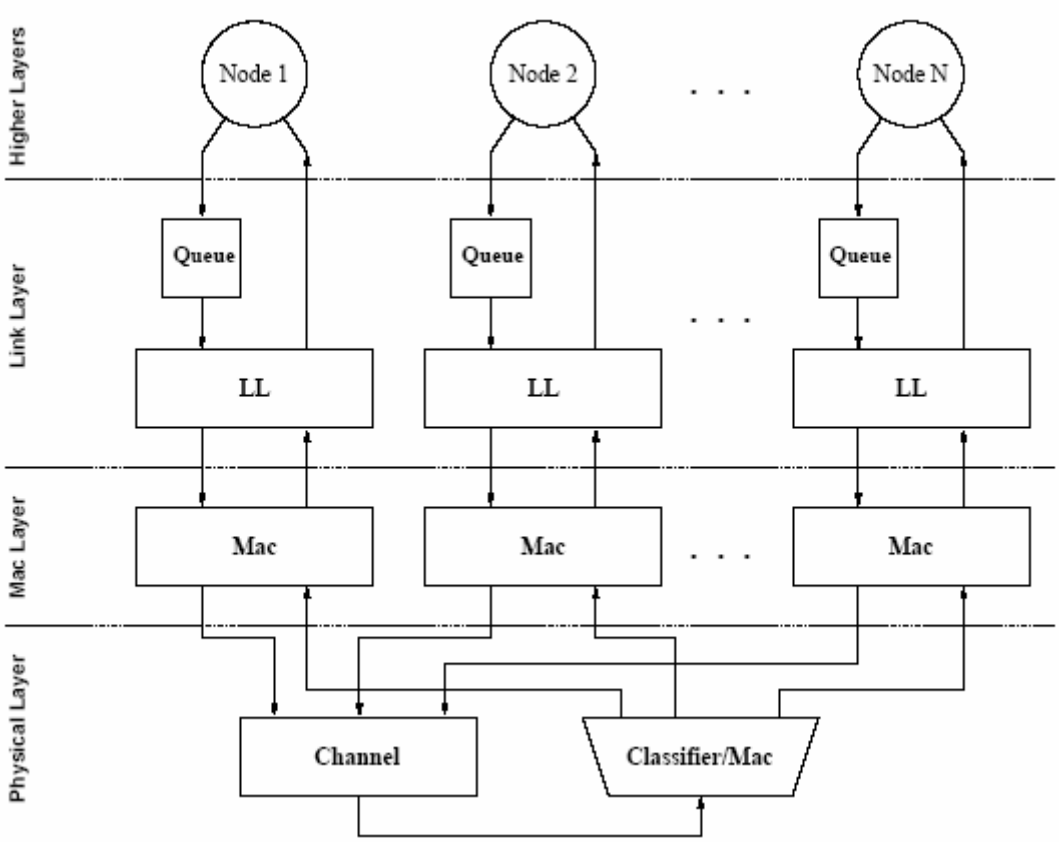

figura 14: Conexão dentro de uma LAN

A figura 14 [NS 2003], ilustra a pilha de camadas de rede que faz simulações de rede local possíveis no NS. Um pacote emitido percorre a pilha através da camada de ligação (*Queue and LL*), depois passa à camada MAC (*Mac*), e por fim à camada física (*Channel e Classifier/Mac*). No receptor o pacote faz então o percurso contrário através da pilha.

A camada física é composta por dois objetos: *Channe*l e o *Classifier/Mac*. O objeto *Channel* simula o meio compartilhado e suporta os mecanismos de acesso ao meio dos objetos MAC no lado da transmissão. No lado da recepção, o *Classifier/Mac* é responsável por entregar e opcionalmente replicar os pacotes aos objetos de recepção do MAC. A função do MAC (Controle de Acesso ao Meio – em português) é permitir que dispositivos compartilhem a capacidade de transmissão de uma rede. Ele controla o acesso ao meio de transmissão, de modo a se ter um uso ordenado e eficiente deste

meio. Dependendo do tipo de camada física, a camada MAC deve conter determinadas funcionalidades como: carrier *sense, collision detection, collision avoidance*, etc. Estas funcionalidades afetam os lados de transmissão e recepção, e são executadas em um único objeto MAC. O objeto MAC segue um determinado protocolo de acesso ao meio, antes de enviar um pacote. Para receber, a camada MAC é responsável por entregar o pacote à camada de ligação.

Acima da camada MAC, a camada de ligação pode possuir muitas funcionalidades, tais como manutenção da fila e retransmissão. O objeto fila (*queue*), pertence à mesma classe de filas possíveis no NS. O objeto da camada de ligação é responsável por simular os protocolos de ligação para transmissão dos dados. Muitos protocolos podem ser executados dentro desta camada tais como: fragmentação e remontagem de pacotes, e protocolo de ligação confiável. Uma outra função importante da camada de ligação é colocar o endereço de destino MAC no cabeçalho MAC do pacote.

No código abaixo é determinado a criação de uma rede local (LAN), entre os nos três, quatro e cinco, figura 13. A rede local possui 500 *Kbps*, de largura de banda, com atraso de propagação de 40 milisegundos. A fila do tipo *DropTail* implementa o modelo FIFO (*First in First out*) e descarte de pacotes a partir do estouro do tamanho da fila, que por padrão no NS pode possuir qualquer valor. A tecnologia *Csma/Cd* também é determinada com o parâmetro *MAC/Csma/Cd*.

```
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]
```

Por padrão, no NS existe apenas um objeto *LanRouter* por LAN, o qual é criado quando a LAN é inicializada. Para cada no da LAN, o objeto *link layer* (*LL*) possui um ponteiro que remete ao *LanRouter*, desta forma o no fica apto a achar o próximo salto para o pacote que foi enviado através da LAN.

Quando a LAN é criada utilizando tanto *make-lan* ou *newLan*, um no virtual LAN, *LanNode*, é criado. *LanNode* mantêm interligados todos os objetos compartilhados pela LAN, tais como: *Channel*, *Classifier/Mac*, e *LanRouter*. *LanNode* é apenas mais um no conectado a cada no da rede local (LAN). Os links que conectam o *LanNode* com os nos pertencentes a LAN são denominados “virtuais” (*Vlink*). O custo de roteamento padrão de cada link é 1/2, sendo assim, o custo para transpor dois *Vlinks* (n1 → LAN → n2) é conta0do como apenas um salto, como pode ser visualizado na figura 15, [NS 2003].

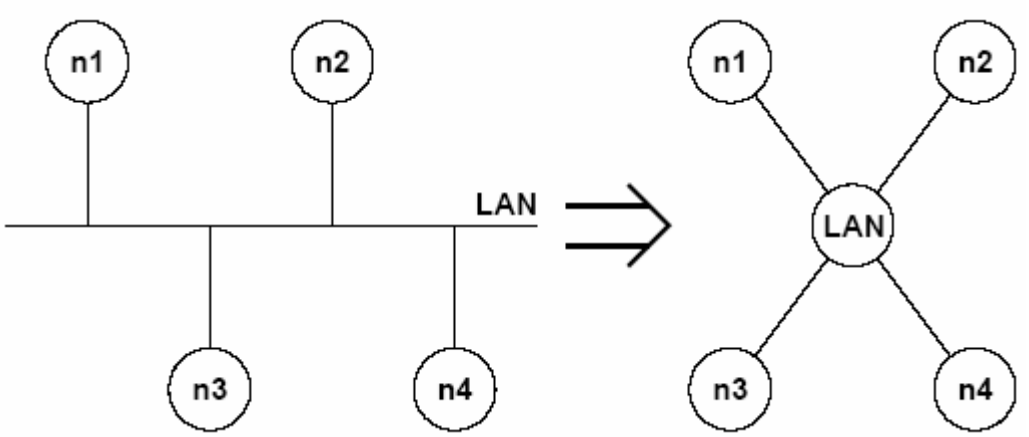

figura 15: Configuração atual da LAN (esquerda) e como é vista pelo rotemanto no NS (direita)

A posição dos nos para visualização na NAM, é configurado nas linhas de código abaixo. Define-se apenas a disposição dos nos na tela do visualizador.

```
$ns duplex-link-op $n0 $n2 orient left-down
$ns duplex-link-op $n1 $n2 orient left-up
$ns simplex-link-op $n2 $n3 orient left
$ns simplex-link-op $n3 $n2 orient right
```

Um agente do tipo TCP é criado e anexado ao nó 0, figura 13. A classe TCP representa um transmissor TCP simplificado, que envia dados para um agente sorvedouro TCP (TCPSink) e processa os reconhecimentos. O simulador suporta inúmeras versões de transmissores TCP. Estes objetos procuram capturar a essência do

comportamento do controle de erros e congestionamento do TCP, mas não possui a intenção de ser uma replica real de uma implementação TCP. O agente TCP nativo no NS não implementa *slinding window*, a segmentação de números e a computação dos ACKs são feitos, inteiramente, em unidades de pacotes. O sorvedouro para o agente TCP é criado e anexado ao nó 4. A conexão entre os agentes anexados aos nós 0 e 4 também é definida. Em seguida a programação do agente TCP é definida, sendo *fid_* o campo *flow ID*, *window_* tamanho máximo da janela de conexão e *packetSize_* o tamanho do pacote utilizado pelo transmissor em bytes, funciona como o parâmetro MSS (*Maximum Segment Size*) no protocolo TCP.

Neste exemplo o tipo de agente transmissor TCP utilizado é o *Newreno*, que é implementado através da primeira linha do código abaixo. Este agente é baseado no agente TCP Reno, modificando sua ação quando de novos ACKs recebidos. O agente sorvedouro é criado utilizando-se o objeto *delayed-ACK*. Mais informações sobre os agentes transmissores e sorvedouros em [NS 2003].

```
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 8000
$tcp set packetSize_ 552
```

O código abaixo descreve a criação de uma classe de trafego FTP (gerador de trafego). Esta classe é anexada ao agente de conexão TCP. Esta classe trabalha adiantando a contagem de pacotes disponíveis a serem enviados pelo agente de transporte TCP. A transmissão dos pacotes disponíveis é controlada pelo algoritmo de controle de fluxo e congestionamento do agente TCP.

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
```

Um agente do tipo UDP é criado e anexado ao nó 1. No Ns um agente UDP aceita dados em pedaços variáveis ou segmentos de dados da aplicação. O tamanho máximo padrão do pacote (MMS) UDP é de 1000 bytes. O sorvedouro para o agente UDP é criado e anexado ao nó 5. A conexão entre os agentes anexados aos nós 1 e 5 também é definida. Em seguida a programação do agente UDP é definida, sendo *fid_* o campo *flow ID*.

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
```

O código abaixo descreve a criação de um objeto de trafego CBR (gerador de trafego). Este objeto é anexado ao agente de conexão UDP. O objeto CBR gera trafego de acordo com uma razão deterministica e o tamanho dos pacotes gerados são constantes.

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false
```

A configuração do agente de trafego é definida na ultimas linhas do código acima, sendo *packet_size_* o tamanho constante dos pacotes gerados, *rate_* a proporção de envio e *random_* parâmetro que introduz ou não ruído randomicamente no escalonador de tempos de partida. Por padrão no Ns, o valor é *off*.

```
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"
```

As linhas de código acima definem em que intervalos de tempo cada gerador de trafego deve iniciar e parar suas atividades.

```
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set wnd [$tcpSource set window_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 0.1 "plotWindow $tcp $winfile"
```

Abaixo, define-se que em um determinado intervalo de tempo, nos arquivos trace será escrito a mensagem "*packet drop*", ou seja, indicando que um pacote foi descartado.

```
$ns at 5 "$ns trace
annotate \"packet drop\""
```

As últimas linhas do script definem o exato momento para a finalização da simulação e inicia a execução do simulador.

```
$ns at 125.0 "finish"
$ns run
```

Ao final da simulação as estatísticas deste modelo podem ser obtidas através do *Xgraph* e *Network Animator (NAM)*.

Resultados do script csma-cd

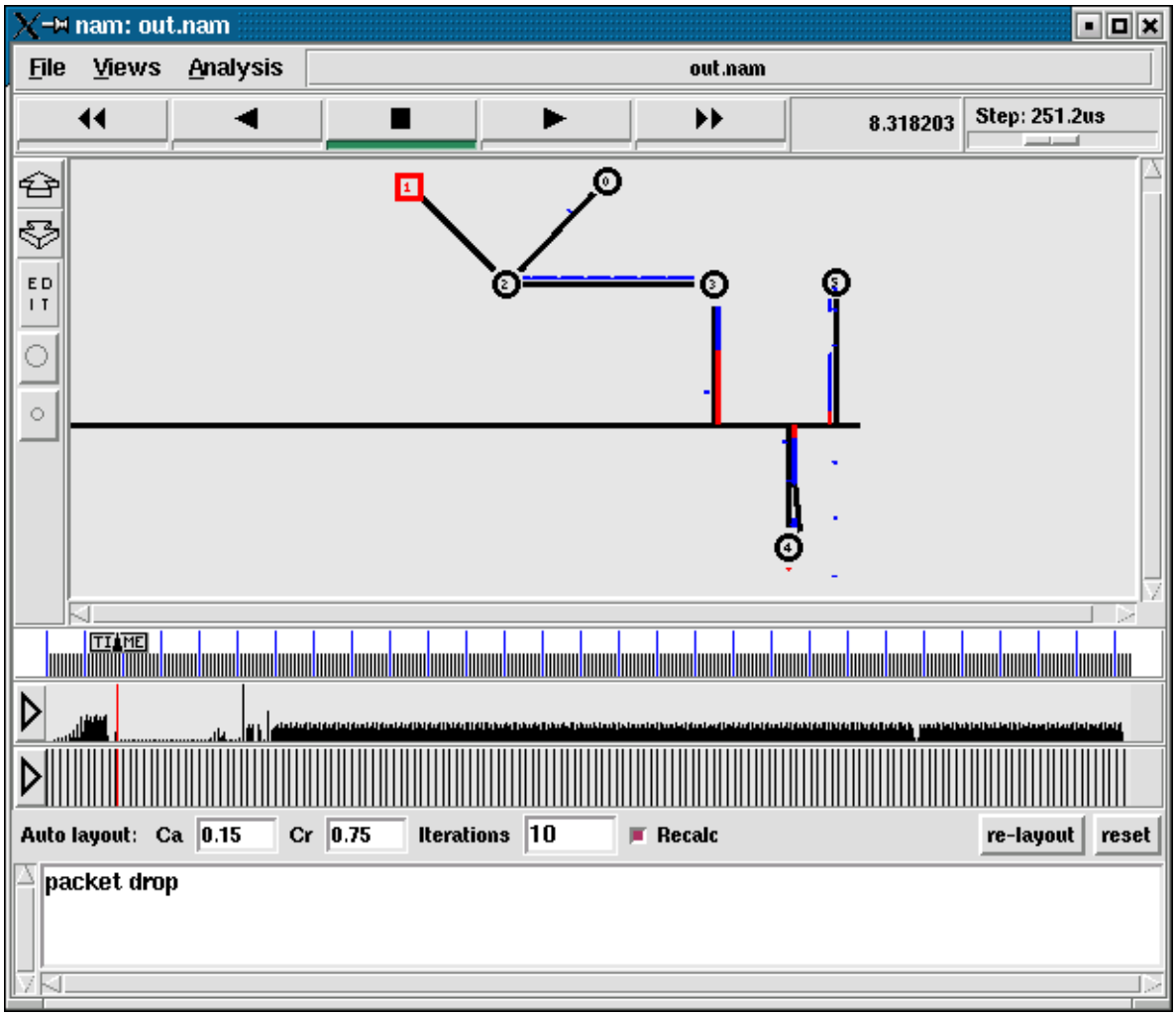

figura 16: Network Animator (NAM) - Script csma-cd

Utilizando-se o *Network Animator (NAM)* alguns resultados podem ser extraídos do script csma-cd. A figura 16 demonstra a topologia do script csma-cd simulado, pode-se observar os pacotes percorrendo todos os nos pertencentes a rede local (LAN), nos 3, 4 e 5.

O no 5 descarta os pacotes de cor azul, enquanto que o no 4 descarta os pacotes de cor vermelha, seguindo a teoria das redes locais que pacotes não endereçados à maquina não são recebidos. Ainda na figura 16 pode-se observar a utilização da largura de banda dos links entre os nos 0 e 2 e entre os nos 1 e 2, respectivamente, de baixo para cima. É importante observar como os pacotes são enviados de forma constante através do link entre os nós 1 e 2. O contrario acontece com o link entre os nos 0 e 2.

Resultados mais apurados podem ser obtidos através do *Traegraph*, utilizando o arquivo de *trace* out.tr.

O Modelo de serviço TCP

O serviço TCP é obtido quando tanto o transmissor quanto o receptor criam pontos terminais, denominados *sockets*. Cada *socket* tem um numero (endereço) que consiste no endereço IP do *host* mais um numero de 16 bits local para esse *host*, chamado porta. Para que o serviço TCP funcione, é necessário que uma conexão seja explicitamente estabelecida entre um *socket* da maquina transmissora e um *socket* da maquina receptora [TANENBAUM 1997].

Todas as conexões TCP são *full-duplex* e ponto a ponto. *Full-duplex* quer dizer que o trafego pode ser feito em ambas as direções ao mesmo tempo. Ponto a ponto quer dizer que cada conexão possui exatamente dois pontos terminais.

Uma conexão TCP é um fluxo de dados e não um fluxo de mensagens. As fronteiras das mensagens não são preservadas de uma extremidade à outra. Se, por exemplo, o processo transmissor executar quatro escritas de 512 bytes cada no fluxo TCP, esses dados poderão ser entregues ao processo receptor em quatro partes de 512 bytes, em duas de 1024 bytes, uma de 2048 bytes, ou em qualquer outra divisão [TANENBAUM 1997].

As entidades TCP transmissoras e receptoras trocam dados na forma de segmentos. Um segmento consiste em um cabeçalho fixo de 20 bytes (mais uma parte opcional), seguido de zero ou mais bytes de dados. O software TCP decide qual deve

ser o tamanho dos segmentos. Ele pode acumular dados de varias escritas em um único segmento ou dividir os dados de uma única escrita em vários segmentos. Dois fatores restringem o tamanho do segmento. Primeiro, cada segmento, incluindo o cabeçalho do TCP, deve caber na carga útil do IP, que é de 65.535 bytes. Segundo, cada rede possui uma MTU (*Maximum Transfer Unit*), e cada segmento deve caber na MTU [TANENBAUM 1997].

O protocolo básico utilizado pelas entidades TCP é o protocolo de janela deslizante (*slindig window*). Quando envia um segmento, o transmissor também dispara um temporizador. Quando o segmento chega ao destino, a entidade TCP receptora retorna um segmento (com ou sem dados, de acordo com as circunstancias) como um numero de confirmação igual ao próximo numero de seqüência que espera receber. Se o temporizador do transmissor expirar antes de a confirmação ser recebida, o segmento será retransmitido. No *network Simulator* (NS) não existe implementação do protocolo básico de janela deslizante utilizado pelo serviço TCP. Para implementar esse protocolo é necessário adicionar arquivos à biblioteca existente no NS, adicionar objetos de agentes transmissores e sorvedouros e por fim arquivos para efetuar a validação deste modelo. A implementação do protocolo de janela deslizante no NS pode ser pesquisada em [TCP 2003].

Quando a carga oferecida a qualquer rede é maior do que sua capacidade, acontece um congestionamento. A Internet não é exceção a essa regra. Em seguida modelos de algoritmos que foram desenvolvidos na ultima década para lidar com o congestionamento serão descritos. Mesmo que a camada de rede também gerencie o congestionamento, o trabalho mais pesado é feito pelo TCP, pois a verdadeira solução para o congestionamento é diminuir a taxa de transmissão de dados [TANENBAUM 1997].

Em seguida a tecnologia *stop and wait* que foi desenvolvida para evitar o congestionamento em Redes de Computadores será descrita.

O controle de congestionamento Stop and Wait

O protocolo “Stop and Wait” é uma técnica fundamental para prover transferência confiável de dados em um sistema não confiável.

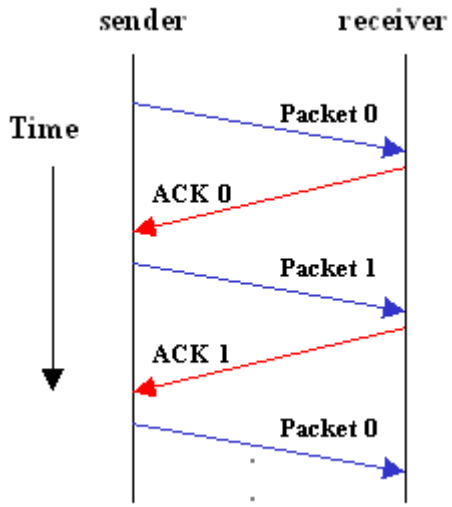

figura 17: Operação normal do protocolo "Stop and Wait"

A figura 17 [TCP 2003], mostra a operação normal do protocolo “Stop and Wait”. Apos transmitir um pacote, o transmissor espera por uma confirmação de recebimento do pacote enviado, advindo do receptor, antes de enviar o próximo pacote.

Para suportar esta característica, o transmissor mantém uma copia de cada pacote enviado. Para evitar confusão causada por pacotes atrasados ou Acks duplicados, o protocolo "*stop and wait*" envia cada pacote com uma seqüência numérica única, os

ACKs recebidos devem possuir essa seqüência para que a correspondência entre pacote enviado e confirmação de recebimento seja efetuada.

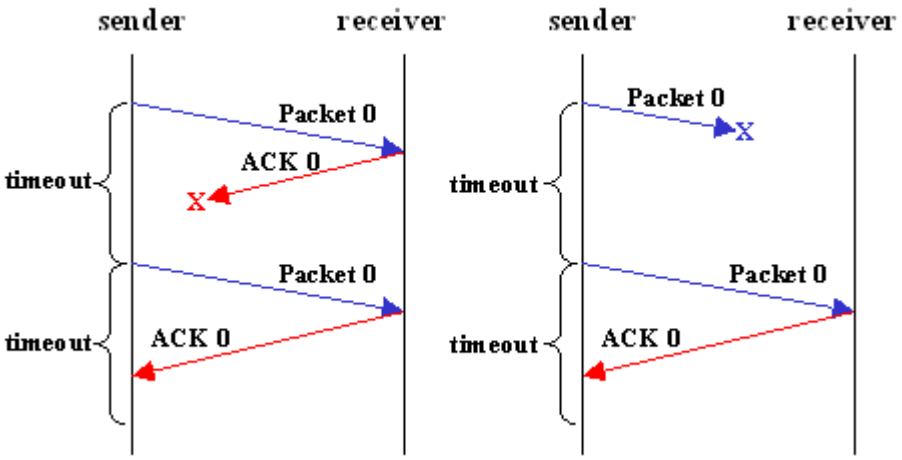

figura 18: Ocorrência de timeout

A figura 18 [TCP 2003], demonstra a utilização do controle de congestionamento "*Stop and Wait*" numa ocorrência de *timeout*. Se o transmissor não recebe um ACK correspondente a transmissão, sem a ocorrência de erros, de um pacote previamente enviado depois de um certo período de tempo, o transmissor interpreta a ocorrência de um *timeout* e retransmite o pacote. Existem dois casos em que o transmissor não recebe o ACK, o primeiro é quando o ACK é perdido e segundo, quando o ACK não chega a ser transmitido. Para suportar esta característica, o transmissor mantém um temporizador ativo para cada pacote.

O script Stop and Wait

Este script foi retirado de [TCP 2003]. Nas tabelas, de cor cinza, o código do script é mostrado, seguido de sua explicação. O código de cada script, em sua integra, poderá ser visualizado no anexo A.

O primeiro passo é a criação do objeto simulador, que é feita da seguinte maneira:

```
set ns [new Simulator]
```

Nas linhas do código abaixo são criados os nós 0 e 1.

```
set n0 [$ns node]
set n1 [$ns node]
```

Nas linhas de código abaixo são criadas legendas para serem utilizadas durante a execução do *Network Animator* (NAM). São utilizadas legendas para os nos 0 e 1, respectivamente.

```
$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
```

As linhas de código abaixo definem a abertura dos arquivos *tracing* para posterior análise, contendo os registros dos eventos de forma mais completa.

Para a visualização da simulação, utilizando-se o *Network Animator* (NAM), abre-se outro arquivo onde os registros de eventos simulados serão escritos. Maiores informações sobre o *Network Animator* (NAM), em [NAM 2003].

```
set nf [open A1-stop-n-wait.nam w]
$ns namtrace-all $nf
set f [open A1-stop-n-wait.tr w]
$ns trace-all $f
```

Configura-se então a ligação entre estes nós, com link do tipo *duplex*. Largura de banda de 200 Kbps com atraso de propagação de 200 *milisegundos*. A fila do tipo DropTail implementa o modelo FIFO (First in First out) e descarte de pacotes a partir

do estouro do tamanho da fila, que por padrão no NS pode possuir qualquer valor. A penúltima linha do código abaixo provê os artifícios necessários para a configuração do limite do tamanho máximo da fila entre os nós participantes da comunicação, neste caso os nós 0 e 1.

```
$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns queue-limit $n0 $n1 10
Agent/TCP set nam_tracevar true
```

Um agente do tipo TCP é criado e anexado ao nó 0. A classe TCP representa um transmissor TCP simplificado, que envia dados para um agente sorvedouro TCP (*TCPSink*) e processa os reconhecimentos. O simulador suporta inúmeras versões de transmissores TCP. Estes objetos procuram capturar a essência do comportamento do controle de erros e congestionamento do TCP, mas não possui a intenção de ser uma replica real de uma implementação TCP. Em seguida a programação do agente TCP é definida, sendo *window_* tamanho máximo da janela de conexão e *maxcwnd_* o limite máximo para a janela de congestionamento para a conexão TCP, por padrão é determinada como 0 (zero), sendo ignorado este parâmetro. Neste exemplo o tipo de agente transmissor TCP utilizado é o *Tahoe*, que é implementado através da primeira linha do código abaixo.

```
set tcp [new Agent/TCP]
$tcp set window_ 1
$tcp set maxcwnd_ 1
$ns attach-agent $n0 $tcp
```

O agente sorvedouro é criado utilizando-se o objeto *TCPSink* e anexado ao no 1. É responsável por enviar os ACKs de resposta para o nó transmissor, gerando um ACK para cada pacote recebido. Maiores informações sobre os agentes transmissores e sorvedouros em [NS 2003].

```
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
```

```
$ns connect $tcp $sink
```

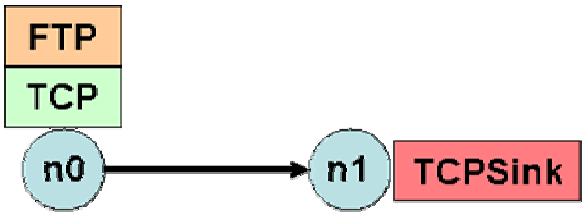

figura 19: Topologia do script Stop and Wait

O código abaixo descreve a criação de uma classe de trafego FTP (gerador de trafego). Esta classe é anexada ao agente de conexão TCP. Esta classe trabalha adiantando a contagem de pacotes disponíveis a serem enviados pelo agente de transporte TCP. A transmissão dos pacotes disponíveis é controlada pelo algoritmo de controle de fluxo e congestionamento do agente TCP. O método *tracevar* do agente *nsagent* é utilizado para criar arquivos de trace com características das variáveis indicadas no ato da criação.

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns add-agent-trace $tcp tcp
$ns monitor-agent-trace $tcp
$tcp tracevar cwnd_
```

As linhas de código abaixo definem em que intervalos de tempo o gerador de trafego deve iniciar e parar suas atividades. Reiniciando os nós 0 e 1, deixando-os com características de agentes nulos.

```
$ns at 0.1 "$ftp start"
$ns at 3.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n1 $sink"
$ns at 3.5 "finish"
```

Nas linhas de código abaixo, é determinada a impressão de determinadas mensagens, em determinados períodos de tempo, para utilizarão no *Network animator* (NAM).

```
$ns at 0.0 "$ns trace-annotate \"Stop and Wait with normal operation\""
$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.1\""
$ns at 0.11 "$ns trace-annotate \"Send Packet_0\""
$ns at 0.35 "$ns trace-annotate \"Receive Ack_0\""
$ns at 0.56 "$ns trace-annotate \"Send Packet_1\""
$ns at 0.79 "$ns trace-annotate \"Receive Ack_1\""
$ns at 0.99 "$ns trace-annotate \"Send Packet_2\""
$ns at 1.23 "$ns trace-annotate \"Receive Ack_2  \""
$ns at 1.43 "$ns trace-annotate \"Send Packet_3\""
$ns at 1.67 "$ns trace-annotate \"Receive Ack_3\""
$ns at 1.88 "$ns trace-annotate \"Send Packet_4\""
$ns at 2.11 "$ns trace-annotate \"Receive Ack_4\""
$ns at 2.32 "$ns trace-annotate \"Send Packet_5\""
$ns at 2.55 "$ns trace-annotate \"Receive Ack_5    \""
$ns at 2.75 "$ns trace-annotate \"Send Packet_6\""
$ns at 2.99 "$ns trace-annotate \"Receive Ack_6\""
$ns at 3.1 "$ns trace-annotate \"FTP stops\""
```

O código abaixo define o processo de finalização da simulação. Carregando os arquivos de trace com os dados gerados e fechando-os após o término da simulação. O *Network Animator* (NAM) é executado para visualização da simulação.

```
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        puts "filtering..."
        exec tclsh ../ns-allinone-2.1b5/nam-1.0a7/bin/namfilter.tcl A1-stop-n-wait.nam
      puts "running nam..."
      exec nam A1-stop-n-wait.nam &
        exit 0
}
```

A última linha do script, finalmente, inicia a execução do simulador.

```
$ns run
```

Resultados do script Stop and Wait

Após a conclusão da simulação o *Network Animator* é inicializado, como mostra a figura 20 [TCP 2003]. O primeiro pacote é enviado e a resposta de recebimento do

pacote será aguardada. No *Network Animator* pode-se visualizar uma legenda, onde podemos acompanhar a progressão da simulação.

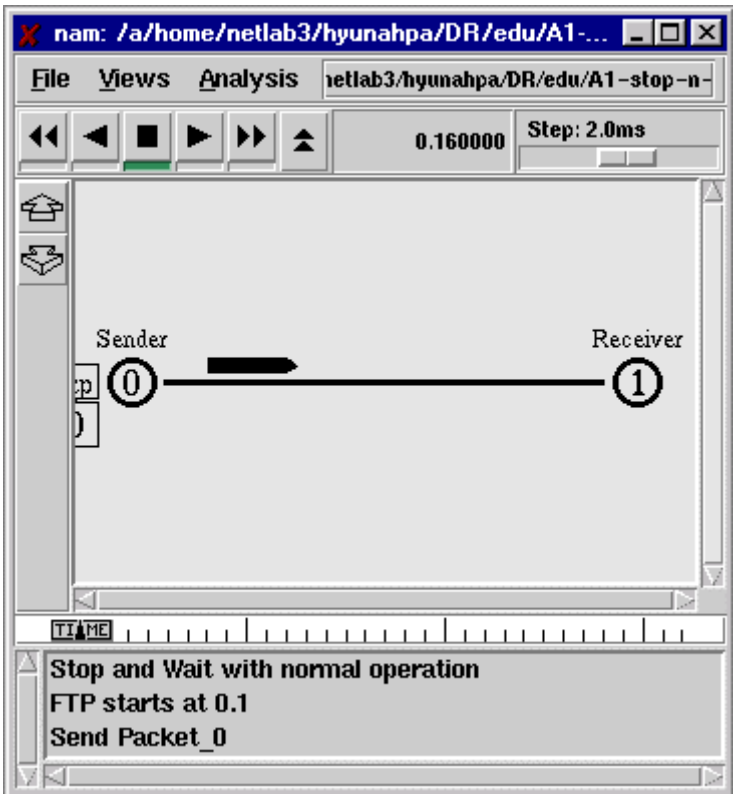

figura 20: Inicialização da NAM após termino da simulação

Em seguida o ACK correspondente ao pacote enviado será recebido, como mostra a figura 21 [TCP 2003].

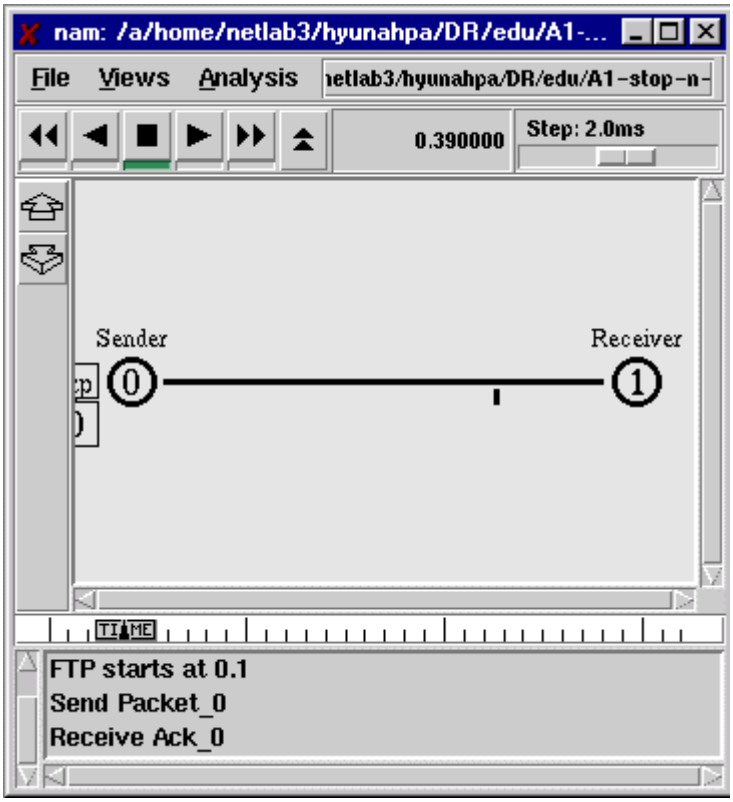

figura 21: Recebimento do ACK correspondente ao primeiro pacote

Após o recebimento do ACK, correspondendo ao perfeito envio do primeiro pacote, o transmissor envia o segundo pacote, como mostra a figura 22 [TCP 2003]. Estes procedimentos serão repetidos até o fim da conexão entre os nos participantes.

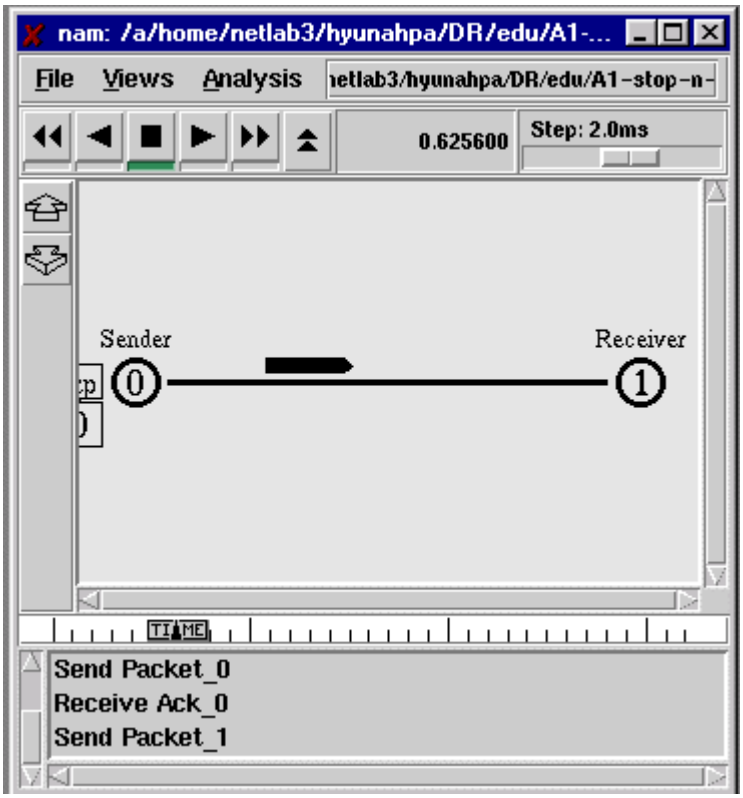

figura 22: Envio do segundo pacote após recebimento do primeiro ACK

A deficiência do algoritmo de controle de congestionamento "*stop and wait*" é que ele prove ao transmissor apenas a possibilidade de enviar um pacote por cada vez, ou seja, apenas um pacote "habita" o link por vez. O transmissor deve esperar até receber um ACK referente ao pacote enviado anteriormente para poder enviar um novo pacote. Como resultado, este algoritmo prove um desperdício substancial de largura de banda. Nas redes atuais esse algoritmo é altamente ineficiente.

Considerações finais

A pesquisa científica tem na simulação uma importante e vital ferramenta, que permite modelar sistemas complexos e obter resultados para cenários diferentes. A Engenharia de Redes de Computadores é uma dessas áreas de pesquisa que usa extensamente os simuladores, para teste de algoritmos de qualidade de serviço e desempenho.

O presente trabalho foi importante na formação acadêmica do autor, pois propiciou um aprofundamento na pesquisa das tecnologias empregadas em Redes de Computadores e sistemas computacionais. O Simulador *Network Simulator (NS)* pode ser utilizado para implementar novas tecnologias de Redes de Computadores, testando-as, num ambiente virtual, seguro e com baixos custos financeiros.

A ferramenta de simulação Network Simulator demonstra ser adequada para o desenvolvimento de técnicas de analise de desempenho de sistemas. Outro ponto forte do Network Simulator é a contribuição, a nível mundial, de pesquisadores de diversas universidades, de bibliotecas e funções de tecnologias de redes.

Sendo assim o Network Simulator mostra-se importante nos centros de desenvolvimento e pesquisa de protocolos e tecnologia ligados a redes de computadores.

Referencias Bibliográficas

[NS 2003] **THE NETWORK SIMULATOR – ns-2** [online]. Disponível na Internet via URL: http://www.isi.edu/nsnam. Arquivo capturado em 23/08/2003.

[ARAÚJO 2002] ARAÚJO, Rafael; PORTNOI Marcos. **Uma visão geral da ferramenta de simulação de redes – Network Simulator (NS)**. SEPA – Seminário Estudantil de Produção Acadêmica, Salvador, Ano VI, v. 6, n. 6, p. 173-181, 2002.

[JAIN 1991] JAIN, R. **The art of computer systems perfomance analysis.** John Wiley & Sons, 1991

[TANENBAUM 1997] TANENBAUM, Andrew S. **Redes de computadores.** Editora Campus. Terceira Edição, 1997

[KUROSE 2003] KUROSE, James F; ROSS, Keith W. **Redes de Computadores e a Internet – Uma Nova Abordagem**. Addison Wesley. Primeira Edição – São Paulo, 2003.

[BRITO 2002] BRITO, Sergio de Figueiredo et al. **Simulation tool and usage strategy – a practical and pragmatic approach**. Trabalho apresentado no seminário INDUSCON 2002, Salvador, 03 jul. 2002.

[SCRIPTING 2003] **SCRIPTING: HIGHER LEVEL PROGRAMMING FOR THE 21ST CENTURY** [online]. Disponível na Internet via URL: http://home.pacbell.net/ouster/scripting.html. Arquivo capturado em 07/04/2003.

[SAMAN 2003] **SAMAN (Simulation Augmented by Measurement and Analysis for Networks)** [online]. Disponível na Internet via URL: http://www.isi.edu/saman/index.html. Arquivo capturado em 05/05/2002.

[VINT 2003] **VINT (Virtual InterNetwork Testbed)** [online]. Disponível na internet via URL: http://www.isi.edu/nsnam/vint/index.html. Arquivo capturado em 05/05/2002.

[NS BEGINNERS 2003] **NS SIMULATOR COURSE FOR BEGINNERS** [online]. Disponível na Internet via URL: Arquivo capturado em 24.07.2003

[OTCL 2003] **OTCL (Object Tool Command Language)** [online]. Disponível na Internet via URL: http://otcl-tclcl.sourceforge.net/otcl/. Arquivo capturado em 24.07.2002.

[INSTALLATION PROBLEMS 2003] **INSTALLATION PROBLEMS AND BUG FIXES WEBPAGE** [online]. Disponível na Internet via URL: http://www.isi.edu/nsnam/ns/ns-problems.html. Arquivo capturado em 06/05/2002.

[CHUNG 2003] CHUNG, Jae; CLAYPOOL, Mark**.** **Ns by example** [online]. Disponível na Internet via URL: http://nile.wpi.edu/NS. Arquivo capturado em 05/05/2002.

[NAM 2003] **THE NETWORK ANIMATOR – NAM** [online]. Disponível na Internet via URL: http://www.isi.edu/nsnam/nam/index.html. Arquivo capturado em 16/10/2003.

[TRACEGRAPH 2002] **TRACE GRAPH** [online]. Disponivel na internet via URL: http://www.geocities.com/tracegraph/. Arquivo capturado em 13/07/2002

[TCP 2003] **Directed Research in VINT project** [online]. Disponivel na Internet via URL: http://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_HYUNAH/D-Research/DR.html. Arquivo capturado em 16/09/2003.

Apendice A

Network Simulator – Visão Geral da Ferramenta de Simulação de Redes

Instalação do *ns*

O *ns* foi construído para rodar preferencialmente em plataformas Unix (FreeBSD, SunOS, Solaris, Linux, dentre outras). Pode também rodar em plataforma Microsoft Windows, apesar de que sua instalação não é tão automatizada e não segue os padrões de instalação dos softwares feitos para Windows. Este artigo descreverá características do simulador em sistema operacional Linux (distribuição RedHat 7.1) e arquitetura Intel PC.

Fornecido em fontes, o ns é compilado durante o processo da instalação. Assim, faz-se necessário um compilador C++ no computador onde será instalado. Outros pré-requisitos para o ambiente são fornecidos em 0. Como o simulador é compilado na instalação, ele pode, a princípio, ser executado em qualquer arquitetura de computador, como Intel/AMD ou RISC.

O pacote do simulador é composto dos seguintes módulos básicos:

- Tcl/Tk: Interpretador de linguagem Tcl, que é a interface do simulador com o usuário.
- OTcl: suplemento de orientação a objetos para o Tcl.
- Tclcl: implementação de classes para Tcl.
- ns-2: classes do simulador propriamente dito.
- nam-1: visualizador e animador gráfico de topologias de rede e simulação.

- xgraph: ferramenta de plotagem de gráficos.
- cweb e SGB: bibliotecas requeridas para sgb2-ns e gt-itm, abaixo.
- Gt-itm, gt-itm e sgb2-ns: gerador de topologias.
- zlib: ferramenta de compressão de arquivos.

Há duas formas de instalar o simulador: através do *ns-allinone*, que reúne todos os pacotes acima descritos, ou obtendo cada pacote e fazendo sua instalação separadamente. Nem todos os módulos acima são requeridos para que o simulador funcione, portanto pode-se preparar ambientes com instalações mínimas e outros com mais ferramentas. Já outros módulos externos (como *perl*) são necessários para realizar certas funções. Em 0 mais detalhes estão disponíveis.

Com o pacote ns-allinone, a instalação é razoavelmente simples, desde que o ambiente tenha todos os pacotes externos necessários. Deve ser descompactado com auxílio de uma ferramenta apropriada (como o ark) para o diretório onde se deseja que o simulador permaneça (por padrão, recomenda-se colocar em /usr/local). Dentro deste diretório, executa-se o script de instalação: ./install.

Os fontes de cada pacote do simulador serão então apropriadamente compilados na ordem correta (processo que pode ser bastante moroso em computadores mais antigos). Se não houver nenhum erro na compilação, o script dará mensagem de sucesso e fornecerá instruções complementares para modificação de variáveis de ambiente do sistema operacional, como por exemplo a variável PATH. Essas modificações têm de ser feitas manualmente e para cada usuário que se utilizará do simulador, no arquivo que contém as configurações do usuário (em RedHat Linux, o arquivo oculto .bash_profile, que fica nos diretórios individuais de cada usuário). Se

houver alguma mensagem de erro, o operador deve procurar auxílio na página de apoio 0 ou ainda na lista de discussão mantida pelos desenvolvedores.

Uma vez terminada a compilação, deve-se proceder à validação do simulador, que consiste em executar várias simulações pré-programadas e comparar seus resultados contra resultados-padrão. Isso é feito através do comando *./validate*, dentro do diretório ns (no caso da versão 9, ns-2.1b9). Dependendo da versão do Unix utilizada e da arquitetura do computador, um ou mais testes podem diferir dos resultados-padrão. Novamente, o operador deve pesquisar as razões destas diferenças na Internet (nas páginas de apoio ou listas de discussão) de forma a verificar se tais incongruências prejudicarão resultados posteriores (há diferenças causadas apenas por reordenamento de dados, o que não invalida os resultados).

Anexo A

Script MM1

```
set ns [new Simulator]

set tf [open out.tr w]
$ns trace-all $tf

set lambda 30.0
set mu     33.0

set n1 [$ns node]
set n2 [$ns node]
# Since packet sizes will be rounded to an integer
# number of bytes, we should have large packets and
# to have small rounding errors, and so we take large bandwidth
set link [$ns simplex-link $n1 $n2 100kb 0ms DropTail]
$ns queue-limit $n1 $n2 100000

# generate random interarrival times and packet sizes
set InterArrivalTime [new RandomVariable/Exponential]
$InterArrivalTime set avg_ [expr 1/$lambda]
set pktSize [new RandomVariable/Exponential]
$pktSize set avg_ [expr 100000.0/(8*$mu)]

set src [new Agent/UDP]
$ns attach-agent $n1 $src

# queue monitoring
set qmon [$ns monitor-queue $n1 $n2 [open qm.out w] 0.1]
$link queue-sample-timeout

proc finish {} {
   global ns tf
   $ns flush-trace
   close $tf
   exit 0
}

proc sendpacket {} {
   global ns src InterArrivalTime pktSize
   set time [$ns now]
   $ns at [expr $time + [$InterArrivalTime value]] "sendpacket"
   set bytes [expr round ([$pktSize value])]
```

```
    $src send $bytes
}

set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $src $sink
$ns at 0.0001 "sendpacket"
$ns at 1000.0 "finish"

$ns run
```

Script CSMA/CD

```
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the Trace files
set file1 [open out.tr w]
set winfile [open WinFile w]
$ns trace-all $file1

#Open the NAM trace file
set file2 [open out.nam w]
$ns namtrace-all $file2

#Define a 'finish' procedure
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam out.nam &
        exit 0
}

#Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n1 color red
```

```
$n1 shape box

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd
Channel]

# $ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
# $ns duplex-link $n3 $n5 0.5Mb 30ms DropTail

#Give node position (for NAM)
# $ns duplex-link-op $n0 $n2 orient right-down
# $ns duplex-link-op $n1 $n2 orient right-up
# $ns simplex-link-op $n2 $n3 orient right
# $ns simplex-link-op $n3 $n2 orient left
# $ns duplex-link-op $n3 $n4 orient right-up
# $ns duplex-link-op $n3 $n5 orient right-down


#Set Queue Size of link (n2-n3) to 10
# $ns queue-limit $n2 $n3 20

#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 8000
$tcp set packetSize_ 552

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
```

```
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false

$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"

# next procedure gets two arguments: the name of the
# tcp source node, will be called here "tcp",
# and the name of output file.

proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set wnd [$tcpSource set window_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 0.1 "plotWindow $tcp $winfile"


$ns at 5 "$ns trace-annotate \"packet drop\""


# PPP

$ns at 125.0 "finish"
$ns run
```

Script Stop and Wait

```
# stop and wait protocol in normal situation
# features : labeling, annotation, nam-graph, and window size monitoring

set ns [new Simulator]

set n0 [$ns node]
set n1 [$ns node]

$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
```

```
set nf [open A1-stop-n-wait.nam w]
$ns namtrace-all $nf
set f [open A1-stop-n-wait.tr w]
$ns trace-all $f

$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns queue-limit $n0 $n1 10

Agent/TCP set nam_tracevar_ true

set tcp [new Agent/TCP]
$tcp set window_ 1
$tcp set maxcwnd_ 1
$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns add-agent-trace $tcp tcp
$ns monitor-agent-trace $tcp
$tcp tracevar cwnd_

$ns at 0.1 "$ftp start"
$ns at 3.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n1 $sink"
$ns at 3.5 "finish"

$ns at 0.0 "$ns trace-annotate \"Stop and Wait with normal operation\""

$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.1\""

$ns at 0.11 "$ns trace-annotate \"Send Packet_0\""
$ns at 0.35 "$ns trace-annotate \"Receive Ack_0\""
$ns at 0.56 "$ns trace-annotate \"Send Packet_1\""
$ns at 0.79 "$ns trace-annotate \"Receive Ack_1\""
$ns at 0.99 "$ns trace-annotate \"Send Packet_2\""
$ns at 1.23 "$ns trace-annotate \"Receive Ack_2  \""
$ns at 1.43 "$ns trace-annotate \"Send Packet_3\""
$ns at 1.67 "$ns trace-annotate \"Receive Ack_3\""
$ns at 1.88 "$ns trace-annotate \"Send Packet_4\""
$ns at 2.11 "$ns trace-annotate \"Receive Ack_4\""
$ns at 2.32 "$ns trace-annotate \"Send Packet_5\""
$ns at 2.55 "$ns trace-annotate \"Receive Ack_5    \""
$ns at 2.75 "$ns trace-annotate \"Send Packet_6\""
```

```
$ns at 2.99 "$ns trace-annotate \"Receive Ack_6\""

$ns at 3.1 "$ns trace-annotate \"FTP stops\""

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf

        puts "filtering..."
        exec tclsh ../ns-allinone-2.1b5/nam-1.0a7/bin/namfilter.tcl A1-stop-n-wait.nam
      puts "running nam..."
      exec nam A1-stop-n-wait.nam &
        exit 0
}

$ns run
```