Tutorial de NS-2

Lucas Coelho Gonçalves e Marcos Estevo de Oliveira Corrêa

Julho de 2005

Índice

1. INTRODUÇÃO

A disciplina de Comunicação de Dados IV tem um programa que vai desde as topologias de rede até os padrões e arquiteturas (TCP/IP, IEEE e OSI). Os conceitos teóricos são apresentados sem oferecer algum tipo de experimentação prática. Isso acaba limitando o aprendizado por parte dos alunos.

Os recursos visuais tradicionalmente usados não tem sido suficientes para a compreensão total dos conceitos. Por isso, estamos sugerindo uma nova ferramenta visual para ilustrar os exemplos dados em sala de aula e que poderá ser utilizada como parte da disciplina no desenvolvimento de atividades extras pelos alunos.

Este projeto tem por finalidade apresentar simulações que complementem os aspectos teóricos dados em sala de aula. Através dessas simulações, realizadas no Network Simulator 2 (NS-2)[1], pretendemos ajudar os próximos alunos na compreensão dos conceitos dessa disciplina.

No capítulo 2 damos uma introdução sobre o NS-2 e apresentamos os passos para a criação de uma simulação. Neste ponto inicial explicamos a função de cada passo na criação do script. Em seguida apresentamos algumas funções e conceitos da camada de rede, transporte e aplicação que podem ser exploradas pelo simulador. Ao fim do capítulo 2 falamos sobre a implementação de redes sem fio.

No capítulo 3 descrevemos as simulações desenvolvidas com sucesso neste projeto. Apresentamos sobre cada uma das simulações os conceitos abordados por ela e é feita uma breve descrição do seu funcionamento.

No capítulo 4 apresentamos as conclusões finais do projeto, dos trabalhos realizados durante sua elaboração e do alcance dos nossos objetivos.

2. NETWORK SIMULATOR

O Network Simulator é um simulador de eventos discreto resultante de um projeto conhecido como VINT (Virtual InterNetwork Testbed). Dentre outros, compõem esse projeto a DARPA, USC/ISI, Xerox PARC, LBNL, e a Universidade de Berkeley. Uma grande vantagem do Network Simulator (NS) está no fato de ele ser totalmente gratuito e com código fonte aberto, o que permite ao usuário fazer os ajustes que precisar. O simulador oferece suporte à simulação de um grande número de tecnologias de rede (com e sem fio), diferentes cenários baseados nos protocolos TCP e UDP, diversas políticas de fila, caracterização de tráfego com diversas distribuições estatísticas e muito mais.

A programação do NS é feita em duas linguagens: C++ para a estrutura básica (protocolos, agentes, etc) e OTCL[2] (Object-oriented Tool Command Language) para uso como *fronten*d. OTCL é uma linguagem interpretada, desenvolvida pelo MIT. Nela serão efetivamente escritas as simulações. O motivo para se utilizar duas linguagens de programação baseia-se em duas diferentes necessidades. De um lado existe a necessidade de uma linguagem mais robusta para a manipulação de bytes, pacotes e para implementar algoritmos que rodem um grande conjunto de dados. Nesse contexto C++, que é uma linguagem compilada e de uso tradicional, mostrou-se a ferramenta mais eficaz. De outro lado é fato que, durante o processo de simulação, ajustes são necessários com certa freqüência. Muda-se o tamanho do enlace e faz-se um teste, muda-se o atraso e faz-se um teste, acrescenta-se um nó e faz-se um teste. Enfim, haveria um desgaste muito grande se, a cada mudança de parâmetro, e elas são muitas em uma simulação, houvesse a necessidade de se compilar o programa para testá-lo. O uso da linguagem OTCL, que é interpretada, evita esse desgaste por parte do usuário, pois há uma simplificação no processo interativo de mudar e re-executar o modelo.

A noção de tempo no NS é obtida através de unidades de simulação que podem ser associadas, para efeitos didáticos, a segundos. A rede é construída usando nós os quais são conectados por meio de enlaces. Eventos são escalonados para passar entre os nós através dos enlaces. Nós e enlaces podem ter várias propriedades associadas a eles. Agentes podem ser associados aos nós e eles são responsáveis pela geração de diferentes pacotes. A fonte de tráfego é uma aplicação ao qual é associado um agente

particular. Os agentes precisam de um receptor que receberá seus pacotes. No caso do agente *tcp (transmission control protocol)* esse receptor chama-se *sink* (tanque) e tem a incumbência de gerar os pacotes de reconhecimento *(ACK - Acknowledg*e). No caso do agente *udp(user datagram protocol)* o receptor chama-se *null* (nulo).

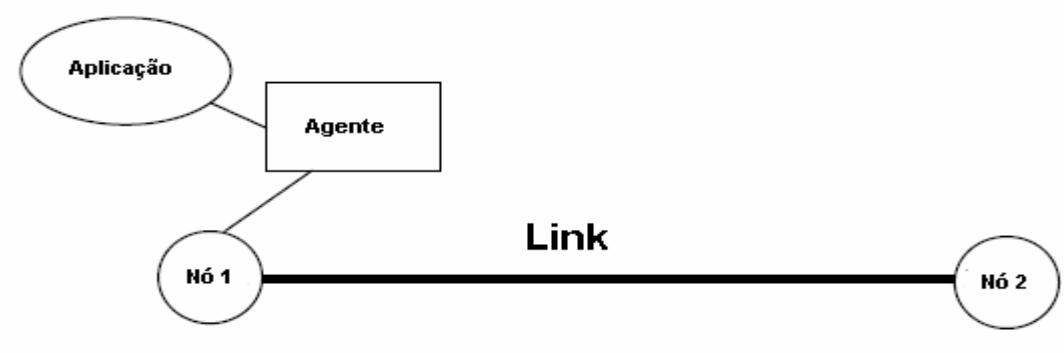

Figura 1 Estrutura dos objetos da simulação

2.1 CRIANDO UMA SIMULAÇÃO

Para criar uma simulação é preciso escrever um script OTcl, que será lido por um interpretador e gerará uma saída específica. Para facilitar a criação desse script, recomenda-se seguir o roteiro abaixo, que representa as principais etapas de construção de uma simulação, adaptando-o de acordo com o tipo de simulação pretendida:

- Criação do objeto simulador (escalonador de eventos);
- Abertura de arquivos para análise posterior (*trace*);
- Criação da topologia da rede (nós e enlaces);
- Criação dos agentes da camada de transporte e conexão com os nós;
- Criação dos geradores de tráfego e conexão com os agentes da camada de transporte;
- Programação dos eventos da simulação (dinâmica);
- Fechamento da simulação, animação (NAM) e geração de estatísticas.

A programação em OTcl requer conhecimentos sobre a sintaxe utilizada pelo NS. A seguir apresentaremos as principais classes que são comuns à maioria das simulações.

• Criação do objeto simulador (escalonador de eventos):

Para iniciar qualquer simulação, é preciso ajustar a variável que identifica o início da simulação, conforme a linha abaixo:

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

Esta linha define a geração de uma instância do objeto simulador e a associa com a variável 'ns', realizando as seguintes tarefas:

• Inicializa o formato dos pacotes;

• Cria um escalonador de eventos;

• Seleciona o formato padrão de endereçamento.

• Abertura de arquivos para análise posterior (trace):

A seguir, pode-se gerar um arquivo *trace* que servirá de base para a construção de uma animação gráfica, de acordo com o cenário utilizado. Essa é uma etapa opcional e pode ser eliminada, caso não seja necessário uma simulação gráfica.

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

A primeira linha acima cria um arquivo (ou abre, caso já exista um) com a extensão '*.nam*', que será lido pelo *Network Animator* (NAM), que é a aplicação responsável por gerar animações gráficas das simulações. A segunda linha diz ao simulador para gravar os passos da simulação no formato de entrada do NAM, ou seja, no arquivo '*out.nam*' gerado na linha anterior. Similarmente, pode-se usar a função '*trace-all*' para gravar arquivos de *trace* com informações em formato geral, normalmente utilizados para análises de simulações. As linhas abaixo mostram a sintaxe para este tipo de saída:

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

• Criação da topologia da rede (nós e enlaces):

A criação da topologia da rede é um dos pontos primordiais de uma simulação. É preciso que se faça um planejamento cuidadoso para que os nós e enlaces representem

um cenário mais realístico. As linhas abaixo definem a criação de 2 nós, chamados de n0 e n1.

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

O parâmetro [$ns node] define a criação de um nó para o objeto simulador '*ns*', criado no início da simulação. Em seguida, é necessário criar os enlaces para que os nós possam se comunicar. A sintaxe utilizada varia de acordo com a tecnologia de rede empregada. Utillizaremos neste exemplo um enlace *full-duplex* entre 2 nós:

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

A linha anterior define a criação de um enlace *full-duplex* (*duplex-link*) entre os nós n0 e n1, com capacidade de 1 Mbit/s e retardo de 10 ms. O último parâmetro define o tipo de fila utilizado, onde '*DropTail*' representa o algoritmo FIFO (*First In, First Out*).

Para facilitar a geração de múltiplos nós e enlaces, é possível criar um laço de repetição, conforme mostrado a seguir, onde a variável '*numNode*' define a quantidade de nós da simulação:

```
set numNode 8

# Criação dos nós
for {set i 0} {$i < $numNode} {incr i} {
set n($i) [$ns node]
}

# Criação dos enlaces
for {set i 0} {$i < $numNode} {incr i} {
for {set j [expr ($i + 1)]} {$j < $numNode} {incr j} {
$ns duplex-link $n($i) $n($j) 1Mb 10ms DropTail
}
}
```

A geração de tráfego no NS-2 é baseada em duas classes de objetos, a classe *Agente* e a classe *Aplicação*. Cada nó na rede que necessite enviar ou receber dados

deve ter um agente sobre ele. No topo de um agente roda uma aplicação. A aplicação determina que tipo de tráfego é simulado.

• Criação dos agentes da camada de transporte e conexão com nós:

Para que se possa realizar a implementação de protocolos de transporte (TCP e UDP) é necessária a criação de agentes, que são componentes da arquitetura NS responsáveis pela simulação destes protocolos. Os agentes criam um canal de comunicação entre os nós transmissor e receptor.

O código abaixo mostra a criação de um agente UDP, anexando-o ao nó n0 (emissor).

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

A seguir, deve-se criar um agente receptor (*Null)*, cuja função é apenas receber os pacotes enviados a ele. O código abaixo mostra a sua criação, anexando-o ao nó n1 (receptor).

```
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

A linha abaixo efetivamente estabelece o canal de comunicação (sessão) entre o emissor e o receptor, a nível de transporte.

```
$ns connect $udp0 $null0
```

• Criação dos geradores de tráfego e conexão com os agentes da camada de transporte:

Após o estabelecimento do canal de comunicação é necessário que um tráfego de uma determinada aplicação seja gerado e transmitido por este. O código abaixo exemplifica a criação de um tráfego CBR (*Constant Bit Rate*), que geralmente é utilizado para *streaming* de áudio e/ou vídeo. Alguns parâmetros são necessários, como o tamanho do pacote (em *bytes*), intervalo de transmissão (em segundos), entre outros.

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
```

```
$cbr0 set interval_ 0.005
```

Feito isso, é necessário anexar a aplicação criada a um agente da camada de transporte, conforme mostra a linha abaixo:

```
$cbr0 attach-agent $udp0
```

• Programação dos eventos da simulação (dinâmica):

Uma das etapas mais importantes em uma simulação é definir o comportamento que esta terá durante a sua execução (ou seja, os eventos que ocorrerão na simulação). Como o NS é um simulador orientado a eventos, a definição da dinâmica da simulação pode ser feita mais facilmente.

Para isso, atribui-se um tempo total de simulação e, durante este período, pode-se invocar eventos específicos, como início de tráfegos, quedas ou perdas de dados nos enlaces, entre outros.

No código a seguir, a aplicação CBR é iniciada no tempo 0.5s e finalizada em 4.5s. Após 5.0s, é chamado o procedimento '*finish*', que encerra a simulação.

```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
```

• Fechamento da simulação, animação (NAM) e geração de estatísticas:

Concluídas as etapas acima, é necessário declarar o fim da simulação, através do procedimento abaixo, que é responsável por fechar e limpar os arquivos de traces já existentes com o mesmo nome, abrir o NAM (caso este seja utilizado) e encerrar efetivamente a simulação. A última linha (*$ns run*) é responsável por executar a simulação e iniciar o escalonador de eventos do NS.

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

```
$ns run
```

• Iniciando a simulação:

Para executar a simulação usa-se o comando, abaixo, que executa a simulação e abre o NAM, caso este seja usado.

```
$ ns nome_do_script.tcl
```

2.2. CAMADA DE REDE E ROTEAMENTO

Para que ocorra a transferência de pacotes de um nó emissor para um nó receptor existe a necessidade de se determinar que caminho os pacotes devem seguir. Essa tarefa é realizada na camada de rede, mais precisamente pelos protocolos de roteamento da camada de rede. O núcleo do protocolo de roteamento é o algoritmo de roteamento, que tem a função de determinar uma rota adequada entre a origem e o destino dos datagramas.

O NS permite a simulação de diversos protocolos de roteamento, dentre os quais serão mostrados os mais utilizados atualmente, como o roteamento estático e os algoritmos dinâmicos, chamados de vetor de distância e estado de enlace.

2.2.1. Roteamento Unicast

O roteamento unicast é utilizado na comunicação ponto-a-ponto, quando um emissor cria um canal de comunicação separado para cada destinatário. Por exemplo, se a transmissão é feita entre um emissor e três destinatários, o emissor enviará três vezes o mesmo arquivo, um para cada destinatário.

Analisando como esta comunicação é feita, percebe-se que há uma sobrecarga do emissor. À medida que o número de receptores cresce, o congestionamento e o atraso de dados ficarão mais intensos.

2.2.1.1. Estático

No roteamento estático as rotas não mudam com muita freqüência e na maioria das vezes uma mudança é ocasionada pela intervenção humana. O roteamento estático é o mecanismo de definição de rotas padrão do NS, que utiliza o algoritmo SPF (Shortest Path First) de Dijkstra. A computação das rotas é feita uma única vez no início da simulação e para isso, o algoritmo utiliza uma matriz de adjacências e os valores de custos de todos os enlaces da topologia.

Para utilizar o roteamento estático, usa-se a sintaxe mostrada abaixo. Porém, esse comando é opcional já que esse algoritmo é o padrão usado no NS.

```
$ns rtproto Static
```

2.2.1.2. Dinâmico

No roteamento dinâmico as rotas podem ser alteradas quando há mudanças nas cargas de tráfego ou na topologia da rede. Esses algoritmos podem rodar periodicamente ou na ocorrência de mudanças na rede como, por exemplo, a topologia ou os custos dos enlaces. Serão abordados dois algoritmos de roteamento dinâmicos, o algoritmo de vetor de distâncias e o algoritmo de estado de enlace.

• Vetor de distância:

Nesse algoritmo, conhecido também como algoritmo de Bellman-Ford, cada roteador mantém uma tabela (vetor) que armazena a melhor distância para se chegar até cada destino e a rota correspondente. Inicialmente um roteador possui apenas informações de custos de enlaces até seus vizinhos diretamente conectados. Periodicamente, o roteador distribui seu vetor de distâncias aos seus vizinhos, atualizando, dessa forma, as tabelas de roteamento dos mesmos. Quando todas as tabelas de distâncias ficarem completas ocorre o que é chamado de convergência e o algoritmo entra em inatividade até que alguma mudança na rede aconteça. O algoritmo vetor de distâncias é dito descentralizado, pois um nó não tem conhecimento sobre os custos de todos os enlaces da rede, mas somente daqueles diretamente ligados a ele.

Para utilizar o roteamento por vetor de distância, usa-se a sintaxe mostrada abaixo:

```
$ns rtproto DV
```

Esse algoritmo apresenta problemas como o fato de poder convergir lentamente caso aconteçam alterações na rede, como por exemplo, o aumento no custo de um enlace ou a falha de um enlace.

• Estado de enlace:

O algoritmo de estado de enlace é um algoritmo global, uma vez que todos os custos de enlace são conhecidos por todos os nós. Cada nó da rede distribui informações

sobre os enlaces conectados diretamente à ele a todos os outros nós da rede através de uma transmissão *broadcast*.

Em seguida, um algoritmo de estado de enlace, como o algoritmo de Dijkstra, calcula o caminho de menor custo entre um nó e todos os outros nós da rede. Caso ocorra uma mudança na rede o algoritmo é executado novamente. O algoritmo de estado de enlace converge mais rapidamente que o algoritmo vetor de distância em caso de alterações na rede. Para utilizar o roteamento por estado de enlace, usa-se a sintaxe mostrada abaixo:

```
$ns rtproto LS
```

2.2.1.3. Utilizando vários caminhos simultaneamente:

Se partindo de um nó existem várias rotas possíveis para um destino particular, é possível fazer com que um tráfego seja dividido entre essas rotas. Para isso, deve-se, primeiramente, utilizar o algoritmo de roteamento vetor de distância. Em seguida, usa-se a sintaxe abaixo:

```
Node set multiPath_ 1
```

A linha acima define que todos os nós da topologia utilizarão esse mecanismo de roteamento, caso seja possível.

De forma alternativa, pode-se desejar que apenas um nó possua esse mecanismo. Se este for o caso, usam–se os comandos abaixo:

```
set n1 [$ns Node]
$n1 set multiPath_ 1
```

De acordo com o código acima apenas o nó ‘n1’ enviará dados para várias rotas simultaneamente.

Exemplo:

```
$ns set [new Simulator]
$ns rtproto DV
set n1 [$ns Node]
$n1 set multiPath_ 1
```

2.2.1.4. Atribuindo custos aos enlaces:

Para atribuir valores de custo a um enlace, utiliza-se o seguinte comando:

```
$ns cost $<node1> $<node1> <value>
```

Onde <value> é o valor do custo associado ao enlace entre os nós <node1> e <node2>. O valor padrão para os custos é 1.

Exemplo:

```
$ns cost $n1 $n2 10
```

2.2.1.5. Inserindo dinâmica na simulação:

Esta seção mostra como simular eventos que podem ocorrer em uma rede real, como a falha de um enlace.

• Falha de Enlace:

Para quebrar um enlace usa-se o seguinte comando:

```
$ns rtmodel-at <time> down $<node1> $<node2>
```

Onde <time> é o instante em segundos no qual o enlace entre os nós <node1> e <node2> vai falhar.

Exemplo:

```
$ns rtmodel-at 1.0 down $n0 $n1
```

2.3 CAMADA DE TRANSPORTE

Para simular protocolos da camada de transporte, o NS faz uso de agentes como o agente UDP e o agente TCP, este último podendo ser escolhido diversas variações (como Tahoe, Reno e Vegas) [3]. A seguir, serão abordados com mais detalhes os aspectos de configuração de cada um desses agentes.

2.3.1. UDP

O UDP é um protocolo da camada de transporte não orientado a conexão que não oferece garantias para a entrega de dados entre um emissor e um receptor, ou seja,

ele não suporta controle de fluxo ou congestionamento, correção de erros ou retransmissão.

Para criar um agente UDP, usa-se a seguinte sintaxe:

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

A primeira linha cria um agente UDP referenciado pela variável 'udp0'. A linha seguinte associa o agente criado a um nó chamado 'n0', que será o nó emissor de pacotes.

Em seguida, é necessário configurar um receptor, que pode ser feito de acordo com o exemplo abaixo:

```
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

O agente Null é responsável por apenas receber pacotes sem realizar nenhum tratamento adicional, e este é associado ao nó 'n1'.

Por fim, faz-se a conexão entre os dois agentes caracterizando a construção de um canal de comunicação entre o nó emissor e o nó receptor. O código abaixo ilustra exatamente essa conexão:

```
$ns connect $udp0 $null0
```

A classe Agent/UDP possui uma variedade de parâmetros que podem ser configurados, que variam desde o tamanho do pacote até o tipo de classe a qual ele pertence. A seguir mostramos alguns dos principais parâmetros utilizados no NS:

• Tamanho do pacote:

Esse parâmetro configura o tamanho do pacote.

Sintaxe:

```
$udp0 set packetSize_ <pktsize>
```

Onde <pktsize> é um número inteiro que representa o tamanho em bytes do pacote. O valor padrão é 1000.

Exemplo:

```
$udp0 set packetSize_ 500
```

• Identificador do fluxo de pacotes:

Serve para identificar o fluxo de pacotes que é originado de um agente UDP.

Sintaxe:

```
$udp0 set fid_ <fid-value>
```

Onde <fid-value> denota o valor que será o identificador do fluxo de pacotes. O valor padrão é 0.

Exemplo:

```
$udp0 set fid_ 1
```

• Tempo de vida do pacote:

Define o número máximo de nós por onde o pacote pode passar antes de ser descartado.

Sintaxe:

```
$udp0 set ttl_ <time-to-live>
```

Onde <time-to-live> é um inteiro que representa o valor do campo TTL. O valor padrão é 32.

Exemplo:

```
$udp0 set ttl_ 64
```

2.3.2 TCP

O protocolo TCP foi projetado para oferecer um fluxo de bytes fim-a-fim confiável. O TCP realiza controle de fluxo e congestionamento, retransmissão, entre outras funções. Ele ainda garante que os dados chegarão de forma correta e ordenada ao seu destino.

Para criar um agente TCP, usa-se a seguinte sintaxe:

```
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
```

A primeira linha cria um agente TCP referenciado pela variável 'tcp0'. A linha seguinte associa o agente criado a um nó chamado 'n0', que será o nó emissor de pacotes. Em seguida, é necessário configurar um receptor. Para o protocolo TCP o receptor é um agente do tipo TCPSink e o exemplo abaixo mostra como criá-lo:

```
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
```

Um agente do tipo TCPSink, que é responsável por enviar um ACK para cada pacote recebido ao emissor do tráfego TCP, servirá de destino para os pacotes e é associado ao nó 'n1'.

Assim como no UDP, também é necessária a conexão entre os dois agentes (TCP e TCPSink), conforme exemplificada abaixo:

```
$ns connect $tcp0 $sink0
```

A seguir serão abordados alguns dos principais parâmetros de configuração que compõem a classe Agent/TCP:

• Tamanho da janela:

Define o tamanho máximo da janela da conexão TCP.

Sintaxe:

```
$tcp0 set window_ <size>
```

Onde <size> define o tamanho da janela em número de pacotes. O valor padrão é 20.

Exemplo:

```
$tcp0 set window_ 50
```

• Tamanho do pacote:

Esse parâmetro configura o tamanho que o pacote vai possuir.

Sintaxe:

```
$tcp0 set packetSize_ <pktsize>
```

Onde <pktsize> é um número inteiro que representa o tamanho em bytes do pacote. O valor padrão é 1000.

Exemplo:

```
$tcp0 set packetSize_ 500
```

• Tamanho da janela de congestionamento:

Define o tamanho máximo da janela de congestionamento da conexão TCP.

Sintaxe:

```
$tcp0 set cwnd_ <size>
```

Onde <size> define o tamanho máximo da janela de congestionamento em número de pacotes. O valor padrão é 0.

Exemplo:

```
$tcp set cwnd_ 10
```

• Tamanho inicial da janela de congestionamento:

Define o tamanho inicial da janela de congestionamento da conexão TCP.

Sintaxe:

```
$tcp0 set windowInit_ <size>
```

Onde <size> define o tamanho inicial da janela de congestionamento em número de pacotes. O valor padrão é 1.

Exemplo:

```
$tcp0 set windowInit_ 5
```

• Tamanho máximo de rajada:

Determina o número máximo de pacotes que o emissor pode enviar em resposta a um ACK.

Sintaxe:

```
$tcp0 set maxburst_ <size>
```

Onde <size> define o número máximo de pacotes que um emissor pode enviar em resposta a um ACK. O valor padrão é 0.

Exemplo:

```
$tcp0 set maxburst_ 50
```

2.3.2.1. Variações do TCP

O NS permite a simulação de diversas variações do protocolo TCP. A seguir, serão abordadas 3 das principais variações.

• TCP Tahoe:

Essa versão do TCP tem como característica trabalhar com a técnica de partida lenta, ou seja, quando há uma perda, ele começa a retransmitir a uma taxa inicial lenta e vai crescendo exponencialmente. Essa é a variação padrão adotada pelo Agente TCP do NS e trabalha de forma similar à versão de TCP lançada com o sistema BSD4.3. A sintaxe utilizada para declarar essa variação é *Agent/TCP*.

• TCP Reno:

O agente TCP Reno é muito similar ao agente TCP Tahoe, exceto pelo fato de incluir uma recuperação rápida, onde a janela de congestionamento corrente é “inflada” pelo número de ACKs duplicados que o emissor TCP recebeu antes de receber um novo ACK. Esse “novo ACK” se refere a qualquer ACK com valor superior ao maior já visto até o momento. Além do mais, o agente TCP Reno não retorna ao estado de partida lenta durante uma retransmissão rápida. Ao invés disso, ele reduz a sua janela de congestionamento à metade da janela atual.

A sintaxe utilizada para declarar essa variação é *Agent/TCP/Reno*.

• TCP Vegas:

O TCP Vegas é uma proposta que surgiu para melhorar o desempenho das transmissões, aumentando a vazão do enlace em até 70% em alguns casos, além de

poder reduzir até à metade a quantidade de perdas se comparado ao TCP Reno. Ele tenta conseguir isso alterando o funcionamento de 3 mecanismos básicos: retransmissão, controle de congestionamento e partida lenta.

A sintaxe utilizada para declarar essa variação é *Agent/TCP/Vegas*.

2.4. CAMADA DE APLICAÇÃO

O NS possui diversas classes que simulam protocolos da camada de aplicação responsáveis por gerar esse tráfego. A seguir serão abordados os principais protocolos disponíveis no NS, de acordo com o protocolo de transporte utilizado (UDP ou TCP):

2.4.1. CBR (Constant Bit Rate)

O serviço CBR é aplicado a conexões que necessitam de banda fixa (estática) devido aos requisitos de tempo bastante limitados entre a origem e o destino. Aplicações típicas deste serviço são: áudio interativo (telefonia), distribuição de áudio e vídeo, áudio e vídeo on demand, e jogos interativos.

Para criar uma aplicação CBR, usa-se a seguinte sintaxe:

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
```

A primeira linha é responsável por gerar um tráfego CBR referenciado pela variável ‘cbr0’. A linha seguinte associa a aplicação criada a um protocolo da camada de transporte, que geralmente é o UDP, chamado de ‘udp0’.

A classe *Application/Traffic/CBR* possui uma variedade de parâmetros que podem ser configurados, conforme mostrados abaixo:

• Taxa de envio:

Esse parâmetro configura a taxa de envio dos pacotes.

Sintaxe:

```
$cbr0 set rate_ <value>
```

Onde <value> corresponde à taxa de transferência dos pacotes (em bits/s). O valor padrão é 448Kb.

Exemplo:

```
$cbr0 set rate_ 64Kb
```

• Intervalo entre pacotes:

Esse parâmetro configura o intervalo de envio entre os pacotes e não deve ser usado em conjunto com a taxa de envio (rate_).

Sintaxe:

```
$cbr0 set interval_ <value>
```

Onde <value> corresponde ao tempo do intervalo entre os pacotes (em segundos).

Exemplo:

```
$cbr0 set interval_ 0.005
```

• Tamanho do pacote:

Esse parâmetro configura o tamanho que o pacote vai possuir.

Sintaxe:

```
$cbr0 set packetSize_ <pktsize>
```

Onde <pktsize> é um número inteiro que representa o tamanho em bytes do pacote. O valor padrão é 210.

Exemplo:

```
$cbr0 set packetSize_ 48
```

2.4.2. Exponencial

O serviço Exponencial gera tráfegos em períodos pré-determinados, chamados de “ligado” e “desligado”. No período “ligado”, os pacotes são gerados a uma taxa constante, enquanto que no “desligado” não é gerado tráfego.

Para criar uma aplicação Exponencial, utiliza-se a seguinte sintaxe:

```
set exponencial0 [new Application/Traffic/Exponential]
$exponencial0 attach-agent $udp0
```

A primeira linha é responsável por gerar um tráfego Exponencial referenciado pela variável ‘exponencial0’. A linha seguinte associa a aplicação criada a um protocolo da camada de transporte, que nesse caso é o UDP, chamado de ‘udp0’.

A classe Application/Traffic/Exponential possui uma variedade de parâmetros que podem ser configurados, conforme mostrados abaixo:

• Taxa de envio:

Esse parâmetro configura a taxa de envio dos pacotes.

Sintaxe:

```
$exponencial0 set rate_ <value>
```

Onde <value> corresponde à taxa de transferência dos pacotes (em bits/s). O valor padrão é 64Kb.

Exemplo:

```
$exponencial0 set rate_ 128Kb
```

• Tamanho do pacote:

Esse parâmetro configura o tamanho que o pacote vai possuir.

Sintaxe:

```
$exponencial0 set packetSize_ <pktsize>
```

Onde <pktsize> é um número inteiro que representa o tamanho em bytes do pacote. O valor padrão é 210.

Exemplo:

```
$exponencial0 set packetSize_ 500
```

• Tempo de transmissão:

Esse parâmetro configura a média de tempo em que o gerador de tráfego permanece no período “ligado”.

Sintaxe:

```
$exponencial0 set burst_time_ <value>
```

Onde <value> corresponde ao tempo de geração de tráfego (em segundos). O valor padrão é 0,5s.

Exemplo:

```
$exponencial0 set burst_time_ 0.5
```

• Tempo de inatividade:

Esse parâmetro configura a média de tempo em que o gerador de tráfego permanece no período “desligado”.

Sintaxe:

```
$exponencial0 set idle_time_ <value>
```

Onde <value> corresponde ao tempo sem geração de tráfego (em segundos). O valor padrão é 0,5s.

Exemplo:

```
$exponencial0 set idle_time_ 0.5
```

2.4.3. FTP (File Transfer Protocol)

O serviço FTP é aplicado a transferência confiável de arquivos, uma vez que requer garantias de entrega dos pacotes. Para criar uma aplicação FTP, usa-se a seguinte sintaxe:

```
set ftp0 [new Application/ FTP]
$ftp0 attach-agent $tcp0
```

A primeira linha é responsável por gerar um tráfego FTP referenciado pela variável ‘ftp0’. A linha seguinte associa a aplicação criada a um protocolo da camada de transporte, que é o TCP, chamado de ‘tcp0’.

Dentre os parâmetros que podem ser configurados na classe *Application/FTP*, pode-se destacar:

• Número máximo de pacotes:

Esse parâmetro define o número máximo de pacotes gerados pela fonte (emissor).

Sintaxe:

```
$ftp0 set maxpkts <value>
```

Onde <value> corresponde à quantidade de pacotes.

Exemplo:

```
$ftp0 set maxpkts 1000
```

2.4.4. Telnet

O serviço Telnet é utilizado para acesso remoto e exige uma transferência confiável de dados. Para criar uma aplicação Telnet, utiliza-se a seguinte sintaxe:

```
set telnet0 [new Application/Telnet]
$telnet0 attach-agent $tcp0
```

A primeira linha é responsável por gerar um tráfego Telnet referenciado pela variável ‘telnet0’. A linha seguinte associa a aplicação criada a um protocolo da camada de transporte, que nesse caso é o TCP, chamado de ‘tcp0’. A classe *Application/Telnet* possui apenas um parâmetro que pode ser configurado, conforme mostrado abaixo:

• Intervalo entre pacotes:

Esse parâmetro configura o tempo médio entre chegadas de pacotes.

Sintaxe:

```
$telnet0 set interval_ <value>
```

Onde <value> corresponde ao intervalo entre os pacotes (em segundos). O valor padrão é 1,0s.

Exemplo:

```
$telnet0 set interval_ 0.005
```

2.4.5. Gerando e finalizando tráfegos

Para iniciar e parar os tráfegos das aplicações, utiliza-se os comandos abaixo:

`$ns at <tempo> “$<protocolo> start”` - Inicia o tráfego no instante denotado por <tempo>.

`$ns at <tempo> “$<protocolo> stop”` - Encerra o tráfego no instante denotado por <tempo>.

Onde <protocolo> representa a variável que referencia um protocolo de aplicação criado anteriormente e <tempo> refere-se a um determinado tempo de simulação em segundos.

Exemplo:

```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
```

2.5. REDES SEM FIO

O NS permite simulação de redes sem fios sendo possível configurar diversos parâmetros, como tipo de antena, propagação, protocolo de roteamento, entre outros.

2.5.1. Redes Ad-hoc

As redes Ad-Hoc não possuem um controle centralizado, com pontos de acesso, e são formadas quando existe uma necessidade de comunicação entre nós próximos. A seguir será mostrado como configurar parâmetros e funções necessárias à simulação desse tipo de rede.

Um nó móvel é constituído de diversos componentes de rede, como tipo de camada de enlace, tipo de fila, subcamada MAC, o canal sem fio, antena, tipo de rádio-propagação, protocolo de roteamento Ad-Hoc, etc. A primeira etapa é a definição desses parâmetros, que é exemplificada a seguir:

```
set val(chan) Channel/WirelessChannel
set val(prop) Propagation/TwoRayGround
set val(netif) Phy/WirelessPhy
set val(mac) Mac/802_11
```

```
set val(ifq) Queue/DropTail/PriQueue
set val(ll) LL
set val(ant) Antenna/OmniAntenna
set val(ifqlen) 100
set val(nn) 3
set val(rp) AODV
```

Uma descrição desses parâmetros é explicada abaixo:

• Tipo de Canal:
Define o meio físico que vai ser utilizado. Para redes sem fio utiliza-se *Channel/WirelessChannel*.

• Modelo de Rádio-Propagação:
Pode assumir os tipos: *FreeSpace* e *TwoRayGround*. O modelo *FreeSpace* é adequado para distâncias curtas enquanto que o modelo *TwoRayGround* geralmente é usado para distâncias longas.

• Interface de Rede:
Atua como uma interface de hardware através da qual o nó móvel acessa o canal e é implementada pela classe *Phy/WirelessPhy*.

• Subcamada MAC:
Usa-se o protocolo IEEE 802.11 implementado pela classe *Mac/802_11*.

• Tipo de Fila:
Geralmente utiliza-se uma fila de prioridade (*Queue/Droptail/PriQueue*), onde os pacotes de roteamento possuem uma prioridade maior. No exemplo acima, define-se o tamanho máximo da fila em 100 pacotes.

• Camada de Enlace:
Usa-se o valor padrão *LL*.

• Antena:
Uma antena omni-direcional (*Antenna/OmniAntenna*) é usada pelos nós móveis.

• Número de nós móveis:
Neste exemplo, foi utilizado o valor três (três nós móveis).

Simulações de redes sem fio necessitam da criação de um arquivo de *trace* genérico, mostrada abaixo:

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

Neste exemplo, é criado um arquivo de *trace* denominado ‘out.tr’ referenciado pela variável ‘tf’. O arquivo de *trace* do NAM precisa registrar os movimentos dos nós. Para isso, usa-se o comando “`namtrace-all-wireless`” exemplificado abaixo:

```
set nf [open out.nam w]
$ns namtrace-all-wireless $nf 500 500
```

As linhas acima definem a criação de um arquivo de *trace* do NAM chamado ‘out.nam’ referenciado pela variável ‘nf’ que conterá todas as informações de mobilidade em uma rede que possui uma área de atuação de 500m x 500m. Em seguida, definem-se as dimensões da área de atuação dos nós móveis (topografia), como no exemplo abaixo:

```
set topo [new Topography]
$topo load_flatgrid 500 500
```

Neste caso, foi criada uma topografia de 500m x 500m. O próximo passo é a criação de um objeto denominado GOD (General Operations Director), mostrada a seguir:

```
create-god $val(nn)
```

Onde ‘$val(nn)’ representa o número de nós móveis que foi definido anteriormente. O GOD é responsável por armazenar informações sobre o ambiente, a rede, os nós. É um observador onipresente, mas que não é conhecido pelos outros participantes da simulação. Para criar os nós móveis é necessário redefinir a maneira como um objeto nó é construído, ou seja, ajustar a API de configuração do nó onde se

pode determinar o tipo de endereçamento, o protocolo de roteamento Ad-Hoc, camada de enlace, camada MAC, etc. Esse processo é ilustrado a seguir:

```
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace ON \
-movementTrace OFF \
-channel [new $val(chan)]`
```

Os valores desses parâmetros serão substituídos pelos valores das variáveis ‘$val’ e outras definidas anteriormente. Depois, criam-se os nós:

```
set n(0) [$ns node]
set n(1) [$ns node]
set n(2) [$ns node]
```

Em seguida, os nós são inicializados desabilitando movimentos aleatórios nos mesmos:

```
$n(0) random-motion 0
$n(1) random-motion 0
$n(2) random-motion 0
```

As posições iniciais dos nós são determinadas a seguir e exemplificadas abaixo:

```
$n(0) set X_ 200.0
$n(0) set Y_ 100.0
$n(0) set Z_ 0.0
$n(1) set X_ 100.0
$n(1) set Y_ 100.0
```

```
$n(1) set Z_ 0.0
$n(2) set X_ 100.0
$n(2) set Y_ 10.0
$n(2) set Z_ 0.0
```

É necessário definir a posição inicial dos nós no NAM. Isso é feito de acordo com a seguinte sintaxe:

```
$ns initial_node_pos <node> <size>
```

Onde <size> define o tamanho do nó no NAM.

Exemplo:

```
for {set i 0} {$i < $val(nn)} {incr i} {
$ns initial_node_pos $n($i) 50
}
```

Para dar mobilidade aos nós usa-se a sintaxe abaixo que é modelada como um evento:

```
$ns at <time> $<node> setdest <pos_x> <pos_y> <vel>
```

Determinando que no instante <time> o nó <node> irá se mover para a posição (<pos_x> <pos_y) à uma velocidade de <vel> m/s.

Exemplo:

```
$ns at 0.4 "$n(2) setdest 300.0 350.0 600.0"
```

É preciso indicar ao NAM o instante em que a simulação termina, utilizando a sintaxe abaixo (na forma de evento):

```
$ns at <time> "$ns nam-end-wireless <stop_time>"
```

Onde <time> é um instante de tempo qualquer e <stop_time> é o tempo onde a simulação vai terminar.