UNIVERSIDADE FEDERAL FLUMINENSE

Uso do NS-2 na Disciplina de Comunicação de Dados

Lucas Coelho Gonçalves e Marcos Estevo de Oliveira Corrêa

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).

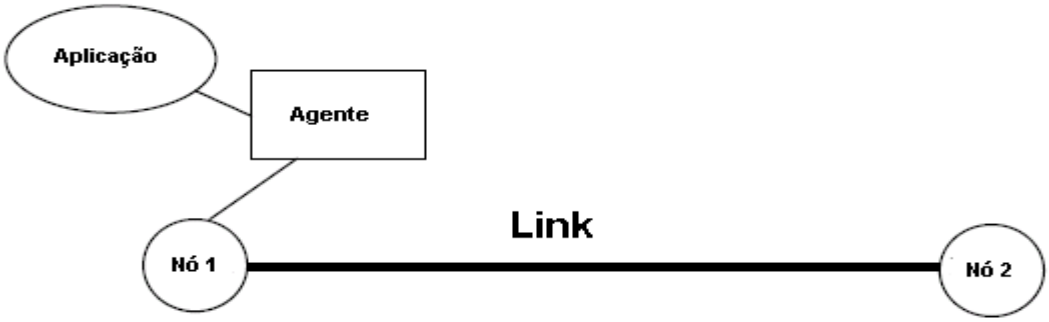


Figura 1 Estrutura dos objetos da simulação

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.

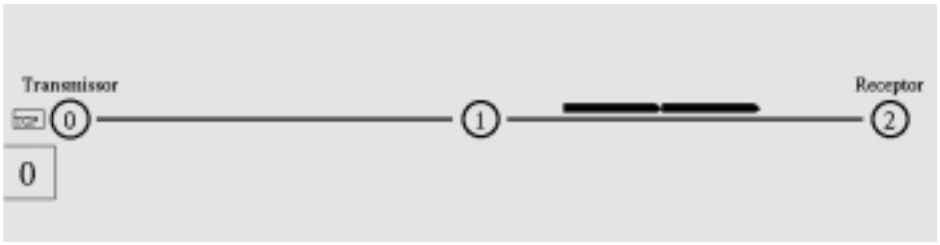
3. SIMULAÇÕES IMPLEMENTADAS

3.1. SIMULAÇÃO 1

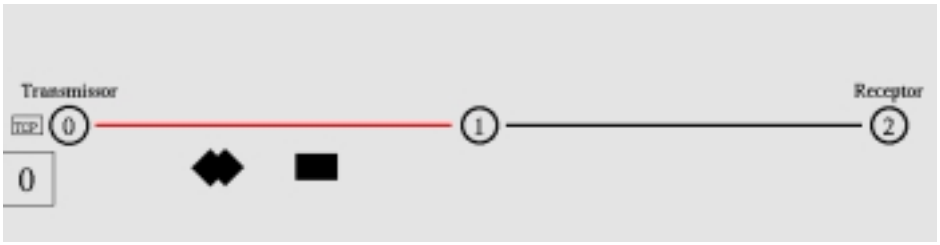
Esta simulação tem como objetivo mostrar o mecanismo de controle de fluxo janela deslizante, do protocolo TCP. A topologia é composta por três nós. O nó transmissor se conecta ao nó receptor através de um nó intermediário. O enlace entre o nó transmissor e o nó intermediário tem capacidade maior que o enlace entre o nó intermediário e o nó receptor, a fim de gerar uma fila de pacotes no nó intermediário. Quando a capacidade da fila é excedida, o último pacote a entrar é descartado. Esse método de gerenciamento de fila é conhecido como FIFO (*First In, First Out*). É possível acompanhar, durante a animação, o crescimento da fila e o descarte dos pacotes. Como há descarte de pacotes, o transmissor não recebe os pacotes de reconhecimento (*ACK*) e, depois de excedido um certo limite de tempo, há retransmissão do pacote descartado e redução do tamanho da janela de congestionamento.

O tamanho da janela de congestionamento é visualizado através de um monitor de variáveis, configurado para monitorar a variável "**cwnd_**".

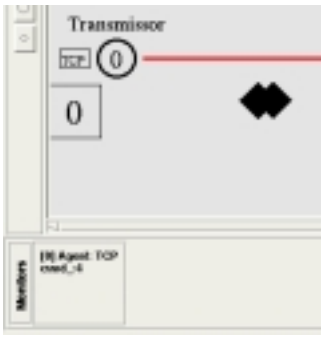
Em t = 1,2 segundos, é simulada uma falha entre o nó zero e nó um, acarretando perda de pacotes.



(a) O nó zero transmite pacotes ao nó dois



(b) O enlace entre o nó zero e o nó um falha, provocando perda de pacotes



(c) No canto inferior esquerdo é possível acompanhar o tamanho da janela de congestionamento

Figura 2 Exemplo de simulação usando TCP

Alterando um pouco essa simulação, fizemos uma comparação entre três implmentações do TCP. Nesta nova simulação, retiramos a falha do enlace e a aplicação FTP gerava um total de 500 pacotes. O tempo total de simulação é de 30 segundos. Repetimos a simulação para cada tipo de TCP, usando a mesma topologia.

Com a utilização de uma ferramenta de modelagem gráfica como Xgraph é possível desenhar um gráfico da evolução do tamanho da janela de congestionamento do TCP durante a transmissão.

Geramos este gráfico para três implementações do TCP.

Primeiro para o TCP Tahoe:

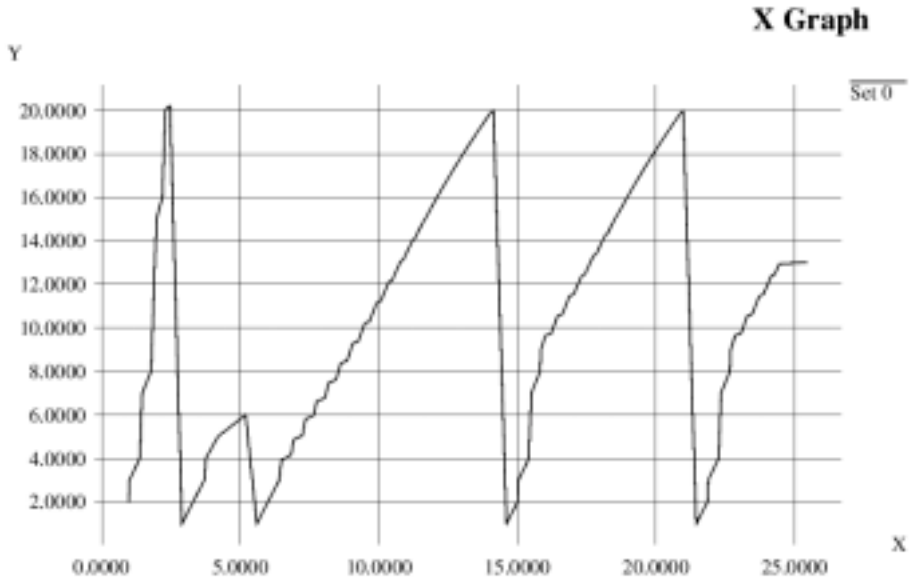


Figura 3 Gráfico da janela de congestionamento do protocolo TCP para a implementação Tahoe.

Depois para o TCP Reno:

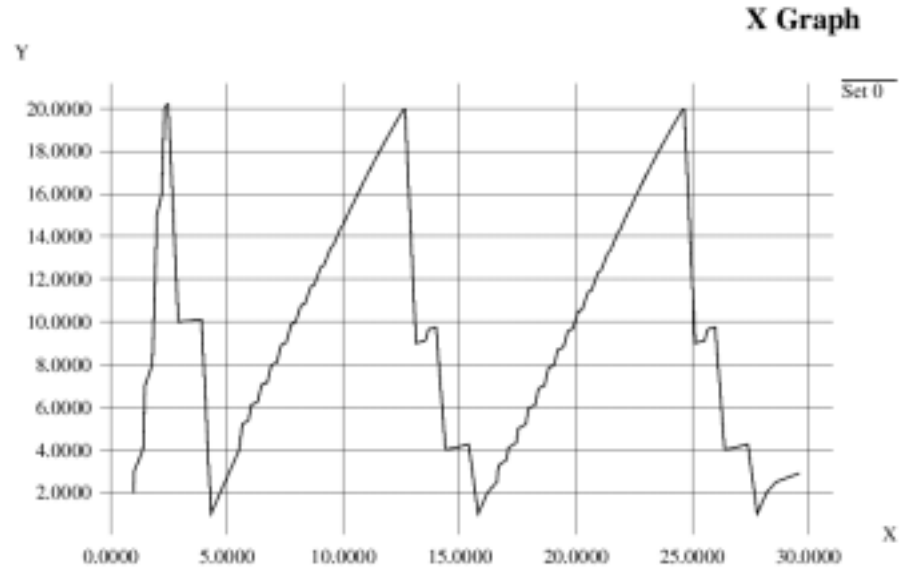


Figura 4 Gráfico da janela de congestionamento do protocolo TCP para a implementação Reno.

E por último, para o TCP Vegas:

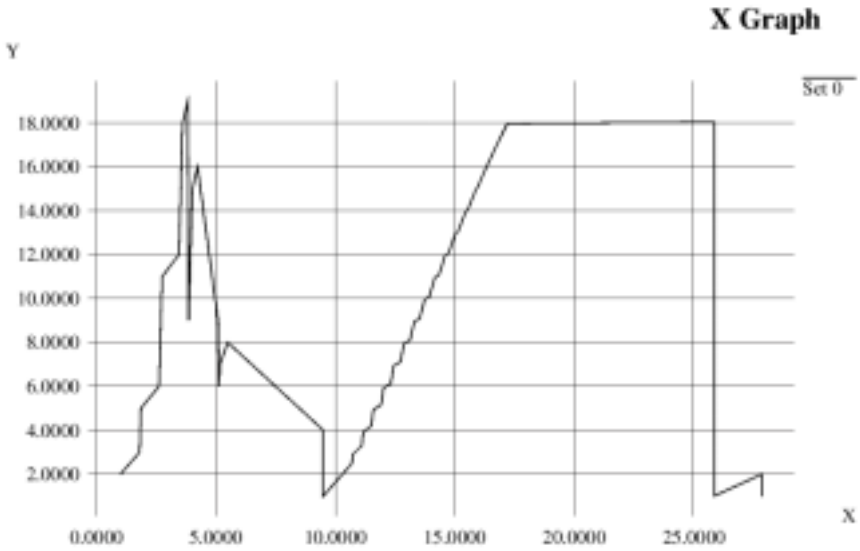


Figura 5 Gráfico da janela de congestionamento do protocolo TCP para a implementação Vegas.

Analisando os arquivos de *trace* podemos verificar que o TCP Reno apresentou 13 perdas de pacote, o TCP Tahoe apresentou 9 perdas de pacote, e o TCP Vegas apresentou apenas 6 perdas de pacote.

3.2. SIMULAÇÃO 2

Esta simulação tem dois objetivos principais. O primeiro é mostrar que pacotes do mesmo fluxo podem percorrer rotas diferentes dentro de uma rede comutada por pacotes e não orientada a conexão. O segundo é mostrar a multiplexação dos canais, ou seja, dois ou mais fluxos compartilhando o mesmo canal.

Esta topologia é composta por sete nós e sete enlaces. Todos os enlaces têm capacidade de transmissão de 300 Kbps e retardo de 40 milissegundos. Duas aplicações rodam nessa topologia: uma aplicação CBR (*Constant Bit Rate*), tendo o nó um como transmissor e o nó sete como receptor, e uma aplicação FTP (*File Transfer Protocol*) tendo o nó zero como transmissor e o nó sete como receptor.

A aplicação CBR gera pacotes de tamanho 1Kbyte. A taxa de transmissão é de 200 Kbps. Essa aplicação utiliza um agente UDP, cujo fluxo é identificado pela cor laranja. A aplicação CBR inicia a geração de tráfego em t = 0,1 segundos e termina em t = 4,9 segundos.

A aplicação FTP, rodando sobre um agente TCP, gera pacotes de 1Kbyte e o tamanho máximo da janela de congestionamento é de três pacotes. Esse fluxo é identificado pela cor azul. A aplicação FTP inicia a geração de tráfego em t = 0,2 segundos e termina em t = 4,5 segundos.

O comando **“$n2 set multiPath_ 1**” força o nó dois a utilizar múltiplos caminhos no encaminhamento dos pacotes. Dessa forma os dois caminhos possíveis até o nó sete são utilizados.

Em t = 1,15 segundos, o enlace entre os nós dois e cinco falha. O nó dois automaticamente redireciona todo o fluxo para o nó três. O enlace é restabelecido em t = 3,0 segundos, e o nó dois volta a usar esse enlace para direcionar os pacotes. A simulação é encerrada em t = 5,0 segundos.

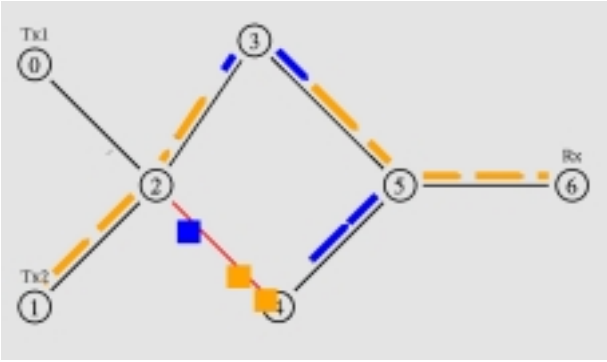


Figura 6 Topologia da simulação 2

3.3. SIMULAÇÃO 3

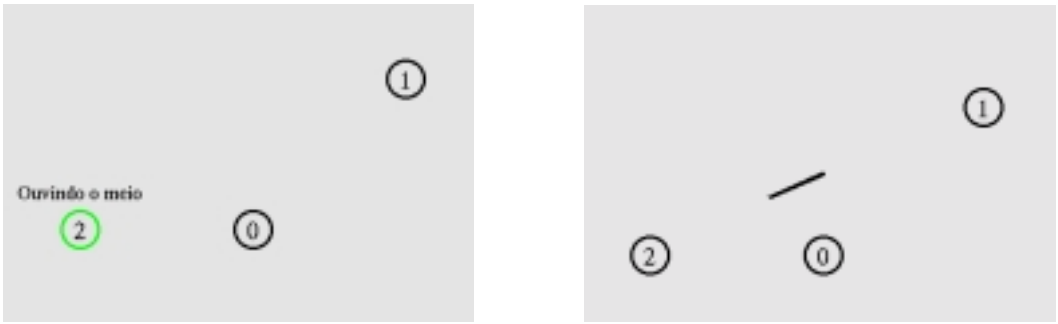
Esta simulação tem como objetivo ilustrar o protocolo de acesso CSMA/CA usado em uma rede sem fio e o problema da estação escondida (*hidden terminal*), mostrando uma das possíveis soluções. Este exemplo se divide em duas partes.

Parte 1 – Alguns cenários de transmissão

O problema da estação escondida ocorre quando duas estações fora do alcance uma da outra querem transmitir para uma única estação receptora, dentro do alcance das duas. Como as estações transmissoras estão fora do alcance uma da outra, não conseguem verificar que o meio está sendo utilizado e podem transmitir simultaneamente, causando colisão e perda de pacotes.

CASO 1: Sem contenção

Nesta primeira situação apenas um nó deseja transmitir, portanto não há contenção. O transmissor ouve o meio, que não está sendo utilizado neste momento, e transmite os dados.



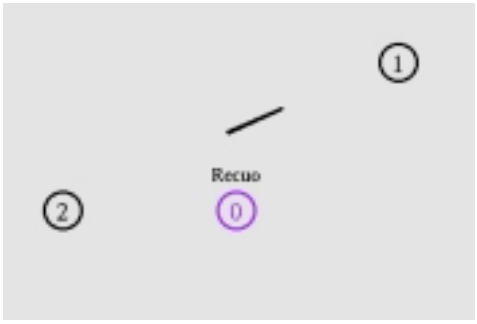
(a) Nó dois verifica disponibilidade do meio **(b)** Nó dois envia dados ao nó um

Figura 7 Transmissão sem contenção

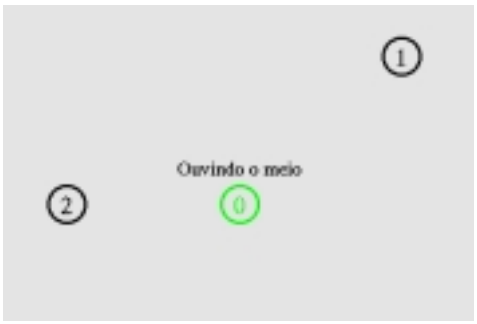
CASO 2: Recuo

Neste caso, o nó zero deseja transmitir, mas encontra o meio ocupado pelo nó dois, que está transmitindo para o nó um. O nó zero então recua, e espera o tempo estimado no pacote RTS para ouvir o meio novamente. Ele encontra o meio desocupado e agora pode transmitir.

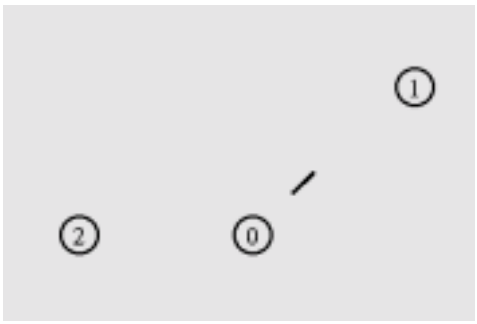
(a) Nó zero verifica disponibilidade do meio enquanto nó dois transmite



(b) Nó zero recua ao verificar que o meio está sendo utilizado



(c) Nó zero testa disponibilidade do meio novamente

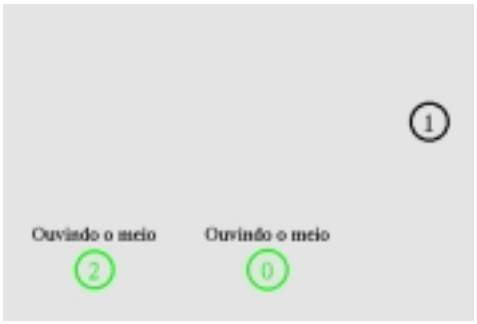


(d) Nó zero finalmente realiza transmissão

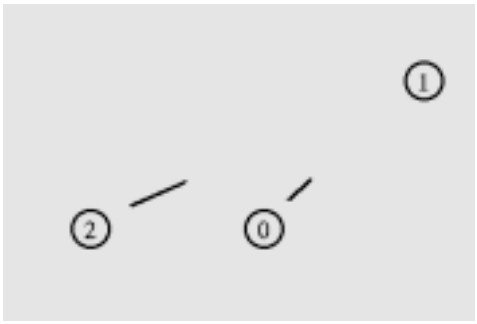
Figura 8 Situação de Recuo

CASO 3: Colisão quando os nós transmissores ouvem o meio ao mesmo tempo

Nesta situação, os dois nós transmissores estão dentro do alcance um do outro, mas ouvem o meio ao mesmo tempo. Dessa forma, os dois percebem o meio livre e transmitem ao mesmo tempo. Os pacotes chegam ao mesmo tempo no receptor, que identifica a colisão.



(a) Os dois nós ouvem o meio simultaneamente



(b) Os dois nós iniciam transmissão simultaneamente

(c) A chegada simultânea dos dois pacotes provoca colisão no nó um

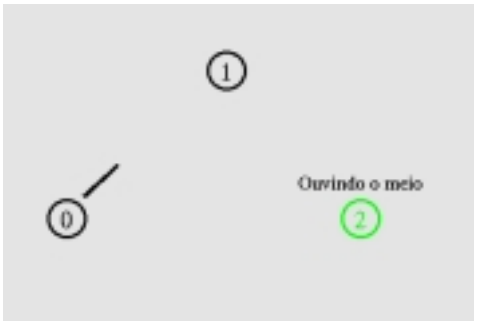
Figura 9 Colisão por verificação de disponibilidade do meio simultânea

CASO 4: Colisão em cenário de estação escondida

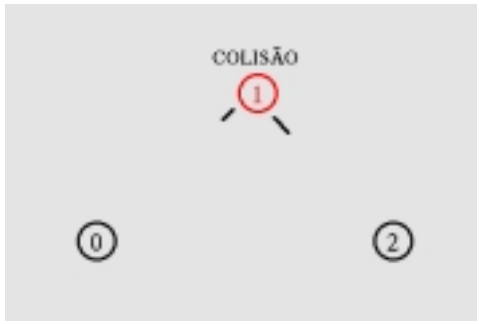
O nó dois se move e permanece ao alcance do nó um, mas fica fora do alcance do nó zero. O nó zero ouve o meio e, verificando que este está livre, transmite para o nó um. O nó dois ouve o meio enquanto o nó zero está transmitindo. Por estar fora do alcance do nó zero, o nó dois encontra o meio livre e também transmite. Os pacotes chegam ao mesmo tempo no receptor, provocando colisão.

(a) O nó zero e o nó dois estão fora do alcance um do outro

(b) O nó dois verifica a disponibilidade do meio



(c) Nó dois verifica disponibilidade do meio enquanto nó zero transmite



(d) A transmissão do nó zero não é percebida pelo nó dois, que inicia sua transmissão e provoca colisão no nó um

Figura 10 Colisão em cenário de estação escondida

Parte 2 – Utilização dos pacotes de controle RTS e CTS

O pacote RTS (*Request to Send*) atua de duas formas: reservando o meio para transmissão e verificando se a estação de destino está preparada para recepção. Esse pacote contém uma estimativa do tempo necessário para a futura transmissão.

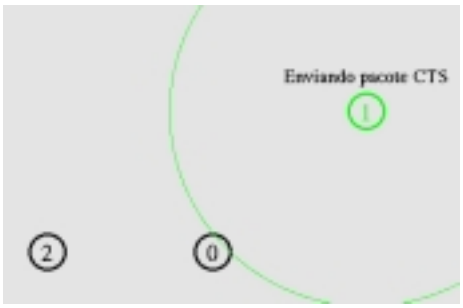
A estação receptora, em resposta ao RTS, envia um pacote CTS (*Clear to Send*) indicando que está preparada para a recepção. Só então, após receber o pacote CTS, o transmissor inicia o envio dos dados.

CASO 1: Sem contenção

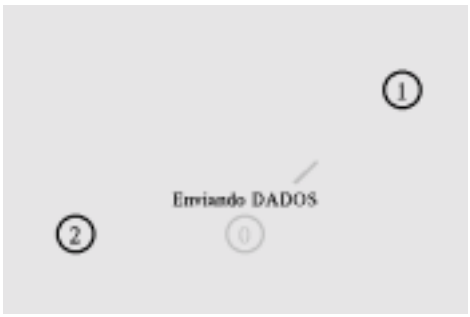
No primeiro caso não há contenção, pois apenas o nó zero deseja transmitir. Esse nó ouve o meio, envia um pacote RTS e recebe um pacote CTS, vindo do nó um. Agora o nó zero pode iniciar a transmissão dos dados. Após receber o pacote de dados, o nó um envia um pacote de reconhecimento (*ACK*).

(a) início do processo de transmissão

(b) reserva do meio e aviso à estação receptora



(c) confirmação do pedido de transmissão



(d) envio dos dados

Figura 11 Transmissão sem contenção

CASO 2: Recuo devido a pacote RTS

O nó zero deseja transmitir para o nó um, então, ouve o meio e envia um pacote RTS. O nó dois deseja transmitir para o nó zero, e ao ouvir o meio e recebe o pacote RTS enviado pelo nó zero ao nó um. Ao receber este pacote o nó dois recua, ou seja, adia por um tempo especificado no RTS. O nó um envia um pacote CTS para o nó zero,

que inicia a transmissão. Após finalizada a transmissão o nó um envia um pacote de reconhecimento, indicando que a transmissão foi realizada com sucesso. O nó dois faz nova verificação do meio e, ao encontrá-lo livre, inicia sua transmissão para o nó zero.

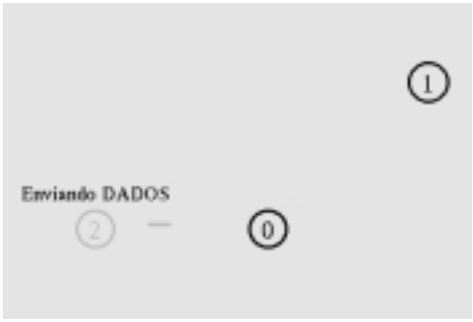
(a) nó zero reserva o meio para transmissão ao nó um

(b) nó dois recua ao receber pacote RTS endereçado ao nó um

(c) nó zero envia dados

(d) nó um confirma recebimento dos dados

(e) nó dois inicia processo de transmissão com o nó zero



(f) nó dois finalmente envia dados

Figura 12 Recuo devido a pacote RTS

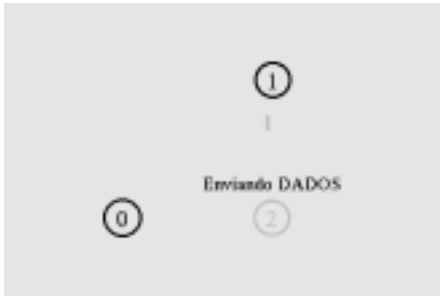
CASO 3: Recuo devido a pacote RTS

Este caso se assemelha ao anterior. Agora, porém, o nó dois se moveu, e os três nós estão ao alcance uns dos outros e o nó dois deseja transmitir para o nó um. Novamente o nó dois recua devido ao recebimento de um pacote RTS.

(a) nó dois recebe pacote RTS

(b) nó dois aguarda a transmissão entre o nó zero e o nó um

(c) nó dois envia pacote RTS ao nó um



(d) nó dois envia dados

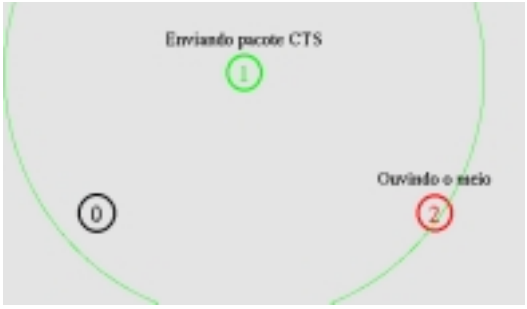
Figura 13 Outro caso de recuo devido a pacote RTS

CASO 4: Recuo devido a pacote CTS

O nó dois se movimenta até ficar fora do alcance do nó zero, mas ainda dentro do alcance do nó um. O nó zero inicia o processo de transmissão enviando um pacote RTS ao nó um. O nó dois também deseja transmitir para o nó um e verifica se meio está ocupado. O pacote RTS enviado pelo nó zero não é recebido pelo nó dois, pois estes nós estão fora do alcance um do outro. O nó um responde com um pacote CTS. O nó dois ao verificar se o meio está ocupado recebe o pacote CTS enviado pelo nó um e recua, concluindo que o meio será ocupado por outra transmissão. Após o tempo especificado no pacote CTS, o nó dois ouve novamente o meio e, ao encontrá-lo livre, inicia o processo de transmissão.

Este é cenário de estação escondida. Podemos observar que o uso dos pacotes de controle RTS e CTS foi suficiente para que a transmissão ocorresse sem colisão.

(a) pedido de estabelecimento de transmissão entre



(b) confirmação do pedido de transmissão

o nó zero e o nó um

(c) envido de dados pelo nó zero

(d) pedido de estabelecimento de transmissão entre o nó dois e o nó 1

Figura 14 Transmissão bem sucedida em cenário de estação escondida.

4. CONCLUSÃO

Acreditamos que estes simples exemplos que escolhemos poderão auxiliar os alunos da disciplina de comunicação de dados a consolidar a teoria discutida em sala de aula.

Através das simulações, os alunos poderão compreender melhor a dinâmica do funcionamento das redes de computadores. É evidente que as animaçõesajudam a concretizar os conceitos aprendidos nas aulas teróricas da discplina.

A linguagem utilizada para escrever os scripts Tcl é intuitiva e relativamente simples. Dessa forma, os exemplos mostrados neste trabalho podem ser modificados pelos alunos de acordo com as suas necessidades. Assim, este trabalho poderá ser continuado pelas próprias experimentações dos alunos, criando novos cenários para simulação.

Uma das contribuições deste projeto é a disponibilização de um resumo contendo os principais elementos de simulação suportados pelo NS-2. Com esse resumo os alunos poderão dar os primeiros passos no sentido de fazer suas próprias simulações.

Durante o primeiro período de 2005, já começamos a realizar algumas experiências com a turma de Comunicação de Dados IV. Primeiramente foi dado um treinamento ao monitor da disciplina, e depois foi ministrada uma aula para a turma sobre o simulador. Além disso, a turma desenvolveu um trabalho utilizando NS-2 como ferramenta. O trabalho consistiu em realizar simulações com três implementações diferentes de TCP (Tahoe, Reno e Vegas) numa mesma topologia, e apresentar os gráficos com a evolução da janela de congestionamento para cada implementação do TCP.

O trabalho foi realizado com sucesso e a turma não teve dificuldades para gerar os gráficos. Houve, porém, a necessidade de nós e o monitor da disciplina prestarmos suporte no laboratório durante a execução do trabalho.

Dente os objetivos não alcançados nesse projeto, está a implementação da simulação de uma lan ethernet. Dedicamos tempo e esforço para solucionar este problema, porém, devido a limitações do NAM não conseguimos chegar a uma visualizão adequada. Fica com isso um espaço em aberto para a complementação deste trabalho.

5. REFERÊNCIAS BIBLIOGRÁFICAS

[1] **"*The Network Simulator*" – NS-2 – http://www.isi.edu/nsnam/ns**

[2] **OTcl and TclCL home**
http://otcl-tclcl.sourceforge.net/

[3] **TANENBAUM, Andrew S. – Computer Networks – 4ª Edição – Capítulo 6.5**

APÊNDICE – SCRIPTS DAS SIMULAÇÕES

SIMULAÇÃO 1

```
set ns [new Simulator]

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$ns at 0.0 "$n0 label Transmissor"
$ns at 0.0 "$n2 label Receptor"

set nf [open janela.nam w]
$ns namtrace-all $nf
set f [open janela.tr w]
$ns trace-all $f

$ns duplex-link $n1 $n0 1.5Mb 95ms DropTail
$ns duplex-link $n1 $n2 0.3Mb 90ms DropTail

$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right

$ns queue-limit $n1 $n2 5

$ns duplex-link-op $n1 $n2 queuePos 0.5

Agent/TCP set nam_tracevar_ true

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $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_

$ns at 0.2 "$ftp produce 500"
$ns at 30.0 "finish"

$ns rtmodel-at 1.2 down $n0 $n1
$ns rtmodel-at 1.3 up $n0 $n1

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

        exec nam janela.nam &
      exit 0
}

$ns run
```

SIMULAÇÃO 2

```
#Criando simulador de objeto
set ns [new Simulator]

#Definicao de cores dos fluxos
$ns color 1 Blue
$ns color 2 Orange

#Abrindo o NAM
set nf [open roteamento.nam w]
$ns namtrace-all $nf

#Definindo procedimento de termino
proc finish {} {
	global ns nf
	$ns flush-trace
	   #Fechamento do NAM
	   close $nf
	   #Executando o NAM
	exec nam roteamento.nam &
	exit 0
}

#Criacao dos nos
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$n2 set multiPath_ 1
$ns rtproto DV


#Criacao dos links
$ns duplex-link $n0 $n2 0.5Mb 40ms DropTail
$ns duplex-link $n1 $n2 0.5Mb 40ms DropTail
$ns duplex-link $n2 $n3 0.5Mb 40ms DropTail
$ns duplex-link $n2 $n4 0.5Mb 40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 40ms DropTail
$ns duplex-link $n4 $n5 0.5Mb 40ms DropTail
$ns duplex-link $n5 $n6 0.5Mb 40ms DropTail


#tamanho das fila: 10
$ns queue-limit $n0 $n2 10
```

```
$ns queue-limit $n1 $n2 10
$ns queue-limit $n2 $n4 10
$ns queue-limit $n3 $n5 10
$ns queue-limit $n4 $n5 10
$ns queue-limit $n5 $n6 10
$ns queue-limit $n2 $n3 10

#posicionando os nos
$ns duplex-link-op $n0 $n2 orient 315deg
$ns duplex-link-op $n1 $n2 orient 45deg
$ns duplex-link-op $n2 $n4 orient right-down
$ns duplex-link-op $n3 $n5 orient right-down
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n5 $n6 orient right
$ns duplex-link-op $n2 $n3 orient 60deg

#monitores de fila
$ns duplex-link-op $n0 $n2 queuePos 0.5
$ns duplex-link-op $n1 $n2 queuePos 0.5
$ns duplex-link-op $n2 $n4 queuePos 0.5
$ns duplex-link-op $n3 $n5 queuePos 0.5
$ns duplex-link-op $n4 $n5 queuePos 0.5
$ns duplex-link-op $n5 $n6 queuePos 0.5
$ns duplex-link-op $n2 $n3 queuePos 0.5



Agent/TCP set nam_tracevar_ true

#Estabelecendo conexao TCP (no2 e no7)
set tcp [new Agent/TCP]
$tcp set window_ 5
$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n6 $sink
$ns connect $tcp $sink
$tcp set fid_ 1


#Estabelecendo uma conexão UDP(no2 e no7)
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n6 $null
$ns connect $udp $null
$udp set fid_ 2

#Estabelecendo uma aplicacao FTP sobre uma conexao TCP
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
$ftp set type_ FTP


#Estabelecendo uma aplicação CBR sobre UDP
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.4mb
$cbr set random_ false

$ns rtmodel-at 1.67 down $n2 $n4
$ns rtmodel-at 2.0  up  $n2 $n4

#Definindo os labels
$ns at 0.0 "$n0 label Tx1"
$ns at 0.0 "$n1 label Tx2"
$ns at 0.0 "$n6 label Rx"



#narrando simulacao
$ns at 0.0 "$ns trace-annotate \"flooding\""
$ns at 0.4 "$ns trace-annotate \"Inicio da transmissao FTP\""
$ns at 0.5 "$ns trace-annotate \"Inicio da transmissao CBR\""
$ns at 4.5 "$ns trace-annotate \"fim das trasnmissoes\""


#Tempo de inicio das aplica??es
$ns at 0.5 "$cbr start"
$ns at 0.4 "$ftp start"
$ns at 4.5 "$ftp stop"
$ns at 4.5 "$cbr stop"
$ns at 5.0 "finish"

$ns run
```

SIMULAÇÃO 3 – Parte 1

```
#Copyright (c) 1997 Regents of the University of California.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
```

```
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in
the
#    documentation and/or other materials provided with the
distribution.
# 3. All advertising materials mentioning features or use of this
software
#    must display the following acknowledgement:
#      This product includes software developed by the Computer
Systems
#      Engineering Group at Lawrence Berkeley Laboratory.
# 4. Neither the name of the University nor of the Laboratory may be
used
#    to endorse or promote products derived from this software without
#    specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''
AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
# SUCH DAMAGE.
#

Mac/Simple set bandwidth_ 1Mb

set MESSAGE_PORT 42
set BROADCAST_ADDR -1


#set val(chan)           Channel/WirelessChannel    ;#Channel Type
set val(prop)           Propagation/TwoRayGround   ;# radio-
propagation model
set val(netif)          Phy/WirelessPhy            ;# network
interface type



#set val(mac)            Mac/802_11                 ;# MAC type
#set val(mac)            Mac                 ;# MAC type
set val(mac)            Mac/Simple


set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue
type
set val(ll)             LL                         ;# link layer type
```

```
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         32768                         ;# max packet in
ifq


set val(rp) DumbAgent


set ns [new Simulator]

set f [open sht-temp.tr w]
$ns trace-all $f
$ns eventtrace-all
set nf [open sht-temp.nam w]
$ns namtrace-all-wireless $nf 700 200

# set up topography object
set topo       [new Topography]

$topo load_flatgrid 700 200

$ns color 3 green;
$ns color 8 red;
$ns color 1 black;
$ns color 7 purple;

#
# Create God
#
create-god 3

set mac0 [new Mac/Simple]

$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) \
            -channelType Channel/WirelessChannel \
                -topoInstance $topo \
                -agentTrace OFF \
                -routerTrace OFF \
                -macTrace ON \
                -movementTrace OFF

for {set i 0} {$i < 3} {incr i} {
      set node_($i) [$ns node]
      $node_($i) random-motion 0
}

$node_(0) color black
$node_(1) color black
$node_(2) color black


$node_(0) set X_ 200.0
$node_(0) set Y_ 30.0
$node_(0) set Z_ 0.0
```

```
$node_(1) set X_ 330.0
$node_(1) set Y_ 150.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 60.0
$node_(2) set Y_ 30.0
$node_(2) set Z_ 0.0

$ns at 0.25 "$node_(2) setdest 500.0 30.0 10000.0"

# subclass Agent/MessagePassing to make it do flooding

Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

Agent/MessagePassing/Flooding instproc recv {source sport size data} {
    $self instvar messages_seen node_
    global ns BROADCAST_ADDR

    # extract message ID from message
    set message_id [lindex [split $data ":"] 0]
    puts "\nNode [$node_ node-addr] got message $message_id\n"

    if {[lsearch $messages_seen $message_id] == -1} {
      lappend messages_seen $message_id
        $ns trace-annotate "[$node_ node-addr] received {$data} from
$source"
        $ns trace-annotate "[$node_ node-addr] sending message
$message_id"
      $self sendto $size $data $BROADCAST_ADDR $sport
    } else {
        $ns trace-annotate "[$node_ node-addr] received redundant
message $message_id from $source"
    }
}

Agent/MessagePassing/Flooding instproc send_message {size message_id
data port} {
    $self instvar messages_seen node_
    global ns MESSAGE_PORT BROADCAST_ADDR

    lappend messages_seen $message_id
    $ns trace-annotate "[$node_ node-addr] sending message
$message_id"
    $self sendto $size "$message_id:$data" $BROADCAST_ADDR $port
}



# attach a new Agent/MessagePassing/Flooding to each node on port
$MESSAGE_PORT
for {set i 0} {$i < 3} {incr i} {
    set a($i) [new Agent/MessagePassing/Flooding]
    $node_($i) attach  $a($i) $MESSAGE_PORT
    $a($i) set messages_seen {}
}

$ns at 0.1 "$a(0) send_message 500 1 {first_message} $MESSAGE_PORT"
$ns at 0.101 "$a(2) send_message 500 2 {second_message} $MESSAGE_PORT"
```

```
$ns at 0.2 "$a(0) send_message 500 11 {eleventh_message}
$MESSAGE_PORT"
$ns at 0.2 "$a(2) send_message 500 12 {twelfth_message} $MESSAGE_PORT"

$ns at 0.35 "$a(0) send_message 500 3 {third_message} $MESSAGE_PORT"
$ns at 0.351 "$a(2) send_message 500 4 {fourth_message} $MESSAGE_PORT"

$ns at 0.45 "$a(0) send_message 500 13 {thirteenth_message}
$MESSAGE_PORT"
$ns at 0.45 "$a(2) send_message 500 14 {fourteenth_message}
$MESSAGE_PORT"

for {set i 0} {$i < 3} {incr i} {
      $ns initial_node_pos $node_($i) 30
      $ns at 4.0 "$node_($i) reset";
}

$ns at 4.0 "finish"
$ns at 4.1 "puts \"NS EXITING...\"; $ns halt"



#INSERT ANNOTATIONS HERE

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

}

puts "Starting Simulation..."

$ns run
```

SIMULAÇÃO 3 – Parte 2

```
#Copyright (c) 1997 Regents of the University of California.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in
the
#    documentation and/or other materials provided with the
distribution.
# 3. All advertising materials mentioning features or use of this
software
#    must display the following acknowledgement:
```

```
#      This product includes software developed by the Computer
Systems
#      Engineering Group at Lawrence Berkeley Laboratory.
# 4. Neither the name of the University nor of the Laboratory may be
used
#    to endorse or promote products derived from this software without
#    specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''
AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
# SUCH DAMAGE.
#

Mac/Simple set bandwidth_ 1Mb

set MESSAGE_PORT 42
set BROADCAST_ADDR -1


#set val(chan)           Channel/WirelessChannel    ;#Channel Type
set val(prop)           Propagation/TwoRayGround   ;# radio-
propagation model
set val(netif)          Phy/WirelessPhy            ;# network
interface type



set val(mac)            Mac/802_11                 ;# MAC type
#set val(mac)            Mac                  ;# MAC type
#set val(mac)           Mac/Simple


set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue
type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         32768                         ;# max packet in
ifq


set val(rp) DumbAgent
```

```
set ns [new Simulator]

set f [open rts-cts-data-ack.tr w]
$ns trace-all $f
$ns eventtrace-all
set nf [open rts-cts-data-ack-temp.nam w]
$ns namtrace-all-wireless $nf 700 200

# set up topography object
set topo       [new Topography]

$topo load_flatgrid 700 200

$ns color 3 green;
$ns color 8 red;
$ns color 1 black;
$ns color 7 purple;
$ns color 6 tan;
$ns color 2 orange;

#
# Create God
#
create-god 3


#set mac0 [new Mac/802_11]

$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) \
            -channelType Channel/WirelessChannel \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace OFF \
                -macTrace ON \
                -movementTrace OFF

for {set i 0} {$i < 3} {incr i} {
      set node_($i) [$ns node]
      $node_($i) random-motion 0
}

$node_(0) color black
$node_(1) color black
$node_(2) color black


$node_(0) set X_ 200.0
$node_(0) set Y_ 30.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 330.0
$node_(1) set Y_ 150.0
$node_(1) set Z_ 0.0
```

```
$node_(2) set X_ 60.0
$node_(2) set Y_ 30.0
$node_(2) set Z_ 0.0

$ns at 0.6 "$node_(2) setdest 330.0 30.0 10000.0"
$ns at 1.1 "$node_(2) setdest 500.0 30.0 10000.0"

# subclass Agent/MessagePassing to make it do flooding

Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

Agent/MessagePassing/Flooding instproc recv {source sport size data} {
    $self instvar messages_seen node_
    global ns 1

    # extract message ID from message
    set message_id [lindex [split $data ":"] 0]
    puts "\nNode [$node_ node-addr] got message $message_id\n"

    if {[lsearch $messages_seen $message_id] == -1} {
      lappend messages_seen $message_id
        $ns trace-annotate "[$node_ node-addr] received {$data} from
$source"
        $ns trace-annotate "[$node_ node-addr] sending message
$message_id"
      $self sendto $size $data 1 $sport
    } else {
        $ns trace-annotate "[$node_ node-addr] received redundant
message $message_id from $source"
    }
}

Agent/MessagePassing/Flooding instproc send_message {size message_id
data port} {
    $self instvar messages_seen node_
    global ns MESSAGE_PORT 1

    lappend messages_seen $message_id
    $ns trace-annotate "[$node_ node-addr] sending message
$message_id"
    $self sendto $size "$message_id:$data" 1 $port
}



# attach a new Agent/MessagePassing/Flooding to each node on port
$MESSAGE_PORT
for {set i 0} {$i < 3} {incr i} {
    set a($i) [new Agent/MessagePassing/Flooding]
    $node_($i) attach  $a($i) $MESSAGE_PORT
    $a($i) set messages_seen {}
}

$ns at 0.1 "$a(0) send_message 500 1  {first_message} $MESSAGE_PORT"
$ns at 0.1 "$a(2) send_message 500 2  {second_message} $MESSAGE_PORT"

$ns at 0.8 "$a(0) send_message 500 5  {fifth_message} $MESSAGE_PORT"
$ns at 0.8 "$a(2) send_message 500 6  {sixth_message} $MESSAGE_PORT"

$ns at 1.3 "$a(2) send_message 500 15 {fifteenth_message}
$MESSAGE_PORT"
```

```
$ns at 1.3 "$a(0) send_message 500 16 {sixteenth_message}
$MESSAGE_PORT"

for {set i 0} {$i < 3} {incr i} {
      $ns initial_node_pos $node_($i) 30
      $ns at 20.0 "$node_($i) reset";
}

$ns at 20.0 "finish"
$ns at 20.1 "puts \"NS EXITING...\"; $ns halt"


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

}

puts "Starting Simulation..."

$ns run
```