Arquitectura de Redes 2004/05

Introdução ao simulador de redes *ns2*

Tópicos

- Arquitectura do simulador de redes ns2
- Programação tcl/otcl
- Componentes de rede ns2
- Scripts de simulação ns2

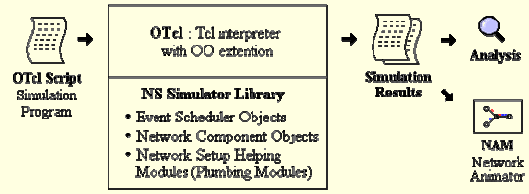
NS2?

- Simulador de eventos discreto para redes de comunicação
- Simulação do comportamento de diferentes tipos de rede/Protocolos
  - Fixas/Móveis
  - Wireless
  - Ad-hoc
  - Sensores
  - Satélite
  - Multicast
  - IP móvel

NS2?

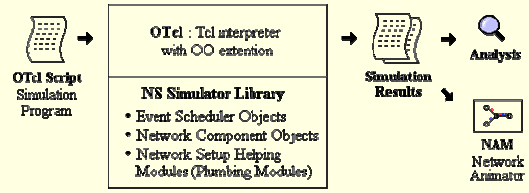
- Criação de topologias de rede
  - fixas/móveis
- Componentes de rede suportados:
  - Meios de transmissão (incluindo meios sem fio)
  - Nós
  - Protocolos MAC
  - Algoritmos de routing
  - Filas de espera: Drop Tail, RED, CBQ…
  - Protocolos de transporte: TCP, UDP;
  - Fontes de Tráfego: FTP, Telnet, 'Web', CBR, VBR…
- Desenvolvimento de novos componentes de rede!!
- Extração de informação relevante sobre o comportamento de das redes/protocolos

NS2 – Estrutura 1



- Interpretador de scripts OTcl
  - Escalonador de eventos
  - Componentes de rede
  - Componentes de ligação
- Geração de 'traces' da simulação
  - Análise / processamento estatístico
  - Obter resultados da simulação…
- Geração de 'ficheiros de animação' para vizualização externa (e.g. nam)

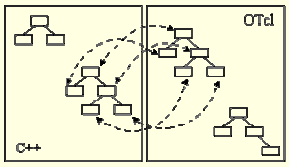
NS2 – Estrutura 2

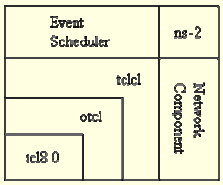


- Script OTcl:
  - Inicialização do escalonador de eventos
  - Abertura de ficheiros para *output* da simulação
  - Definição da topologia da rede
    - Instanciação de componentes de rede
      - Nós, agentes, fontes de tráfego
    - Definição de 'ligações'/associações entre componentes de rede
  - Escalonamento explicito de eventos
    - início/fim de transmissão em fontes de tráfego
    - Escrita de informação sobre simulação em ficheiros
    - Fim de simulação (e acções consequentes…)

Dualidade OTcl – C++

- Ns2 está escrito em C++ e OTcl
  - Aumento de eficiência → C++
  - Flexibilidade para a 'descrição' da simulação → OTcl
- Plano de dados (C++)
  - Escalonador de eventos
  - Componentes de rede
- Plano de controlo (OTcl)
  - Objecto OTcl ←→ objecto C++ (OTcl linkage)
  - Acesso a variáveis e funções membro de objectos C++ partir dos objectos OTcl





NS2 – Modo Interactivo

```
[slc@host]$ns
% set ns [new Simulator]
_o4
% $ns at 1 "puts \"Hello World!\""
1
% $ns at 1.5 "exit"
2
% $ns run
Hello World!
[slc@host]$
```

NS2 – Modo Passivo

example.tcl

```
set ns [new Simulator]
$ns at 1 “puts \“Hello World!\””
$ns at 1.5 “exit”
$ns run
```

[slc@host]$ns example.tcl

Hello World!

[slc@host]$

TCL

example1.tcl

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}

# Calling the "test" procedure created above
test
```

[slc@host]$ns example1.tcl

```
k < 5, pow = 1.0
k < 5, pow = 1120.0
k < 5, pow = 1254400.0
k < 5, pow = 1404928000.0
k < 5, pow = 1573519360000.0
k >= 5, mod = 0
k >= 5, mod = 4
k >= 5, mod = 0
k >= 5, mod = 0
k >= 5, mod = 4
```

OTCL

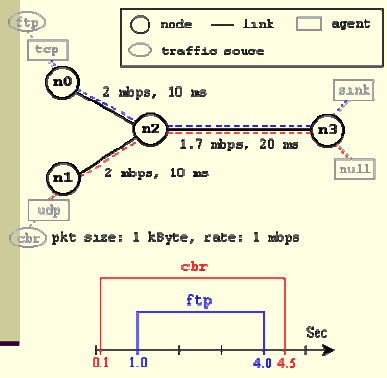
example2.tcl

```
# add a member function call 'greet'
Class mom
mom instproc greet {} {
    $self instvar age_
    puts "$age_ year old mom say:
    How are you doing?"
}

# Create a child class of "mom" called "kid"
# and overide the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
    $self instvar age_
    puts "$age_ year old kid say:
    What's up, dude?"
}

# Create a mom and a kid object, set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function 'greet' of each object
$a greet
$b greet
```

[slc@host]$ns example2.tcl

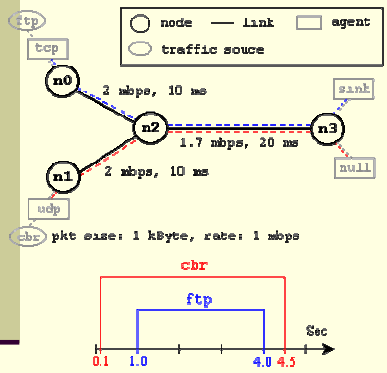
```
45 year old mom say:
    How are you doing?
15 year old kid say:
    What's up, dude?
```

NS2 – exemplo de simulação 1



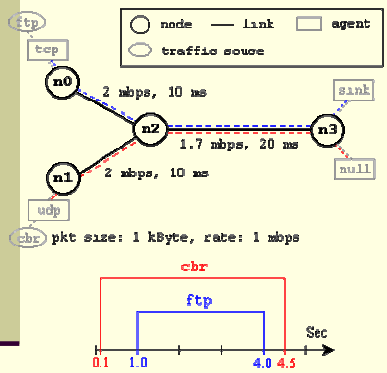

```
#Create a simulator object
set ns [new Simulator]

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

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

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace
file
        exec nam out.nam &
        exit 0
}
```

NS2 – exemplo de simulação 2



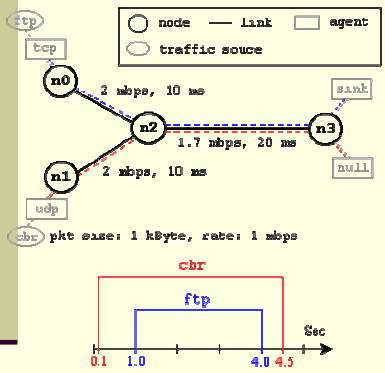

```
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

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

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

#Monitor the queue for link (n2-n3). (for
NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

NS2 – exemplo de simulação 3




```
#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

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

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
```

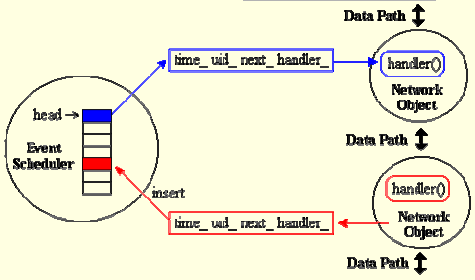
NS2 – exemplo de simulação 4




```
#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really
necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns
detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds
of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set
packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

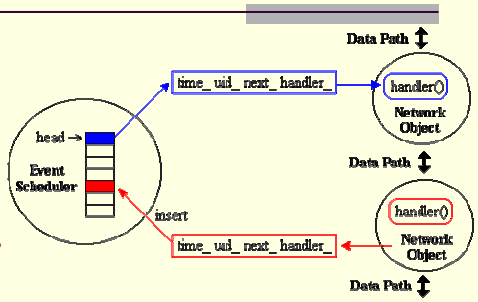
Escalonador de eventos 1

- Objectos recorrem ao escalonador de eventos
  - Atraso no tratamento de pacotes
  - *Timers*
- Evento será tratado pelo objecto que o escalonou
- Escalonamento explícito de eventos
  - $ns at 0.1 "$cbr start”
  - $ns at 5.0 "finish“



Escalonador de eventos 2

- Criação do Simulador de eventos (→ escalonador)
  - set ns [new Simulator]
- Escalonamento de um evento
  - $ns at <time> <event>
  - <event>: qualquer instrução ns/tcl
- Arranque do Simulador/Escalonador
  - $ns run

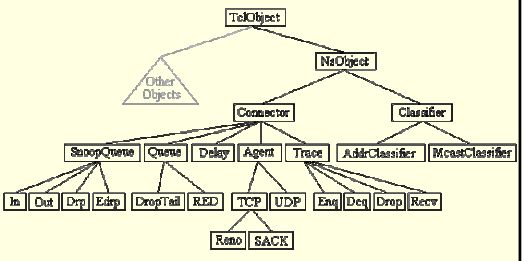


Funções membro do Simulator que fazem interface com o escalonador de eventos:

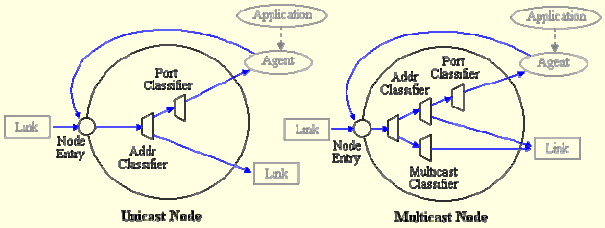
Simulator instproc now
at args
at-now arg
after n args
run args
halt

Componentes de rede
Hierarquia de componentes (OTcl)

- Raiz da Hierarquia – TclObject
  - Other objects → escalonador, timers, nam…, etc
  - NsObject → superclasse de todos componentes de rede (que tratam de pacotes)
- Connector → 1 só output para
- Classifier → multiplexagem entre para diferentes outputs



Componentes de rede
Nós / *routing*



- Unicast (por defeito)
  - $ns rtproto type
  - type: Static, Session, DV, cost, multi-path
- Multipath
  - $ns multicast
  - $ns mrtproto type
  - *type*: CtrMcast, DM, ST, BST

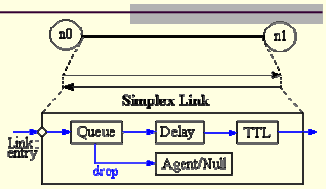
- Criação de nós
  - set n0 [$ns node]
  - set n1 [$ns node]

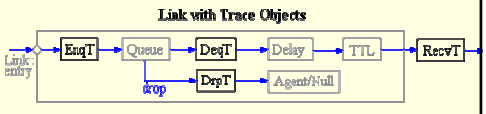
Componentes de rede
Links / Tracing

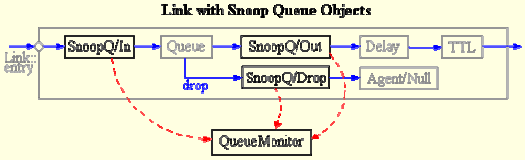
- *duplex-link = 2 simplex-links*
- Fila de saída de um nó na verdade é implementada como componente de um *simplex-link*;

- $ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>
- <queue_type>: DropTail, RED, CBQ, FQ, SFQ, DRR

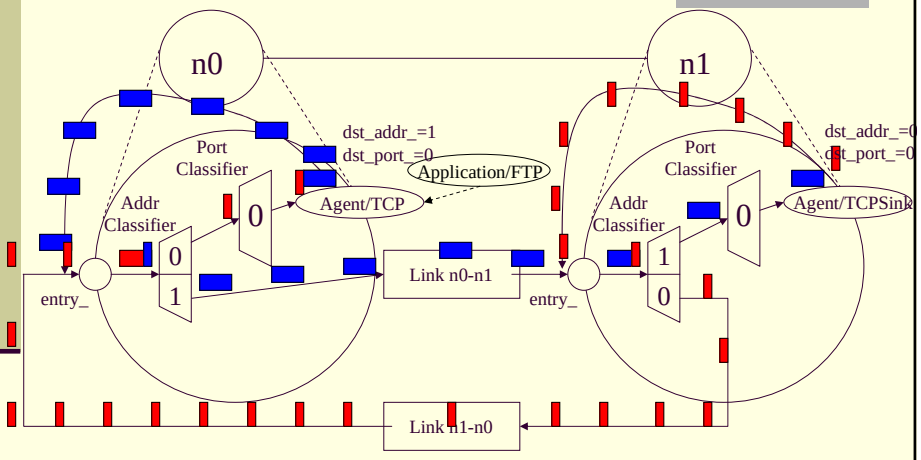
- Tracing → registo (para ficheiro) de todos eventos relacionados com *links*
  - $ns trace-all file
  - $ns namtrace-all file



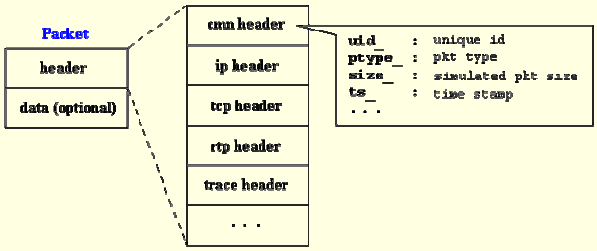




Fluxo de Pacotes



Componentes de rede
Formato de Pacote



Análise de Traces

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

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Close the Trace file
        close $tf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
. . .
```

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop (at queue)

src_addr : node.port (3.0)
dst_addr : node.port (0.0)

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```