

Corso di Laurea in Ingegneria Informatica



Corso di Reti di Calcolatori

Roberto Canonico (roberto.canonico@unina.it)

Giorgio Ventre (giorgio.ventre@unina.it)

Il simulatore di reti ns2

I lucidi presentati al corso sono uno strumento didattico che NON sostituisce i testi indicati nel programma del corso

Nota di copyright per le slide COMICS



Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,
Marcello Esposito, Roberto Canonico, Giorgio Ventre

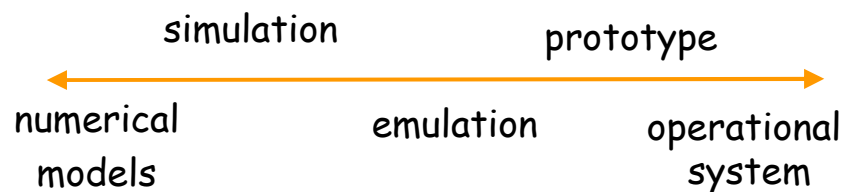
Simulazione di reti di calcolatori

Per valutare protocolli di comunicazione ed algoritmi distribuiti in reti di calcolatori sono possibili varie alternative:

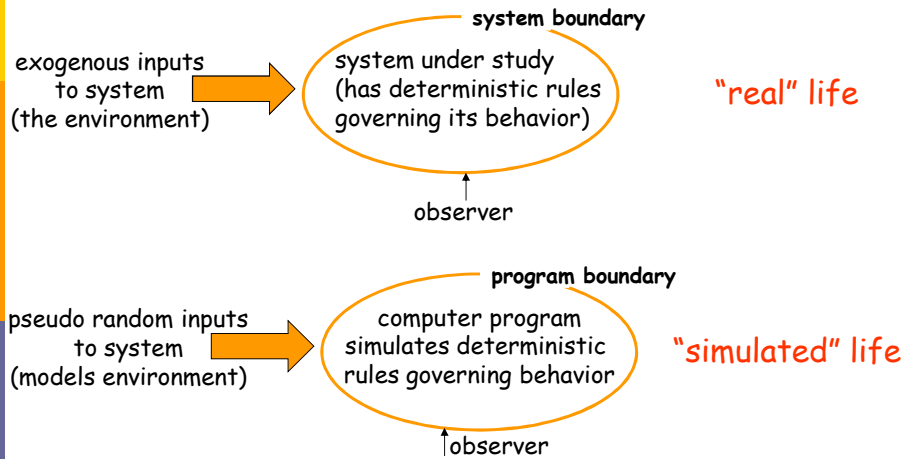
- 1) Setup sperimentali su piccola scala (in laboratorio)
- 2) Testbed sperimentali su media scala (*wide area*), PlanetLab
- 3) Sistemi di network emulation
- 4) Ambienti di simulazione generali per reti di calcolatori
 - ns-2, GLOMOSIM, OPNET, NCTUns, ...
- 5) Strumenti di simulazione sviluppati *ad hoc*
- 6) Modelli matematici del sistema



The evaluation spectrum



What is simulation?



Why Simulation?

- **goal:** study system *performance, operation*
- real-system not *available, is complex/costly or dangerous* (eg: space simulations, flight simulations)
- quickly evaluate design *alternatives* (eg: different system configurations)
- evaluate *complex functions* for which closed form formulas or numerical techniques not available



Requisiti per un simulatore di reti

- ✓ Astrazione
- ✓ Generazione di scenari e topologie
- ✓ Programmabilità
- ✓ Estendibilità
- ✓ Disponibilità di un'ampia gamma di moduli di protocolli riutilizzabili, affidabili e validati
- ✓ Possibilità di modificare protocolli esistenti
- ✓ Visualizzazione dei risultati
- ✓ Emulazione



Programming a simulation

What 's in a simulation program?

- **simulated time**: internal (to simulation program) variable that keeps track of simulated time
- **system "state"**: variables maintained by simulation program define system "state"
 - e.g., may track number (possibly order) of packets in queue, current value of retransmission timer
- **events**: points in time when system changes state
 - each event has associated **event time**
 - e.g., arrival of packet to queue, departure from queue
 - precisely at these points in time that simulation must take action (change state and may cause new future events)
 - model for time between events (probabilistic) caused by external environment

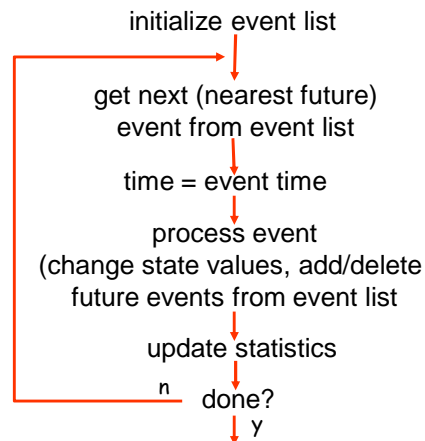


Discrete Event Simulation

- simulation program maintains and updates list of future events: **event list**
- simulator structure:

Need:

- well defined set of events
- for each event: simulated system action, updating of event list



Things to remember about Discrete Event Simulation

- The programming model revolves around “**events**” (eg: packet arrivals):
 - Events trigger particular sub-routines
 - Huge “switch” statement to classify events and call appropriate subroutine
 - The subroutine may schedule new events! (cannot schedule events for past, i.e., events are causal)
 - Rarely you might introduce new event *types*
- Events have associated with them:
 - Event type, event data structures (eg: packet)
 - Simulation time when the event is scheduled
- **Key event operations:** *Enqueue* (i.e. schedule a event)
 - Dequeue is handled by the simulation engine



Discrete Event Simulation: Scheduler

11

- Purpose: maintain a notion of simulation time, schedule events. A.k.a: “simulation engine”
- **Simulation time \neq Real time**
 - A simulation for 5 sec of video transmission *might* take 1 hour!
- Events are sorted by simulation time (not by type!):
priority queue or heap data structure
 - After all subroutines for an event have been executed, control is transferred to the simulation engine
 - The simulation engine schedules the next event available at the same time (if any)
 - Once all the events for current time have been executed, simulation time is advanced and nearest future event is executed.
 - Simulation time = time of currently executing event

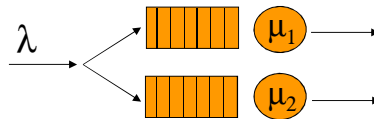


roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Simulation: example

12

- packets arrive (avg. interarrival time: $1/\lambda$) to router (avg. execution time $1/\mu$) with two outgoing links
- arriving packet joins link i with probability ϕ_i



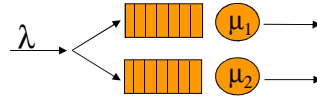
- state of system: size of each queue
- system events:
 - packet arrivals
 - service time completions



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Simulation: example

Simulator actions on *arrival* event

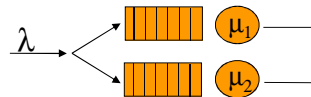


- choose a link
 - if link idle {place pkt in service, determine service time (random number drawn from service time distribution) add future event onto event list for pkt transfer completion, set number of pkts in queue to 1}
 - if buffer full {increment # dropped packets, ignore arrival}
 - else increment number in queue where queued
- create event for next arrival (generate interarrival time) stick event on event list



Simulation: example

Simulator actions on *departure* event



- remove event, update simulation time, update performance statistics
- decrement counter of number of pkts in queue
- If (number of jobs in queue > 0) put next pkt into service – schedule completion event (generate service time for put)



Un simulatore generale di reti: ns-2

- ✓ Simulatore di reti sviluppato ad ISI/USC (University of Southern California)
- ✓ Codice sorgente di pubblico dominio
- ✓ <http://www.isi.edu/nsnam/>
- ✓ Tipologia: **simulatore ad eventi discreti**
- ✓ Il pacchetto è l'entità logica di base della simulazione
 - NON il più adatto a simulare il livello fisico



"ns" components

- ns, the simulator itself
- nam, the Network AniMator
 - visualize ns (or other) output
 - GUI input simple ns scenarios
- pre-processing:
 - traffic and topology generators
- post-processing:
 - simple trace analysis, often in Awk, Perl, or Tcl
- tutorial: <http://www.isi.edu/nsnam/ns/tutorial/index.html>
- ns by example: <http://nile.wpi.edu/NS/>



Un simulatore generale di reti: ns-2

Adatto per simulazione di protocolli a livello:

- ✓ Data-link
 - *Schemi MAC in reti wired/wireless*
- ✓ Rete
 - *Routing, Scheduling, Multicast*
- ✓ Trasporto
 - *Congestion control*
- ✓ Applicazione
 - *Caching, Streaming*



ns-2: piattaforme supportate

Sistemi UNIX o UNIX-like:

- Linux
- Sistemi con supporto POSIX

Microsoft Windows

- Richiede Cygwin
- Versioni vecchie stand-alone

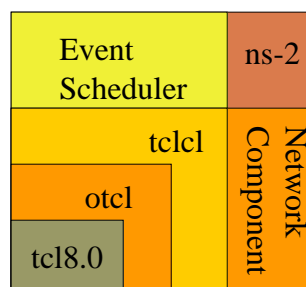


ns-2: architettura software

- ✓ Basato su un'architettura sw modulare
- ✓ *Object-oriented* sia nella struttura interna che nella interfaccia di programmazione
- ✓ Motore di simulazione ad eventi in C++
- ✓ Linguaggio di simulazione interpretato: Tcl ad oggetti – OTcl
- ✓ OTcl per il controllo della simulazione
- ✓ C++ per gestire gli eventi legati al trattamento dei pacchetti



ns-2: architettura software (2)



Separazione C++/OTcl

- Il motore della simulazione è scritto in C++ e gestisce la coda degli eventi e le elaborazioni da eseguire per pacchetto
- Maggior parte dei protocolli implementata in OTcl
- L'utente di solito scrive codice OTcl
- Solo se si vogliono implementare politiche di trattamento dei pacchetti particolari è necessario creare o modificare classi C++



C++ and OTcl Separation

- “data” / control separation
 - C++ for “data”:
 - per packet processing, core of *ns*
 - fast to run, detailed, complete control
 - OTcl for control:
 - Simulation scenario configurations
 - Periodic or triggered action
 - Manipulating existing C++ objects
 - fast to write and change
- + running vs. writing speed
- Learning and debugging (two languages)



I linguaggi Tcl ed OTcl

I linguaggi Tcl ed OTcl

24

- Tcl (**T**ool **C**ommand **L**anguage) è un linguaggio di scripting di tipo general-purpose
- Tcl è spesso utilizzato congiuntamente a Tk
 - Tk è un insieme di librerie e comandi che consentono di creare con facilità interfacce grafiche
- OTcl è un insieme di estensioni del linguaggio Tcl che consentono di implementare un ambiente di programmazione di tipo OO



Tcl: variabili

Variabili:

- In Tcl le variabili non devono essere dichiarate prima di essere utilizzate
- Le variabili Tcl vengono create automaticamente all'atto della prima assegnazione:
 - `set month 2`
 - `set day 3`
 - `set year 97`
- Per referenziare le variabili Tcl è necessario utilizzare il simbolo \$:
 - `set date "$month:$day:$year"`

Espressioni:

- In Tcl è possibile valutare espressioni di tipo differente (matematiche, logiche, ...) utilizzando la parola chiave `expr`:
 - `expr 0 == 1`
 - `expr 4 + 5`



Tcl: Controllo di Flusso

Tcl supporta i classici costrutti per il controllo di flusso:

□ If-else:

```
if {$temp < 21} {
  puts "It's a little chilly."
} else {
  puts "Warm enough for me."
}
```

□ If-elseif:

```
if {$my_planet == "earth"} {
  puts "I feel right at home."
} elseif {$my_planet == "venus"} {
  puts "This is not my home."
} else {
  puts "I am neither from Earth, nor from Venus."
}
```



Tcl: Controllo di Flusso (2)

□ switch:

```
switch $num_legs {  
  2 {puts "It could be a human."}  
  4 {puts "It could be a cow."}  
  6 {puts "It could be an ant."}  
  8 {puts "It could be a spider."}  
  default {puts "It could be anything."}  
}
```



Tcl: cicli

□ for:

```
for {set i 0} {$i < 10} {incr i 1} {  
  puts "In the for loop, and i == $i"  
}
```

□ while:

```
while {$i < 10} {  
  puts "In the while loop, and i == $i"  
  incr i 1  
}
```



Tcl: procedure

- In Tcl la semantica delle procedure è simile a quella del C

```
proc magnitude {num} {
  if {$num > 0} {
    return $num
  }
  set num [expr $num * (-1)]
  return $num
}
```

- Per eseguire una funzione è sufficiente scriverne il nome seguito dalla lista dei parametri:

```
set abs [magnitude -3]
```



OTcl: concetti di programmazione OO

- **Definizione di una Classe:**

```
class MyNode -superclass Node
```

- **Creazione di un Oggetto:**

```
set mynodevar [new MyNode]
```

- **Definizione di un metodo:**

```
MyNode instproc memberfn1 {} {
  puts "in member function 1"
}
```

- **Attributi (o dati-membro):**

- definiti all'interno delle funzioni membro con la direttiva **instvar**:

```
MyNode instproc memberfn2 {} {
  $self instvar datamember1
  set datamember1 "data member - memberfn2"
}
```



OTcl: concetti di programmazione OO (2)

□ Invocazione di un metodo:

```
$mynodevar memberfn1
```

□ Costruttore:

- Il costruttore in OTcl ha il nome **init**
- Per invocare il costruttore della superclass è necessario utilizzare la parola chiave **next**

```
MyNode instproc init {} {
    $self next
    $self instvar datamember1
    set datamember1 0
}
```



Esempi di programmazione OTcl – (1)

```
Class movie
movie instproc showInfo {} {
    $self instvar title_
    puts "Titolo del film: $title_"
}
```

- Definisce una classe OTcl *movie*
- Crea un metodo *showInfo*
- Crea un attributo *title_*



Esempi di programmazione OTcl – (2)

```
set a [new movie]
$a set title_ "Great Expectations"
$a showInfo
```

- Crea un oggetto *a* di classe *movie*
- Assegna un valore alla variabile *title_*
- Invoca il metodo *showInfo* su *a*



Esempi di programmazione OTcl – (3)

```
Class thriller -superclass movie
thriller instproc showInfo {} {
    $self next_
    puts "Genere del film: thriller"
}
set b [new thriller]
$b set title_ "Psycho"
$b showInfo
```

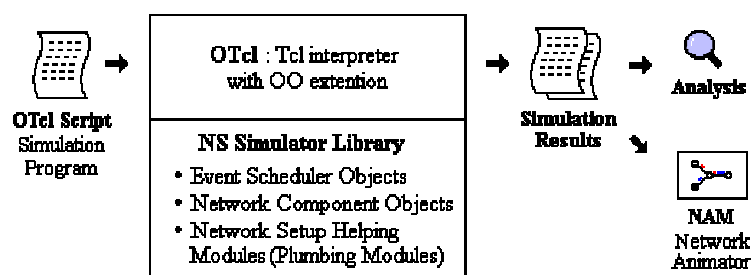
- Deriva la classe *thriller* da *movie*
- Ridefinisce *showInfo*



Programmazione in ns-2

ns: come si usa

36



Output tipico:

- out.tr → per post-elaborazione
- out.nam → per visualizzazione con nam



Scenario di simulazione

In generale, uno scenario di simulazione consiste di tre categorie di componenti:

- ✓ **Topologia** (*Routed Topology*)
- ✓ **Elementi attivi nei terminali**
 - Sorgenti di Traffico (*Application*)
 - Entità di protocollo di trasporto (*Agent*)
 - Connessioni tra entità di protocollo
- ✓ **Eventi esterni e guasti**



Topologia

- ✓ Definisce il numero di nodi e le loro connessioni
- ✓ Per ciascun link è specificata la banda, il ritardo di propagazione, la disciplina di gestione della coda di trasmissione dei pacchetti, ecc.
- ✓ Sono definiti i protocolli che determinano i percorsi di instradamento dei pacchetti (*routing*)
- ✓ E' possibile usare *generatori di topologie* (es. GT-ITM) per ottenere scenari realistici



Topology Generation

- <http://www.isi.edu/nsnam/ns/ns-topogen.html>

Packages	Graphs	Edge Method
NTG	n-level	probabilistic
RTG	Flat random	Waxman
GT-ITM	Flat random, n-level, Transit-stub	various
TIERS	3-level	spanning tree

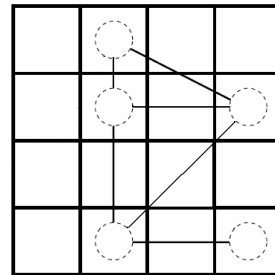


roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Metodi per la generazione di topologie di rete

Modelli flat

- Random $P(u,v) = \alpha$
- Waxman 1 $P(u,v) = \alpha \cdot e^{\frac{-d}{(\beta \cdot L)}}$
- Waxman2 $P(u,v) = \alpha \cdot e^{\frac{-rand(0,L)}{(\beta \cdot L)}}$
- Doar-Leslie $P(u,v) = \alpha \frac{k \cdot \epsilon}{m} \cdot e^{\frac{-d}{(\beta \cdot L)}}$
- Esponenziale $P(u,v) = \alpha \cdot e^{\frac{-d}{(L-d)}}$
- Locale $P(u,v) = \begin{cases} \alpha & \text{se } d < L \cdot \text{raggio} \\ \beta & \text{se } d \geq L \cdot \text{raggio} \end{cases}$

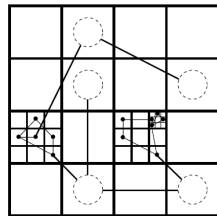
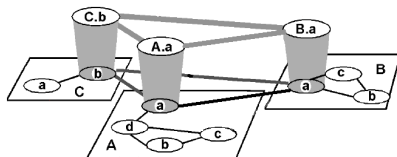


roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

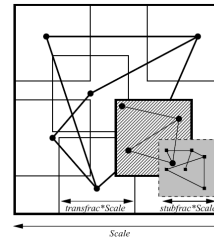
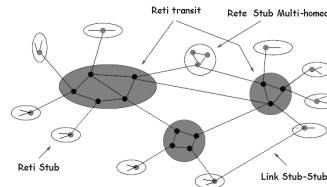
Modelli gerarchici

41

Modello ad n-livelli



Modello Transit-stub



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Elementi attivi nei terminali

42

✓ Agent

- sono entità di protocolli di comunicazione che vengono istanziate negli *end-system* (terminali)
 - TCP, UDP (livello trasporto)
 - Ftp, Telnet (livello applicativo)

✓ Traffic source

- *Constant bit rate* (CBR)
- Stocastiche di assegnata distribuzione (es. Poisson)
- Deterministiche definite da un *trace file*

✓ Connection

- Associazioni tra Agent
 - TCP sender e TCP receiver
 - gruppi di ricevitori multicast



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Eventi esterni e guasti

- ✓ **Eventi esterni:**
 - start / end di una connessione TCP
 - join / leave di un ricevitore ad un gruppo mcast
 - movimento di un nodo mobile (reti wireless)
- ✓ **Guasti (deterministici o probabilistici):**
 - Guasto di un link
 - Pattern di perdita di pacchetti
 - Pattern di congestione



Network dynamics

- **Link failures**
 - route changes reflected automatically
 - can emulate node failure
- **Commands:**
 - \$ns rtmodel-at <time> <up|down> \$n0 \$n1
 - \$ns rtmodel Trace <config_file> \$n0 \$n1
 - \$ns rtmodel <model> <params> \$n0 \$n1
 - <model>: Deterministic, Exponential
 - <params>: [<start>] <up_interval> <down_interval> [<finish>]



Ns programming: logical steps

- Create the event scheduler
- Turn on tracing
- Create network
- Setup routing
- Insert errors
- Create transport connection
- Create traffic
- Transmit application-level data



ns: programmazione degli eventi

```
set ns [new Simulator]
$ns at 1.0 {puts "Hello world!"}
$ns at 1.5 exit
$ns run
```

- Inizializza uno schedulatore di eventi
- Inserisce 2 eventi esterni nello schedulatore
- Fa partire la simulazione



La prima simulazione in ns - (0)

```
set ns [new Simulator]

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



La prima simulazione in ns - (1)



```
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
```



La prima simulazione in ns - (2)



```

set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n1 $sink
$ns connect $tcp $sink
  
```



La prima simulazione in ns - (3)



```

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



La prima simulazione in ns - (4)

```

$ns at 0.1 "$src start"
$ns at 2.0 finish
proc finish {} {
    global ns f nf
    $ns flush-trace
    close $nf
    close $f
    exit 0
}
$ns run

```



Tracing and Monitoring I

□ Packet tracing:

- On all links: \$ns trace-all [open out.tr w]
- On one specific link: \$ns trace-queue \$n0 \$n1\$str

```

<Event> <time> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <attr>
+ 1 0 2 cbr 210 ----- 0 0.0 3.1 0 0
- 1 0 2 cbr 210 ----- 0 0.0 3.1 0 0
r 1.00234 0 2 cbr 210 ----- 0 0.0 3.1 0 0

```

- We have new trace format

□ Event tracing (support TCP right now)

- Record "event" in trace file: \$ns eventtrace-all

```

E 2.267203 0 4 TCP slow_start 0 210 1

```



Tracing and Monitoring II

□ Queue monitor

```
set qmon [$ns monitor-queue $n0 $n1 $q_f $sample_interval]
```

- Get statistics for a queue

```
$qmon set pdrops_
```

- Record to trace file as an optional

```
29.0000000000000142 0 1 0.0 0.0 4 4 0 1160 1160 0
```

□ Flow monitor

```
set fmon [$ns_ makeflowmon Fid]
```

```
$ns_ attach-fmon $slink $fmon
```

```
$fmon set pdrops_
```



Tracing and Monitoring III

□ Visualize trace in nam

```
$ns namtrace-all [open test.nam w]
```

```
$ns namtrace-queue $n0 $n1
```

□ Variable tracing in nam

```
Agent/TCP set nam_tracevar_ true
```

```
$tcp tracevar srtt_
```

```
$tcp tracevar cwnd_
```

□ Monitor agent variables in nam

```
$ns add-agent-trace $tcp $tcp
```

```
$ns monitor-agent-trace $tcp
```

```
$srm0 tracevar cwnd_
```

```
.....
```

```
$ns delete-agent-trace $tcp
```



ns: formato del trace file out.tr

```

+ 0.1      0 1 tcp 1000 ----- 0 0.0 1.0 0 0
- 0.1      0 1 tcp 1000 ----- 0 0.0 1.0 0 0
r 0.1108   0 1 tcp 1000 ----- 0 0.0 1.0 0 0
+ 0.1108   1 0 ack 40  ----- 0 1.0 0.0 0 1
- 0.1108   1 0 ack 40  ----- 0 1.0 0.0 0 1
r 0.120832 1 0 ack 40  ----- 0 1.0 0.0 0 1

```

timestamp

trx

rcv

dim

seq

dst

src

flowid

+ pacchetto inserito nella coda del link
 - pacchetto estratto dalla coda del link
 r pacchetto ricevuto



roberto.canonico@unina.it
 Università degli Studi di Napoli FEDERICO II

ns: formato del trace file out.tr (2)

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)      src_addr : node.port (3.0)
- : dequeue (at queue)     dst_addr : node.port (0.0)
d : drop (at queue)

```

```

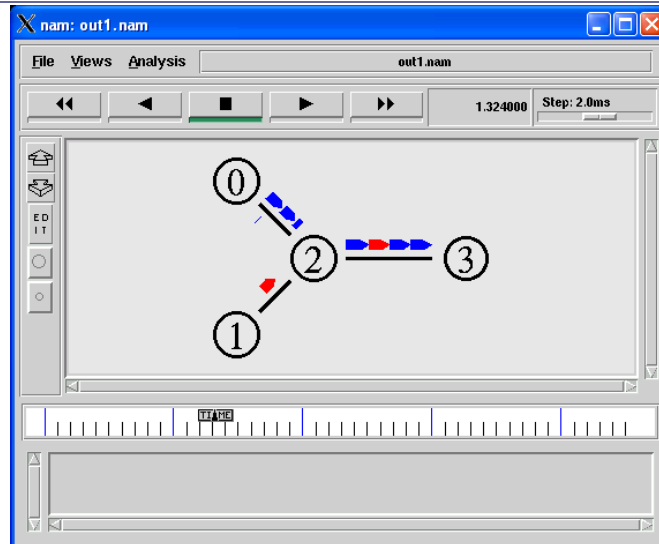
r 1.3556 3 2 ack 40 ----- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ----- 2 1.0 3.1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

```



roberto.canonico@unina.it
 Università degli Studi di Napoli FEDERICO II

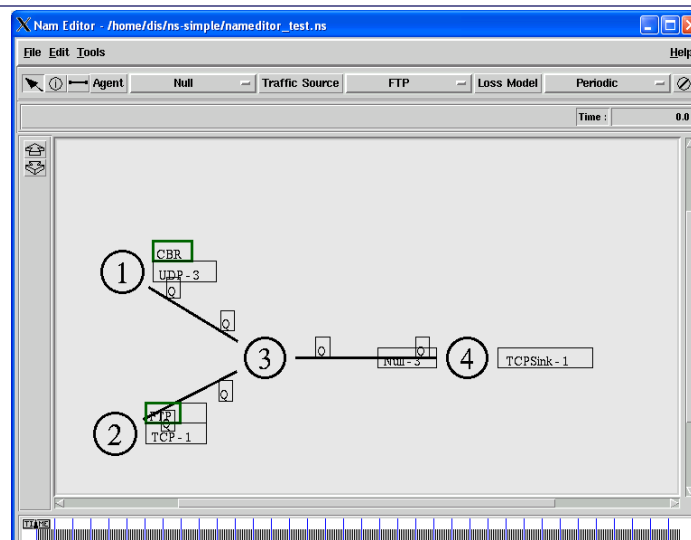
nam: Network Animation tool



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II



Creazione di scenari con Nam Editor



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II



NAM Input: out.nam

- Can define events for nodes, links, queues, packets, agents

- Define a node event

```
n -t * -s 69 -v circle -c grey -z 0.177110
```

- Define a link event

```
l -t * -s 66 -d 67 -r 512000.000000 -D 0.010000 -c
grey -o 236.6deg -l 0.012432
```



nam Interface: Color

- Color mapping

```
$ns color 40 red
```

```
$ns color 41 blue
```

```
$ns color 42 chocolate
```

- Color ↔ flow id association

```
$tcp0 set fid_ 40 ;# red packets
```

```
$tcp1 set fid_ 41 ;# blue packets
```



nam Interface: Nodes

- Color


```
$node color red
```
- Shape (can't be changed after sim starts)


```
$node shape box ;# circle, box, hexagon
```
- Marks (concentric “shapes”)


```
$ns at 1.0 "$n0 add-mark m0 blue box"
      $ns at 2.0 "$n0 delete-mark m0"
```
- Label (single string)


```
$ns at 1.1 "$n0 label \"web cache 0\""
```



nam Interfaces: Links

- Color


```
$ns duplex-link-op $n0 $n1 color "green"
```
- Label


```
$ns duplex-link-op $n0 $n1 label "abcd"
```
- Dynamics (automatically handled)


```
$ns rtmodel Deterministic {2.0 0.9 0.1} $n0 $n1
```
- Asymmetric links not allowed



nam Interface: Topology Layout

- “Manual” layout: specify everything

```
$ns duplex-link-op $n(0) $n(1) orient right
$ns duplex-link-op $n(1) $n(2) orient right
$ns duplex-link-op $n(2) $n(3) orient right
$ns duplex-link-op $n(3) $n(4) orient 60deg
```

- If anything missing → automatic layout



nam Interface: Misc

- Annotation

- Add textual explanation to your simulation

```
$ns at 3.5 "$ns trace-annotate \"packet drop\""
```

- Set animation rate

```
$ns at 0.0 "$ns set-animation-rate 0.1ms"
```



NS-2 internals

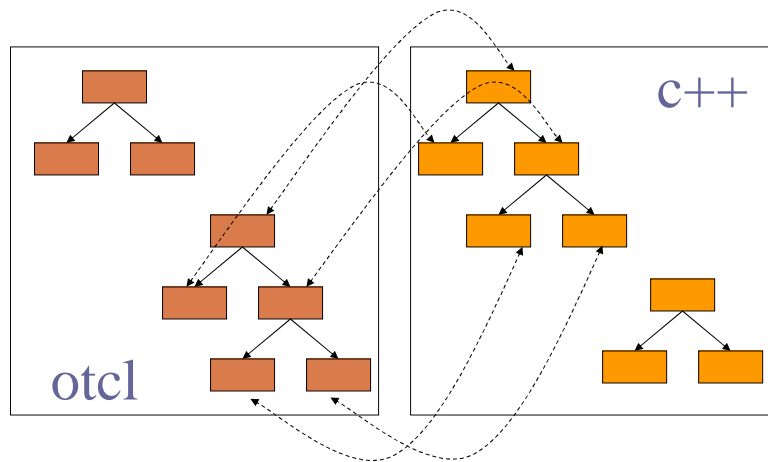
Gestione del tempo nel simulatore

66

- Il motore di simulazione è ad eventi discreti
- Il tempo non avanza autonomamente
- Gli eventi sono caratterizzati dall'istante di simulazione ed inseriti in una *lista degli eventi*
- Il simulatore agisce sulla lista eventi
 - inserisce un nuovo evento nella lista
 - processa l'evento attuale estraendolo dalla lista
 - esegue le azioni associate all'evento
 - elimina l'evento processato
- L'accesso alla lista è gestito da uno *scheduler*

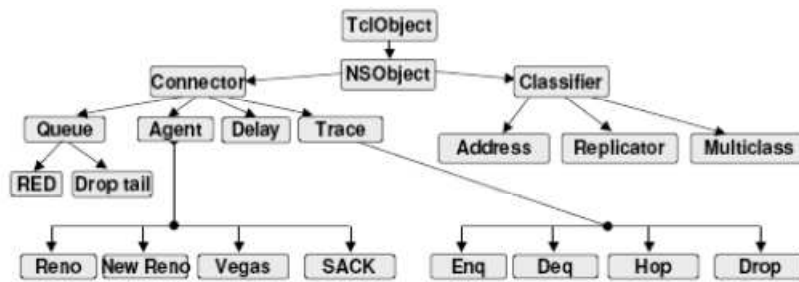


Corrispondenza tra oggetti OTcl e C++



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

C++: gerarchia delle classi



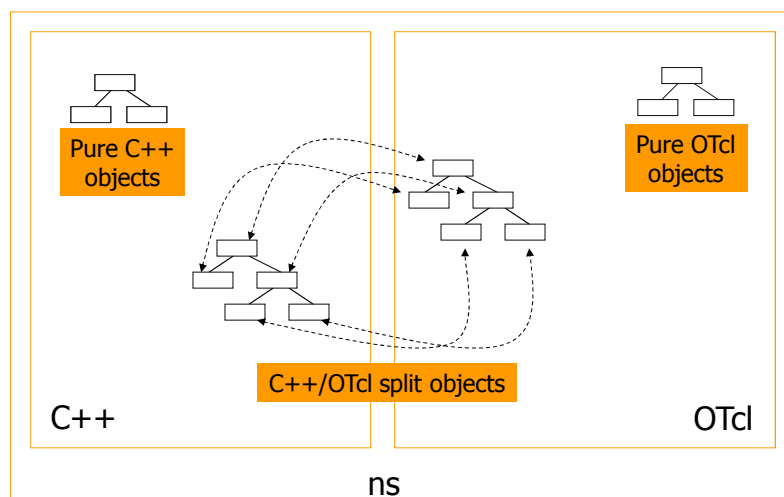
roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Interazioni tra classi C++ e classi OTcl

- Le **classi C++** che costituiscono ns-2 implementano l'insieme dei protocolli disponibili
- L'ambiente OTcl :
 - permette la definizione di **classi OTcl** direttamente connesse alle **classi C++** (*linkage*)
 - Consente di creare anche classi OTcl che non hanno corrispondente in C++
 - fornisce i metodi per l'utilizzo delle classi
- Il programmatore di script di simulazione può lavorare solo con classi OTcl oppure può sviluppare nuovi protocolli mediante nuove classi C++ da rendere accessibili in OTcl

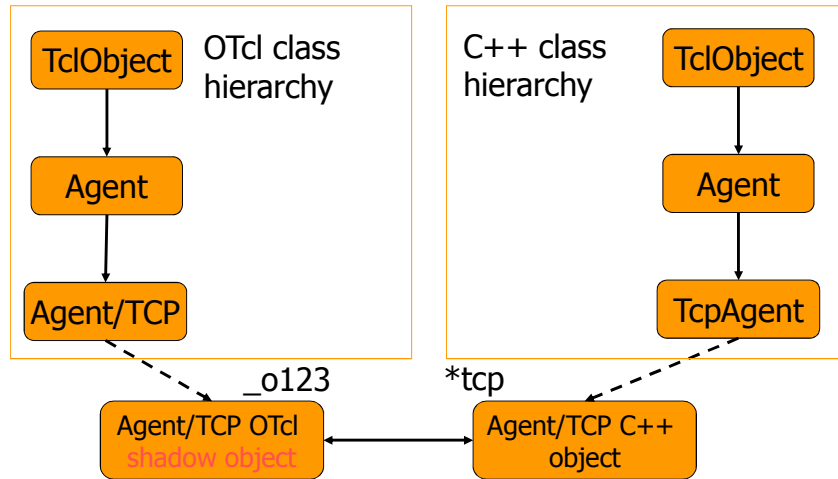


OTcl and C++: The Duality



TclObject: Hierarchy and Shadowing

71



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Creating New Components

72

- Extending ns in Otcl
- Extending ns in C++



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Extending ns in OTcl

- If you don't want to compile
 - source your changes in your sim scripts
- Modifying existing code
 - Recompile
- Adding new files
 - Change Makefile (NS_TCL_LIB),
 - Update tcl/lib/ns-lib.tcl
 - Recompile



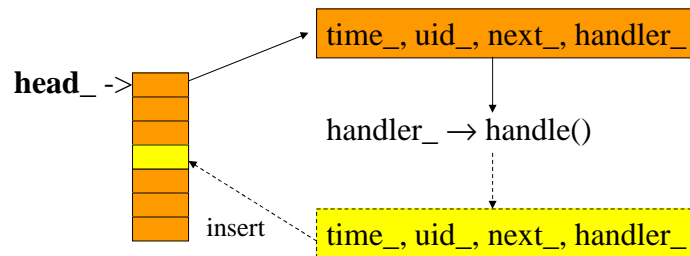
Extending ns in C++

- Modifying code
 - `make depend`
 - Recompile
- Adding code in new files
 - Change Makefile
 - `make depend`
 - Recompile



Gestione degli eventi dello schedatore

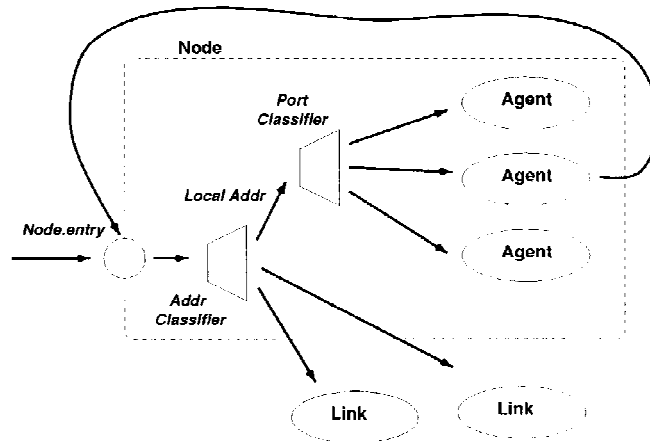
75



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

ns: struttura interna di un nodo

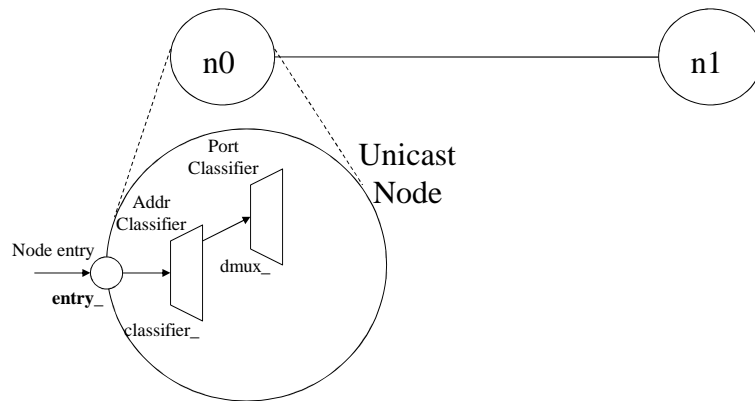
76



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Nodo: struttura interna

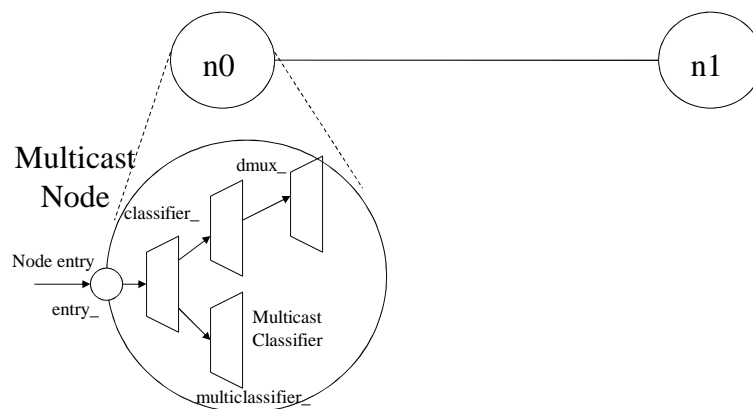
77



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

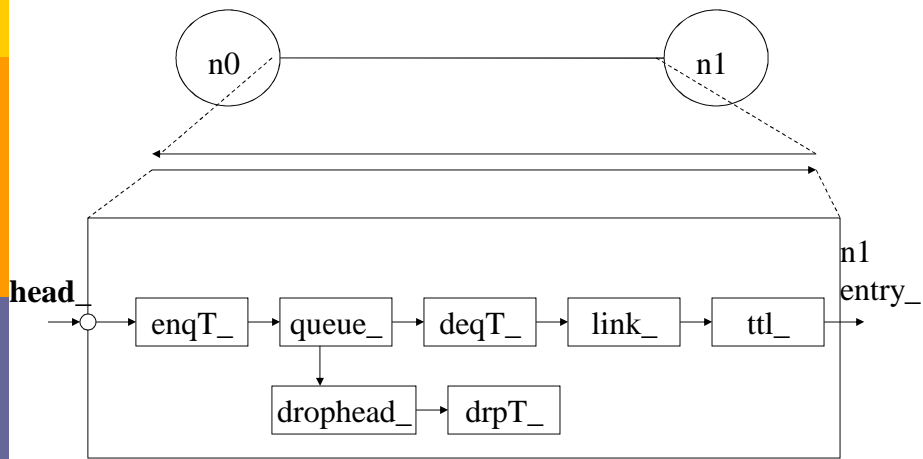
Nodo multicast

78

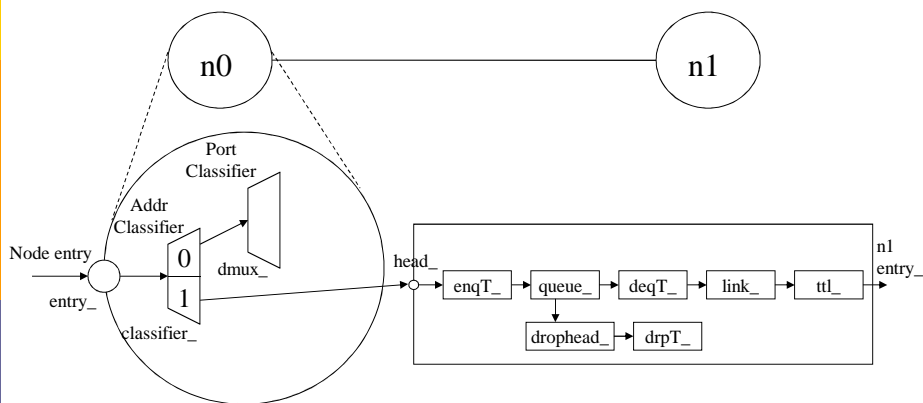


roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Link: struttura interna

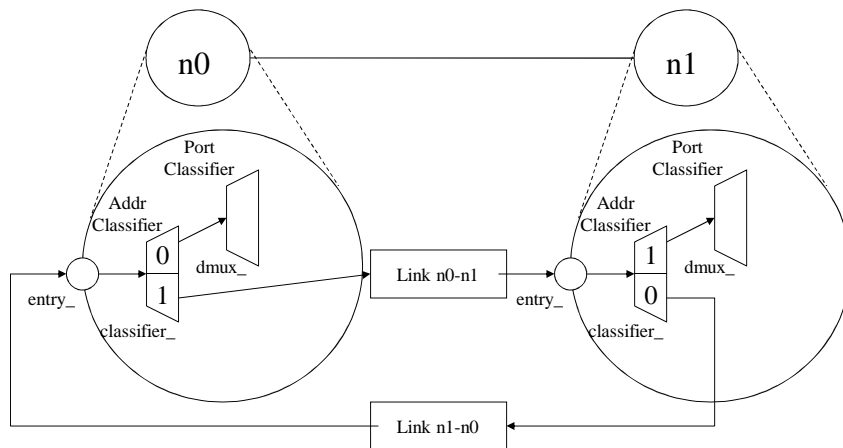


Collegamento nodo-link



Collegamento nodo-link (2)

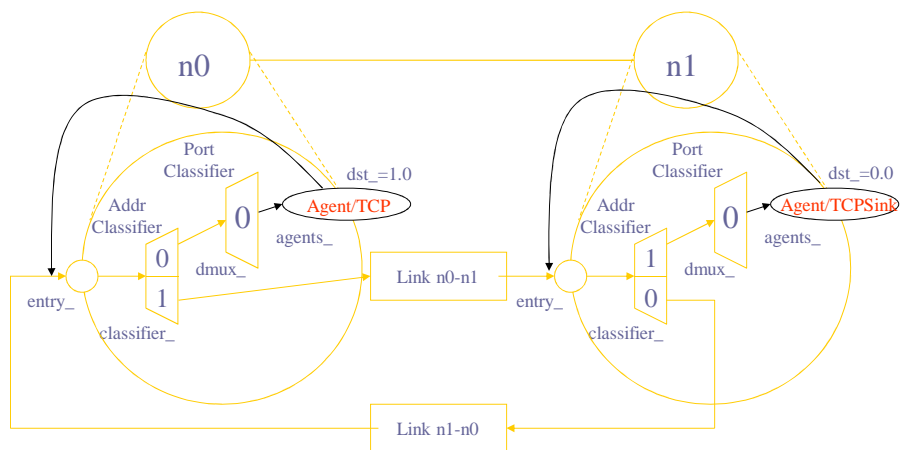
81



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Agenti di trasporto

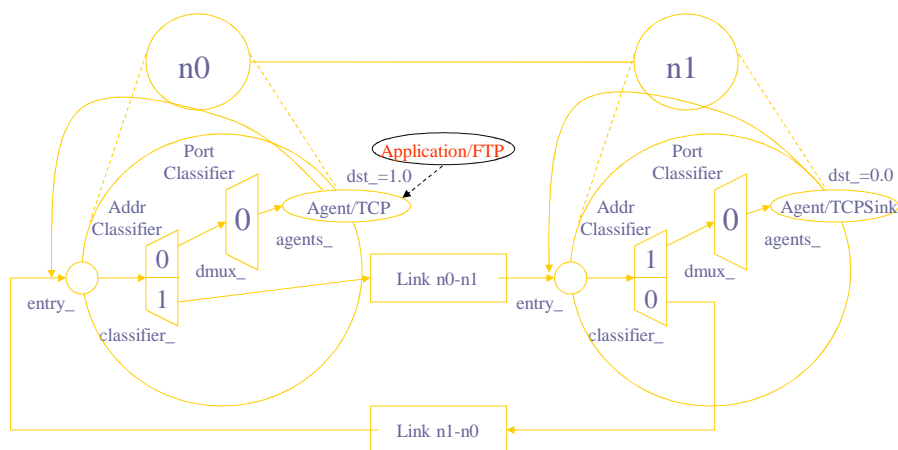
82



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Applicazione

83



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

La classe Agent

84

- L'agent non riceve realmente dati dall'applicazione ma solo la dimensione dell'unità informativa (bytes)
- Diversi tipi di agent per gestire diversi tipi di protocolli di trasporto (es. TCP, UDP,...)
- Per ogni protocollo di trasporto è definito:
 - Un agent **trasmettore**
 - Un agent **ricevitore**
- Al nodo ricevitore i pacchetti vengono "scartati" dall'agent ricevitore (libera la memoria associata alla struttura dati del pacchetto)
 - AGENT NULL (scarto)
 - AGENT TCPSink (scarto +ACK)



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

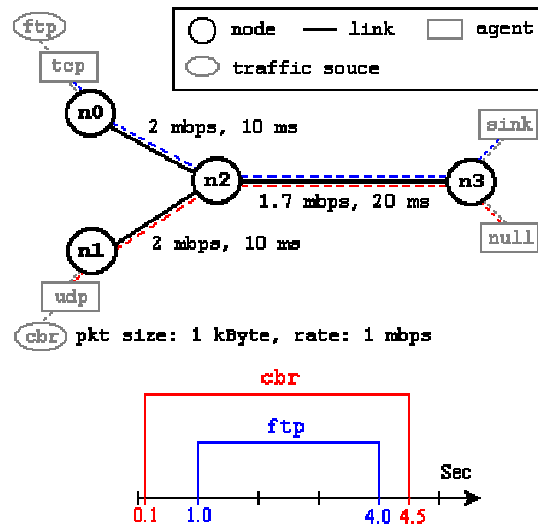
La classe Application

- Ha lo scopo di emulare le applicazioni più comuni e le caratteristiche di profili di traffico noti
- Le unità dati generate da oggetti application sono passati all'oggetto agent tramite funzioni di interfaccia
- Possibilità di:
 - Generare traffico tipico di applicazioni del mondo reale (FTP, WEB)
 - Configurare le caratteristiche dell'applicazione
- In ns2 l'applicazione non genera veri dati ma solo delle dimensioni di file
- I byte generati vengono incapsulati in segmenti TCP o pacchetti UDP dall'agent corrispondente

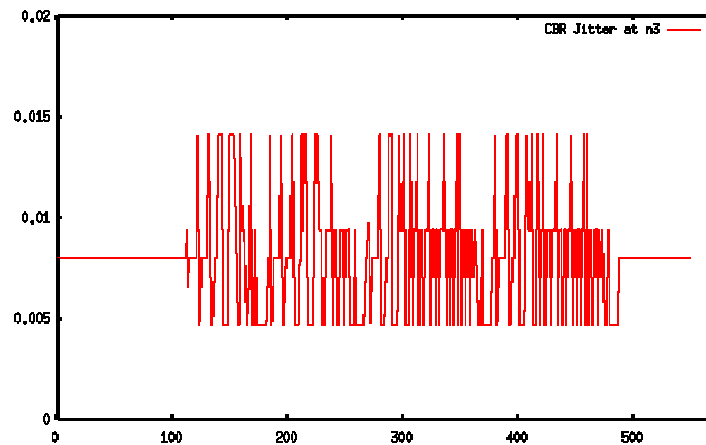


Esempi di script di simulazione

ns-simple.tcl



ns-simple-trace.tcl



bwxd.tcl

Questo file illustra il comportamento degli schemi di trasmissione a “finestra scorrevole” mediante una connessione TCP

- ns bwxd.tcl 10Mb 4ms 1 0.5
 - Finestra = 1 (Stop-and-Wait)
 - Throughput = $1 * 500 * 8 / 0.008 = 500$ kb/s
- ns bwxd.tcl 10Mb 4ms 10 0.5
 - Throughput = $10 * 500 * 8 / 0.008 = 5$ Mb/s
- ns bwxd.tcl 10Mb 4ms 40 0.5
 - Throughput = 10 Mb/s



Esempio

- Animazione [nam](#)



Estensione del simulatore: un nuovo agent

OTcl Linkage

92

- Let's create a new agent "MyAgent"
 - Dummy agent
 - Derived from the "Agent" class



Step 1: Export C++ class to OTcl

```
class MyAgent : public Agent {
public:
    MyAgent();
protected:
    int command(int argc, const char*const* argv);
private:
    int    my_var1;
    double my_var2;
    void  MyPrivFunc(void);
};
```

```
static class MyAgentClass : public TclClass {
public:
    MyAgentClass() : TclClass("Agent/MyAgent0tcl") {}
    TclObject* create(int, const char*const*) {
        return(new MyAgent());
    }
} class_my_agent;
```



Step 2 : Export C++ class variables to OTcl

```
MyAgent::MyAgent() : Agent(PT_UDP) {
    bind("my_var1_otcl", &my_var1);
    bind("my_var2_otcl", &my_var2);
}
```

- set the default value for the variables in the "ns-2/tcl/lib/ns-lib.tcl" file



Step 3: Export C++ Object Control Commands to OTcl

95

```
int MyAgent::command(int argc, const char*const* argv) {
    if(argc == 2) {
        if(strcmp(argv[1], "call-my-priv-func") == 0) {
            MyPrivFunc();
            return(TCL_OK);
        }
    }
    return(Agent::command(argc, argv));
}
```



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Step 4: Execute an OTcl command from C++.

96

```
void MyAgent::MyPrivFunc(void) {
    Tcl& tcl = Tcl::instance();
    tcl.eval("puts \"Message From MyPrivFunc\"");
    tcl.evalf("puts \"      my_var1 = %d\"", my_var1);
    tcl.evalf("puts \"      my_var2 = %f\"", my_var2);
}
```



roberto.canonico@unina.it
Università degli Studi di Napoli FEDERICO II

Step 5: Compile

- Save above code as “`ex-linkage.cc`”
- Open “`Makefile`”, add “`ex-linkage.o`” at the end of object file list.
- Re-compile NS using the “`make`” command.



Step 5: Run and Test “MyAgent”

`ex-linkage.tcl`

```
# Create MyAgent (This will give two warning messages that
# no default vaules exist for my_var1_otcl and my_var2_otcl)
set myagent [new Agent/MyAgentOtcl]

# Set configurable parameters of MyAgent
$myagent set my_var1_otcl 2
$myagent set my_var2_otcl 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```



Step 5: Run and Test "MyAgent"

result

```
warning: no class variable Agent/MyAgentOtc1::my_var1_otc1
      see tcl-object.tcl in tclcl for info about this warning.
warning: no class variable Agent/MyAgentOtc1::my_var2_otc1
Message From MyPrivFunc
  my_var1 = 2
  my_var2 = 3.140000
```



Riferimenti

- The Network Simulator ns-2 Home Page:
www.isi.edu/nsnam/ns
- The Network Simulator ns-2 Manual:
www.isi.edu/nsnam/ns/ns-documentation.html
- Tutorial:
www.isi.edu/nsnam/ns/tutorial/index.html
nile.wpi.edu/NS

