The Network Simulator NS2

casalicchio@ing.uniroma2.it

http://www.ce.uniroma2.it/courses/MMI/

http://www.uniroma2.it/didattica/MMI

Agenda

- Concetti fondamentali
- Eventi e Scheduler
- Architettura
- Esempio
- Nodi e Routing
- Workload e livello di applicazione
- Generazionde delle topologie di rete

Documentazione

- Questa lezione è basata sul materiale seguente
- http://www.isi.edu/nsnam/ns/
  - Marc Greis's tutorial
  - "NS for Beginners" by Altman and Jimenez
  - Ns Manual
  - “NS by Example” by J.Chung and M.Claypool http://nile.wpi.edu/NS/
  - Varie presentazioni e tutorial “ns workshops and presentations”
- Tcl/tk, http://www.tcl.tk/
- Otcl, http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/index.html
- Altri che troverete nel seguito…

Installazione

http://www.isi.edu/nsnam/ns/ns-build.html

All-in-one (Consigliata – Linux/Windows/Mac):

Ultima versione (Jun 17, 2009)

http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.34/

- Tcl release 8.4.18 (required component)
- Tk release 8.4.18 (required component)
- Otcl release 1.13 (required component)
- TclCL release 1.19 (required component)
- Ns release 2.33 (required component)
- Nam release 1.13 (optional component)
- Xgraph version 12 (optional component)
- CWeb version 3.4g (optional component)
- SGB version 1.0 (?) (optional component, builds sgblib for all UNIX type platforms)
- Gt-itm gt-itm and sgb2ns 1.1 (optional component)
- Zlib version 1.2.3 (optional, but required should Nam be used)

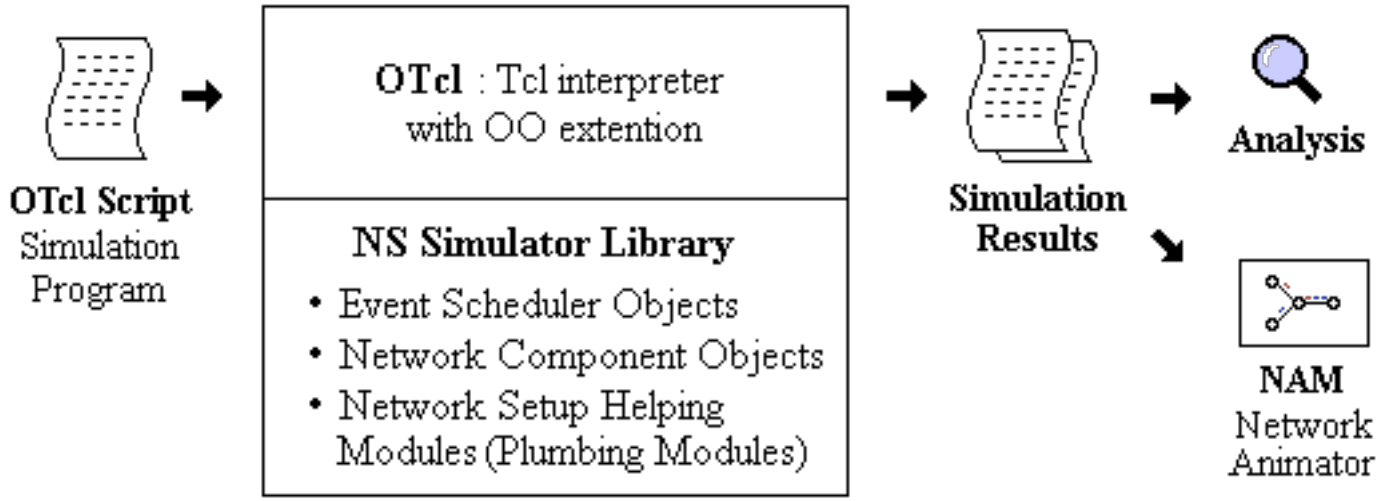
Oppure…

Installazione e configurazione separata dei singoli componenti

Cos'è NS2

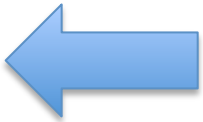
- **Discrete-event simulator**
  - Sviluppato all' UC Berkeley
- Simulazione a **livello di pacchetto**
- Modellazione dal livello **Data Link** a livello **Applicazione**
  - protocolli livello MAC (per simulazioni LAN)
  - Algoritmi di routing: Dijkstra, etc…
  - Meccanismi di gestione delle code dei router: Drop Tail, Random Early Detection/Drop (RED) e (Class-Based Queueing) CBQ
  - Protocolli di rete: TCP, UPD (over IP e IPv6)
  - Sorgenti di traffico: FTP, Telnet, Web, CBR e VBR,
- Open source (vasta comunità di sviluppatori ed utenti)
- Basato su
  - **C++** per simulation engine e modelli
  - **Tcl/OTcl (Object Tool Command Language)** per configurazione scenari e logica di simulazione

Vista utente: concetti fondamentali



- Otcl per setup ed esecuzione simulazione
  - inizializzazione event scheduler,
  - Setup topologia utilizzando “network object” e “plumbing function”
  - Specificare inizio e fine generazione traffico (eventi)
- **Evento** = (**packetID**, **Schedule Time**, **Obj Pointer**)
  - L’**Obj** è un network object che gestirà il pacchetto
- L’**Event Scheduler**
  - Mantiene il tempo simulato
  - Estrae **(fire)** tutti gli eventi (pacchetti) che occorrono al tempo **t** invocando i relativi componenti di rete (network object/component)
  - Lo scheduler è single thread (no problemi locking/competizione risorse)
- I componenti di rete comunicano scambiandosi pacchetti

Agenda

- Introduzione
- Eventi e Scheduler 
- Architettura
- Esempio
- Routing
- Workload e livello di applicazione
- Generazionde delle topologie di rete

Struttura dati Event

- un generico evento comprende generalmente un firing time e una funzione manipolatrice (detta “handler”)
- La classe denominata Handler contiene semplicemente una *funzione virtuale, che quindi dovrà essere specializzata tramite le classi da essa derivate*

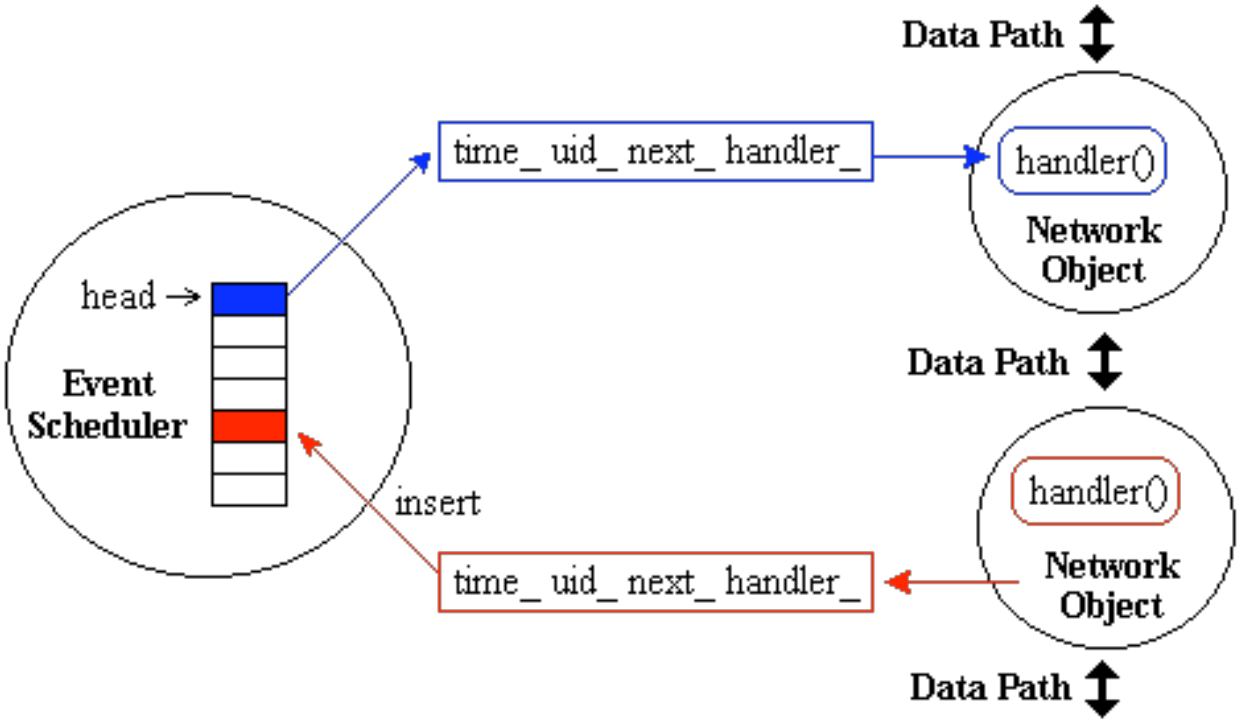
```
class Event
{
   public:
        Event* next_;           /* event list */
        Handler* handler_;      /* handler to call when event ready */
        double time_;           /* time at which event is ready */
        int uid_;          /* unique ID */

        Event() : time_(0), uid_(0) {}

};
/*
* The base class for all event handlers. When an event's scheduled
* time arrives, it is passed to handle which must consume it.
* i.e., if it needs to be freed it, it must be freed by the handler.
*/

class Handler
{
   public:
        virtual void handle(Event* event);
};
```

Event Scheduler



A seconda di come gestisco la coda ho diverse prestazioni, e.g

1) massimizzare il numero di eventi che riesco a processare nell'unità di tempo (reale), e quindi aumentare la velocità della simulazione
2) massimizzare la quantità di eventi che riesco a gestire, indipendentemente dalla loro velocità di processamento (dimensione della coda!!), per aumentare la dimensione dei sistemi che riesco a simulare
3) 1) e 2) strettamente correlati

Scheduler in NS-2

- List scheduler
- Heap scheduler
- Calendar scheduler (default)
- Real-time scheduler

Es:
set ns [new Simulator]
$ns use-scheduler Heap

- **“All made the same task with different performances”**

List Scheduler

- The list scheduler (Scheduler/List../ns-2/scheduler.cc) implements the scheduler using a **simple linked-list structure**
- The **list is kept in time-order** (earliest to latest),
  - so event insertion and deletion require scanning the list to find the appropriate entry
- Choosing the next event for execution requires trimming the first entry off the head of the list
  - This implementation preserves event execution in a FIFO manner for simultaneous events.

Heap Scheduler

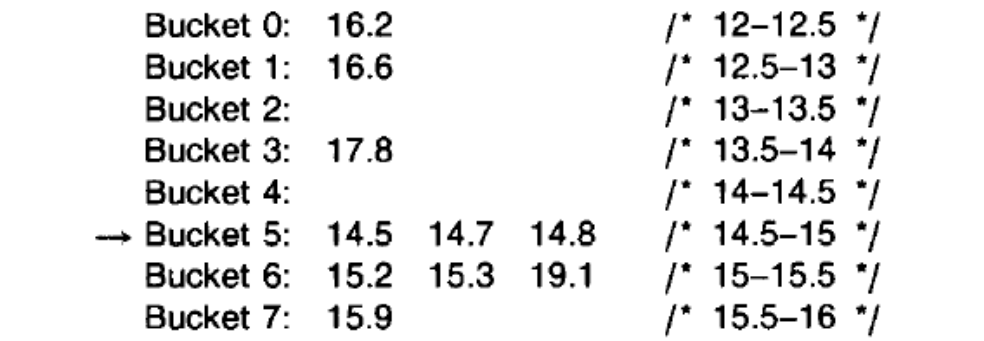
- The heap scheduler (Scheduler/Heap../ns-2/scheduler.cc) implements the scheduler using a heap structure.
- A heap is a specialized tree-based data structure that satisfies the heap property: if B is a child node of A, then key(A) ≥ key(B).
- This structure is superior to the list structure for a large number of events, as insertion and deletion times are in ***O(log n)*** for ***n*** events.

Calendar scheduler

- The calendar queue scheduler (Scheduler/Calendar../ns-2/scheduler.cc) uses a data structure analogous to a one-year desk calendar, in which events on the same month/day of multiple years can be recorded in one day.
- The original implementation of Calendar queues in ns v2 was contributed by David Wetherall.
  - A new implementation from Xi (2005) http://netlab.caltech.edu/projects/ns2tcplinux/ns2patch/
- The original NS-2.28 uses a CalendarQueue to implement the priority queue, which store the future events in the simulation.

Original Calendar scheduler

- The calendar queue stores events into an array of buckets.
  - A bucket corresponds to a "day" in a real calendar.
  - A bucket can hold multiple events, as you can write down multiple notes in each day in a real calendar.
  - The whole bucket array corresponds to a "year". If events in the same "day" but in different "years" share the same bucket.
  - When a new event is inserted, we can calculate the right bucket in O(1) and insert the event into this bucket via linear search.
  - The size of the array may be doubled if the number of events grows larger, or halved if the number of events grows smaller.



The current year begins at 12.0 and ends at 16.0. Bucket 5 is the current date. Note that events scheduled in buckets 0 through 4 are scheduled for next year and are in the range 16–20. 0.5 is the length of a day. The numbers on the right are the time ranges for each day in the current year. Note that the event at 19.1 in bucket 6 is scheduled for next year.

R. Brown. 1988. Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem. *Commun. ACM 31, 10 (October 1988), 1220-1227.*

Original Calendar scheduler

- The efficiency of the calendar queue seems to depend on the width of each bucket.
  - **If the width of a bucket is too long**, many events may be put into one bucket where linear search happens when a new event is inserted
  - **If the width of a bucket is too small**, most of the events in the buckets are of different years and there is a large overhead for dequeue (linear search over the calendar days).

Original Calendar scheduler

- The optimal solution for bucket width, is the average interval between adjacent events.
- To calculate the average interval between adjacent events, one has to traverse the whole queue and calculate the average intervals.
- NS-2 calculates the average interval between adjacent events:

  1) pickup the fullest bucket (with largest number of events),

  2) calculate the average interval in that bucket, as suggested in Dynamic Calendar Queue

  3) set bucket size = average interval*4. (4 is probably the heuristic number that keep the inqueue and dequeue complexity balanced)

- Steps 1-3 may result in a width value much larger than the suggested optimial value.
- If in the fullest bucket there are some events in different "years" and most of the events are clustered within "seconds"
  - each bucket will have a width in unit of "years" and
  - most of the events (clustered within seconds) will go into a few buckets of "years".
- NS2 does not adopt the Dynamic Calendar Queue's resize triggering algorithm.
  - Once the bucket width is set to be a value, it won't change unless the bucket number needs to be changed (by significant change on number of events in the queue)

JongSuk Ahn, SeungHyun Oh. "Dynamic Calendar Queue,"

Speeding up NS-2 scheduler

- A new implementation from Xi (2005)
  - http://netlab.caltech.edu/projects/ns2tcplinux/ns2patch/
- Three changes to the original Calendar Scheduler:

  1) The estimation of the *bucket_size:* ***The patch estimates the bucket_size by the average interval of dequeued events in the past window. This is closer to the real distribution unless the event arrival pattern significantly changes.***

  2) SNOOPy Calendar Queue dynamic adjustment is implemented to react to the change of event arrival pattern.

  Kah Leong Tan, Li-Jin Thng, "SNOOPy Calendar Queue", Proceedings of the 32nd conference on Winter simulation, Orlando, Florida, Pages: 487 - 495, 2000,

  3) For the "insert" function, the original code traverses from the head to the tail. **The patch traverses from the tail to the head, which may traverse less nodes if events are often inserted to the end of the priority queue.**

Confronto di prestazioni tra schedulers

Test	Scenario	NS-2.28 original	NS-2.28 with the patch*	NS-2.28 with Heap**	NS-2.28 with map ***
1	c= 1Gbps buffer=20000pkt d=220ms flow#= 40 no output	17h 9m 12s	25m 6s	40m 15s	1h 8m 2s
5	c= 1Gbps buffer=10000pkt d=220ms flow#= 40 no output	23m 37s	24m 30s	40m 82s	1h 4m 38s
10	c= 250Mbps buffer=125pkt d=220ms flow#= 40 no output	2m 52s	2m 9s	3m 0s	5m 28s

Xi (2005) http://netlab.caltech.edu/projects/ns2tcplinux/ns2patch/

Real Time scheduler

- The real-time scheduler (class Scheduler/RealTime) attempts to synchronize the execution of events with real-time.
  - It is currently implemented as a subclass of the list scheduler.
- The real-time capability is used to introduce an NS simulated network into a real-world topology to experiment with easily-configured network topologies, cross-traffic, etc.
  - This only works for relatively slow network traffic data rates, as the simulator must be able to keep pace with the real-world packet arrival rate, and this synchronization is not presently enforced.

Time Precision

- **Precision of the scheduler clock can be defined as the smallest time-scale of the simulator that can be correctly represented**.
- The **clock variable** for ns is represented by a **double**.
  - As per the IEEE std for floating numbers, a double, consisting of 64 bits must allocate the following bits between its sign, exponent and mantissa fields: sign=1bit exponent =11bit mantissa=52bit
  - Any floating number can be represented in the form **X * 2^n** where **X** is the mantissa and **n** is the exponent.
- Thus the precision of timeclock in ns can be defined as
  - **$1/(2^{52})$=2.2204e-16 sec**.

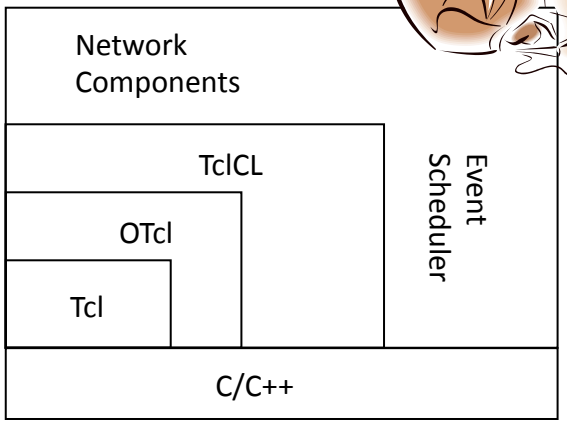
Agenda

- Concetti fondamentali
- Eventi e Scheduler
- Architettura 
- Esempio
- Routing
- Workload e livello di applicazione
- Generazionde delle topologie di rete

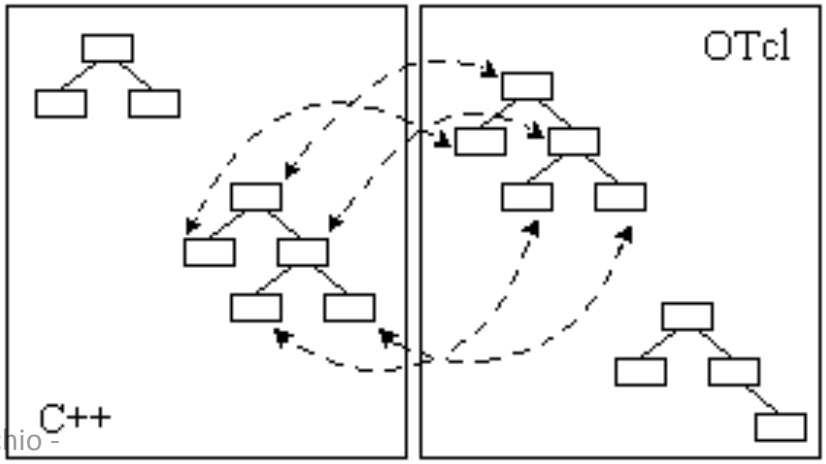
Architettura

- C++ usato per processare I pachetti:
  - Scheduler e Network components scritti in C++
  - Veloce, dettagliato, controllo completo
- Otcl usato per il controllo,
  - setup della simulazione, configurazione, azioni occasionali
  - Rapido da scrivere e modificare
- C++ e OTcl condividono la stessa gerarchia di classi.
  - Corrispodenza (OTcl linkage - TclCL) tra oggetti C++ e Otcl

    C++ obj – TclCL – Otcl obj
  - Solo per gli oggetti che modellano network component



ns-2



Architettura: Overview

- The simulator supports
  - A class hierarchy i nC++, and
  - A similar class hierarchy within the Otcl interpreter
- The two hierarchies are closely related to eachother; from the user's perspective, there is a one-to-one correspondence
  - The root of this hierarchy is the class Tcl Object
- Users create new simulator objects through the interpreter
  - These objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass.
  - User instantiated objects are mirrored through methods defined in the class TclObject.
  - There are other hierarchies in the C++ code and OTcl scripts; these other hierarchies are not mirrored in the manner of TclObject.

Classi principali

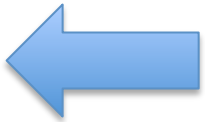
- **TclObject** is the base class for most of the other classes in the interpreted and compiled hierarchies.
  - Every object in the class TclObject is created by the user from within the interpreter.
  - An equivalent shadow object is created in the compiled hierarchy. The two objects are closely associated with each other.
  - The class TclClass contains the mechanisms that perform this shadowing.

```
set srm [new Agent/SRM/Adaptive]
$srm set packetSize_ 1024
$srm traffic-source $s0
```

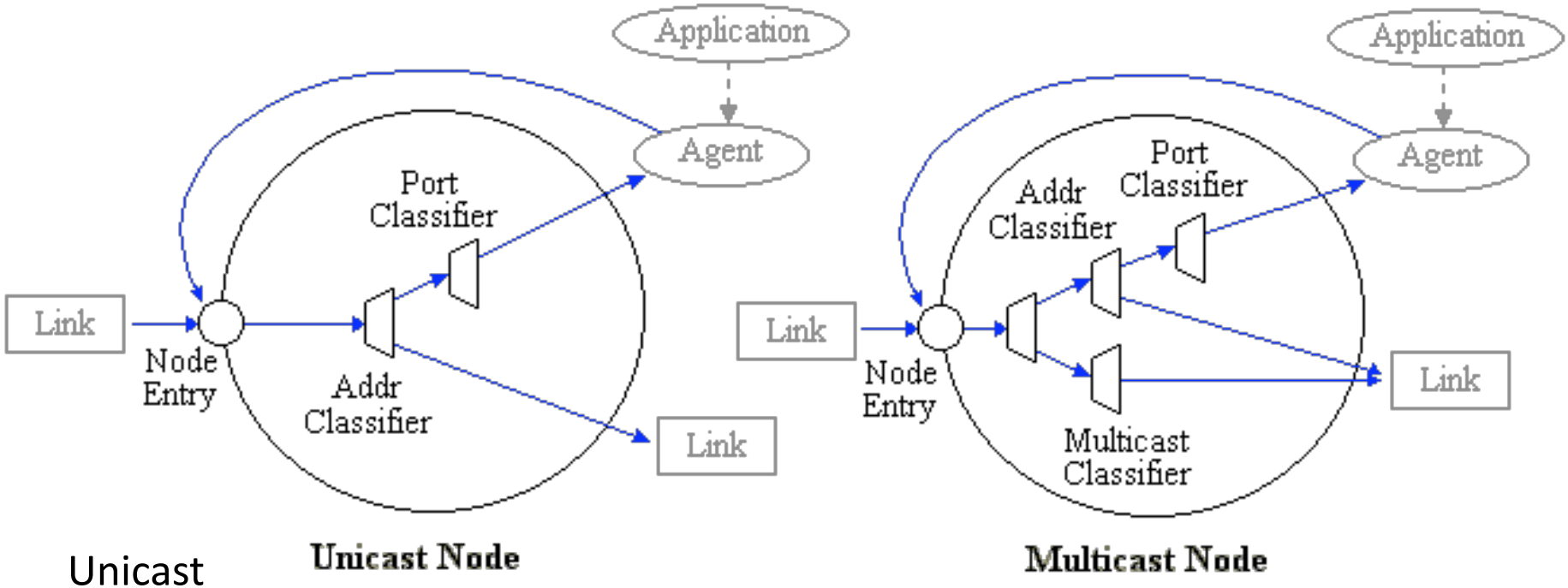
- **TclClass** is a pure virtual class.
  - Classes derived from this base class provide two functions: construct the interpreted class hierarchy to mirror the compiled class hierarchy; and
  - provide methods to instantiate new TclObjects.
  - Each such derived class is associated with a particular compiled class in the compiled class hierarchy, and can instantiate new objects in the associated class.

```
static class RenoTcpClass: public TclClass {
public:
        RenoTcpClass() : TclClass("Agent/TCP/Reno") {}
        TclObject* create(int argc, const char*const* argv) {
                return (new RenoTcpAgent());
        }
} class_reno;
```

Agenda

- Introduzione
- Eventi e Scheduler
- Architettura
- Esempio
- Nodi e Routing 
- Workload e livello di applicazione
- Generazionde delle topologie di rete

Node and Routing



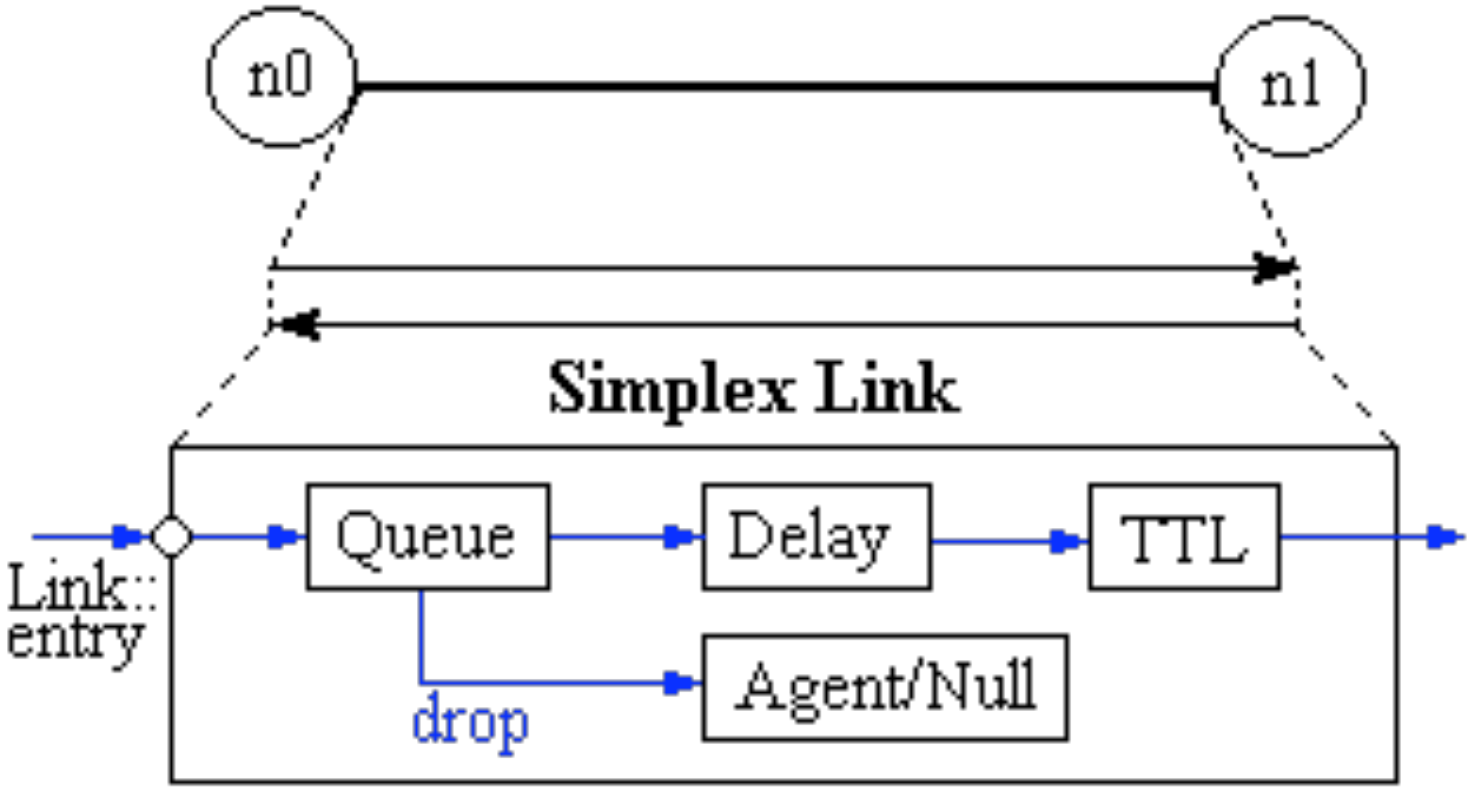
Unicast

- $ns rtproto type
  (type: Static, Session, Distance Vector, cost, multi-path)

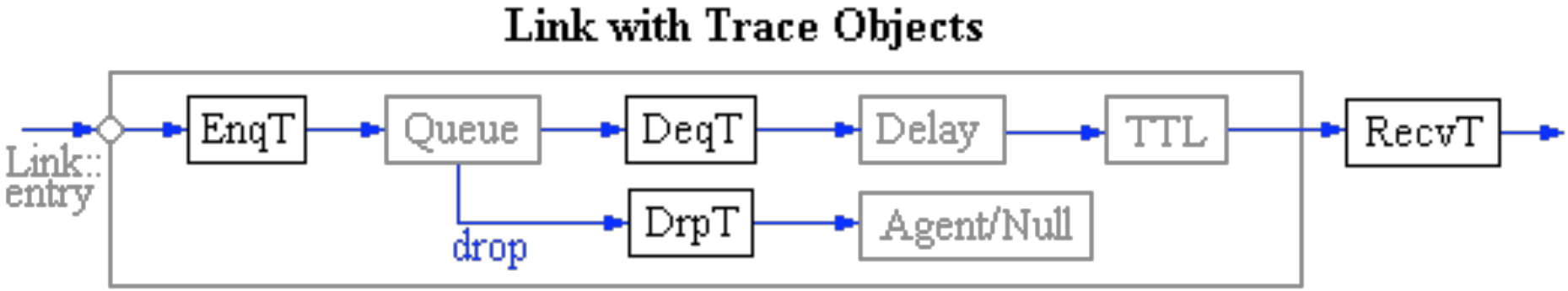
Multicast

- $ns multicast (right after set $ns [new Scheduler])
- $ns mrtproto type
  (type: CtrMcast, DM, ST, BST)

Link

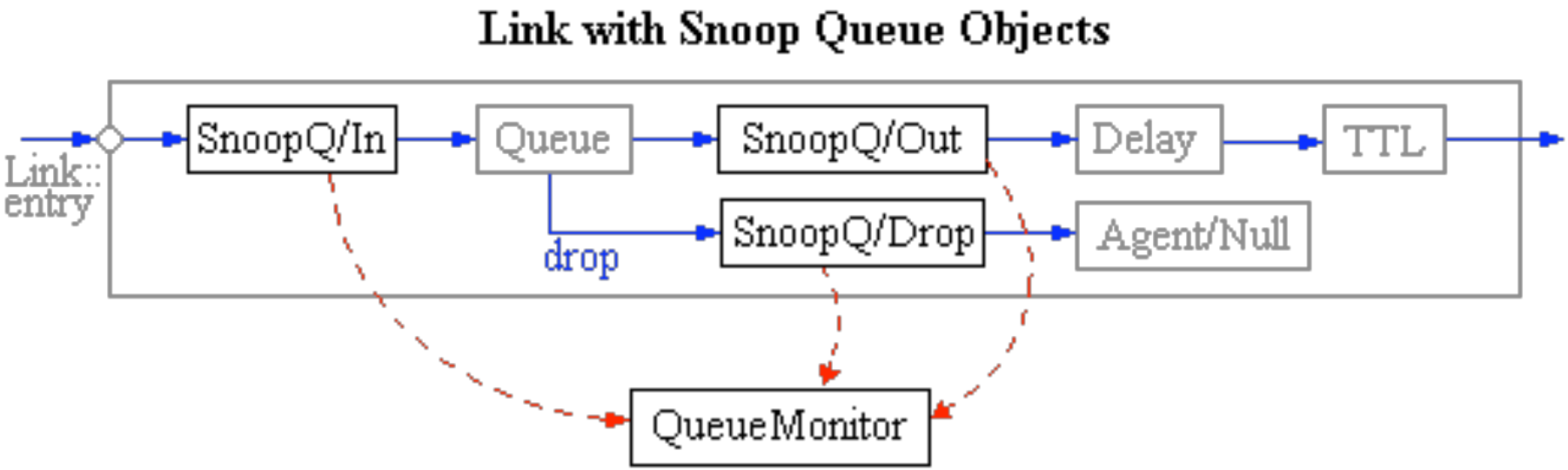

n0
n1
Simplex Link
Queue
Delay
TTL
Link::
entry
drop
Agent/Null

Tracing



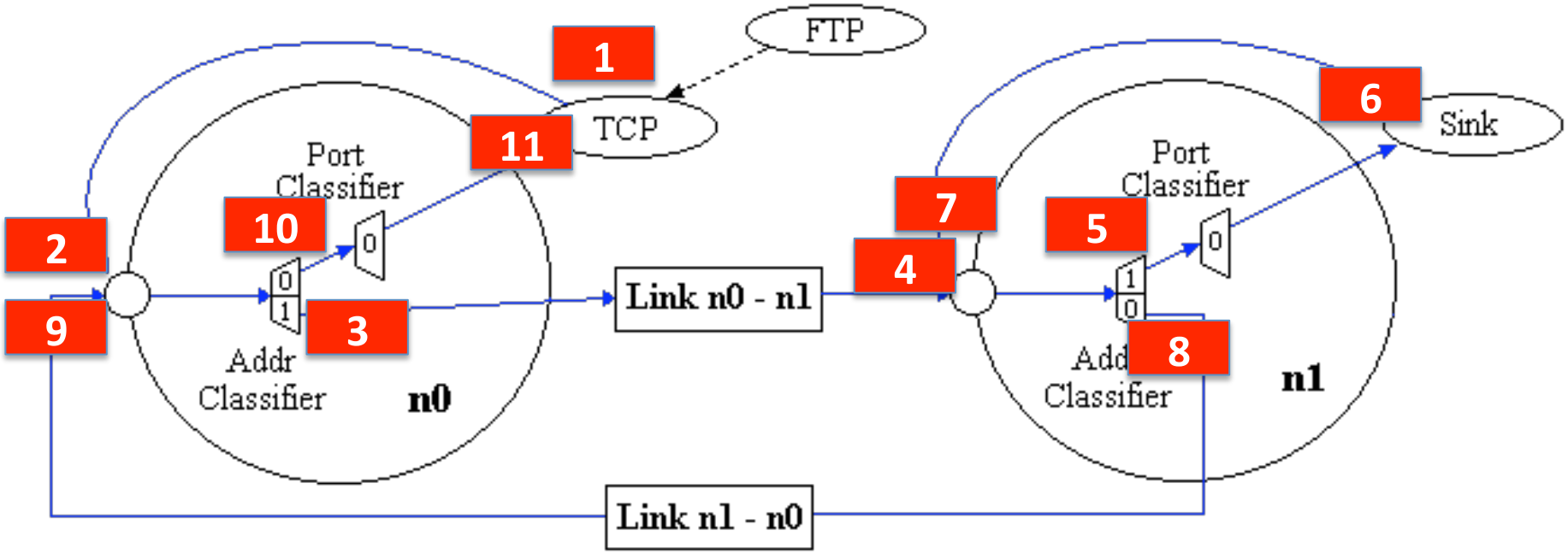
- $ns trace-all file or
- $ns namtrace-all file
- $ns create-trace {type file src dst}

Queue Monitor



Emiliano Casalicchio -
emiliano.casalicchio@uniroma2.it

Packet Flow



Emiliano Casalicchio -
emiliano.casalicchio@uniroma2.it

Agenda

- Concetti fondamentali
- Eventi e Scheduler
- Architettura
- Esempio 
- Routing
- Workload e livello di applicazione
- Generazionde delle topologie di rete

Basic tcl

```
proc test { a b } {
   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]"
     }
   }
}

test 43 27

%oppure

set aa 43
set bb 27
Test $aa $bb
```

Output

```
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
```

Hello World - Interactive Mode

```
swallow 71% ns
% set ns [new Simulator]
_o3
% $ns at 1 “puts \“Hello World!\””
1
% $ns at 1.5 “exit”
2
% $ns run
Hello World!
swallow 72%
```

Hello World - Batch Mode

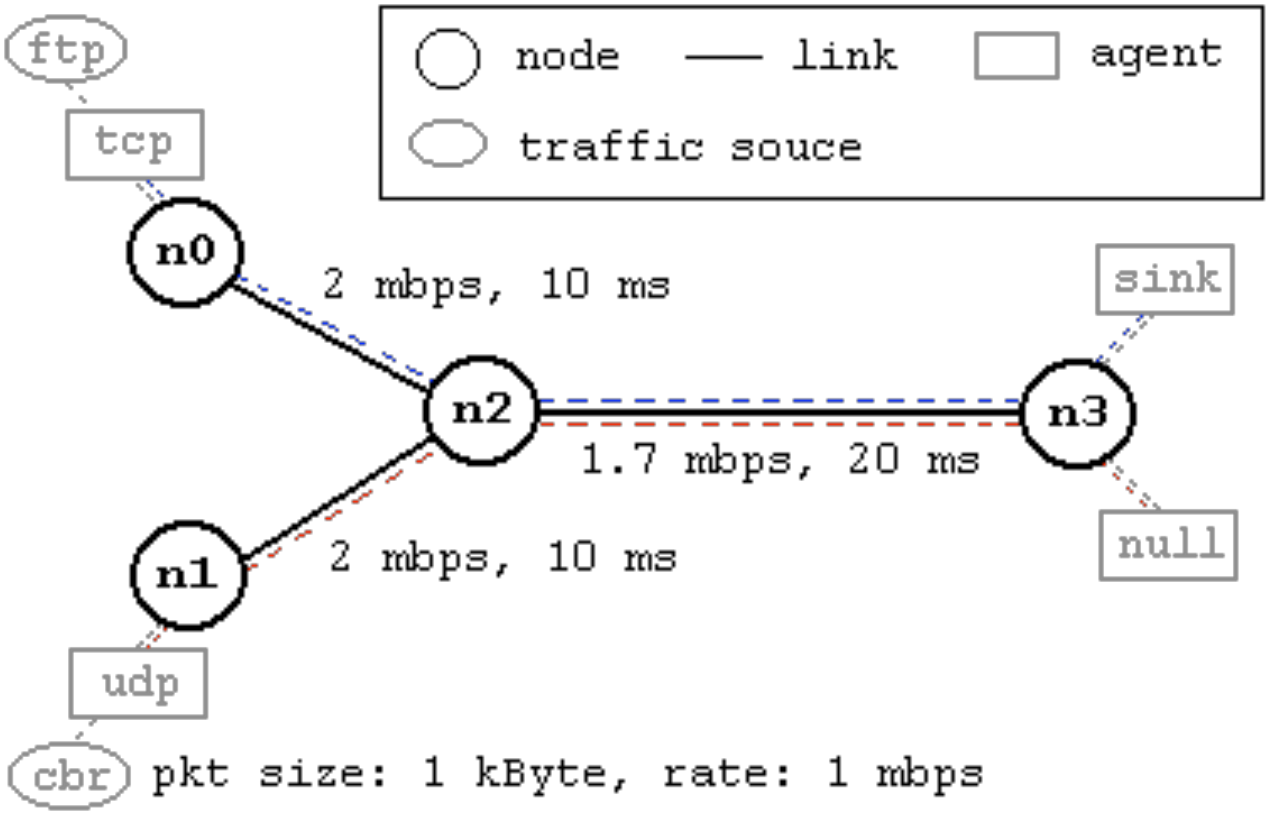
simple.tcl

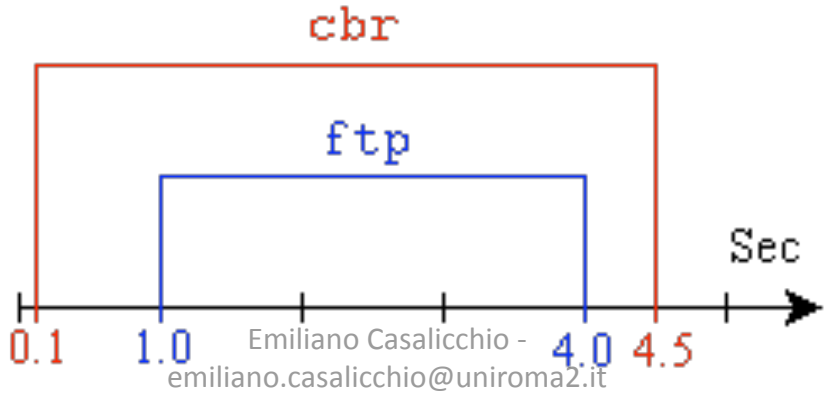
```
set ns [new Simulator]
$ns at 1 “puts \“Hello World!\””
$ns at 1.5 “exit”
$ns run
```

```
swallow 74% ns simple.tcl
Hello World!
swallow 75%
```

Emiliano Casalicchio -
emiliano.casalicchio@uniroma2.it

Esempio (ex-simple.tcl)





Instance simulation object

- **set *ns* [new Simulator]:** generates an NS simulator object instance, and assigns it to variable ***ns***. To instatiate a Simulator object means:
  - Initialize the packet format
  - Create a scheduler (default is **calendar scheduler**)
  - Select the default address format
- The "Simulator" object has member functions to:
  - Create compound objects such as nodes and links
  - Connect network component objects created
  - Set network component parameters (mostly for compound objects)
  - Create connections between agents
  - Specify NAM display options
- The "Simulator" object member function implementations are located in the "ns-2/tcl/lib/ns-lib.tcl" file.

NAM setup

- **$ns color *fid color*:**
  - is to set color of the packets for a flow specified by the flow id (f***id***).
  - is for the NAM display, and has no effect on the actual simulation.
- **$ns namtrace-all *file-descriptor***:
  - tells the simulator to record simulation traces in NAM input format.
  - gives the file name that the trace will be written to later by the command **$ns flush-trace**.
  - Similarly, the member function trace-all is for recording the simulation trace in a general format.

Network object setup

- **proc *finish* {}**:
  - function that specify post-simulation processes
  - is called after the simulation is over by the command **$ns at 5.0 "finish"**
- **set *n0* [$*ns* node]**:
  - creates a node. A node in NS is compound object made of address and port classifiers
  - Users can create a node by separately creating an address and a port classifier objects and connecting them together.
  - To see how a node is created, look at the files: "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-node.tcl".

- **$ns duplex-link *node1 node2 bandwidth delay queue-type***:
  - creates two simplex links of specified bandwidth and delay, and connects the two specified nodes.
  - In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links.
  - In the above simulation script, DropTail queue is used.
  - Link source codes can be found in "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl" files.
  - the user can insert error modules in a link component to simulate a lossy link

- **$ns queue-limit *node1 node2 number***:
  - sets the queue limit of the two simplex links that connect node1 and node2 to the number specified.
  - take a look at "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl", or NS documentation for more information.
- **$ns duplex-link-op *node1 node2 ...***:
  - Effect the NAM animation showing flows of packets
  - To see the effects of these lines, users can comment these lines out and try the simulation.

Setup Traffic Agents

- **set *tcp* [new *Agent/TCP*]**:
  - creates a TCP agent. But in general, users can create any agent or traffic sources in this way.
  - Agents and traffic sources are in fact basic objects (not compound objects), mostly implemented in C++ and linked to OTcl. Therefore, there are no specific Simulator object member functions that create these object instances.
  - To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agnet/TCPSink, Application/FTP and so on).
  - This information can be found in the NS documentation or partly in this documentation. But one shortcut is to look at the "ns-2/tcl/libs/ns-default.tcl" file. This file contains the default configurable parameter value settings for available network objects.

- **$*ns* attach-agent *node agent***:
  - attaches an agent object created to a node object. Actually, what this function does is call the attach member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, $n0 attach $tcp. Similarly, each agent object has a member function attach-agent that attaches a traffic source object to itself.

- **$*ns* connect *agent1 agent2***:
  - to establish a logical network connection between two agents. This line establishes a network connection by setting the destination address to each others' network and port address pair.

Scenario setup

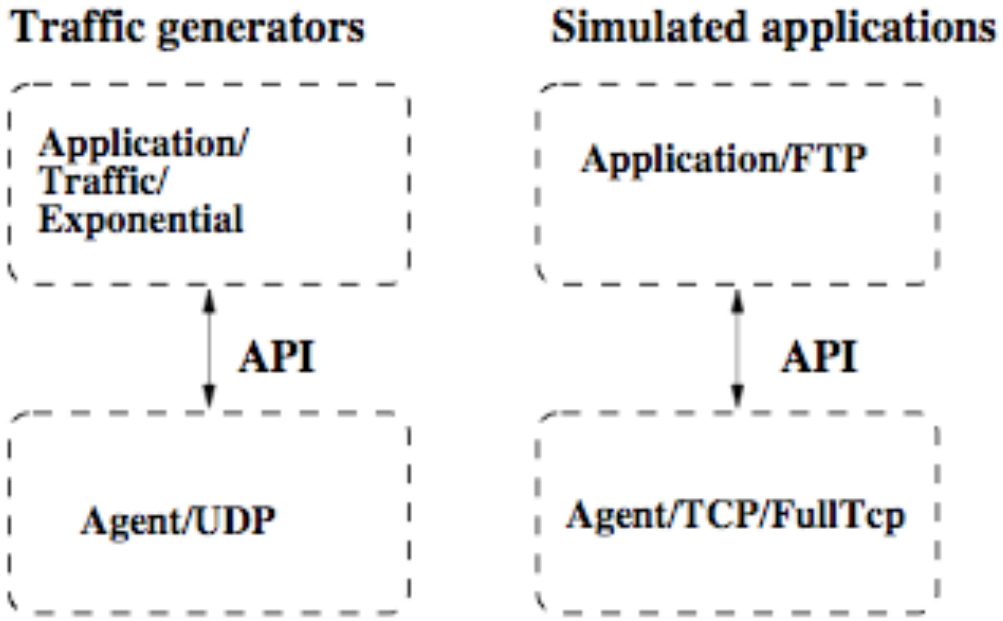
- **$ns at *time* "*string*"**:
  - makes the scheduler (scheduler_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time.
  - For example, **$ns at 0.1 "$cbr start"** will make the scheduler call a **start** member function of the CBR traffic source object, which starts the CBR to transmit data.
- **In NS, usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit, and the agent, just knowing how much of the data to transfer, creates packets and sends them.**

Agenda

- Introduzione
- Eventi e Scheduler
- Architettura
- Esempio
- Nodi e Routing
- Generazione del Workload 
- Generazionde delle topologie di rete

Application

- App. to Agent: system call
  - send
  - sendmsg
  - close
  - listen
- Agent to App.: up call
  - recv
  - resume



- **in *ns* there is no actual data being passed between applications**

Type of application

- HttpApp
- Application/
  - TcpApp
  - FTP
  - Telnet
- Application/Traffic/*
  - Exponential On/Off
  - Pareto On/Off
  - CBR
  - Traffic trace

Exponential On/Off

- An Exponential On/Off object is embodied in the OTcl class Application/Traffic/Exponential.
- The member variables that parameterize this object are:
  - packetSize_ the constant size of the packets generated
  - burst_time_ the average “on” time for the generator
  - idle_time_ the average “off ” time for the generator
  - rate_ the sending rate during “on” times
- A new Exponential On/Off traffic generator can be created and parameterized as follows:

```
set e [new Application/Traffic/Exponential]
$e set packetSize_ 210
$e set burst_time_ 500ms
$e set idle_time_ 500ms
$e set rate_ 100k
```

Pareto On/Off

- A Pareto On/Off object is embodied in the OTcl class Application/Traffic/Pareto.
- The member variables that parameterize this object are:
  - packetSize_ the constant size of the packets generated
  - burst_time_ the average "on" time for the generator
  - idle_time_ the average "off" time for the generator
  - rate_ the sending rate during "on" times
  - shape_ the "shape" parameter used by the pareto distribution
- A new Pareto On/Off traffic generator can be created as follows:

```
set p [new Application/Traffic/Pareto]
$p set packetSize_ 210
$p set burst_time_ 500ms
$p set idle_time_ 500ms
$p set rate_ 200k
$p set shape_ 1.5
```

CBR

- A CBR object is embodied in the OTcl class Application/Traffic/CBR.
- The member variables that parameterize this object are:
  - rate_ the sending rate
  - interval_ (Optional) interval between packets
  - packetSize_ the constant size of the packets generated
  - random_ flag indicating whether or not to introduce random “noise” in the scheduled departure times (default is off)
  - maxpkts_ the maximum number of packets to send

```
set e [new Application/Traffic/CBR]
$e set packetSize_ 48
$e set rate_ 64Kb
$e set random_ 1
```

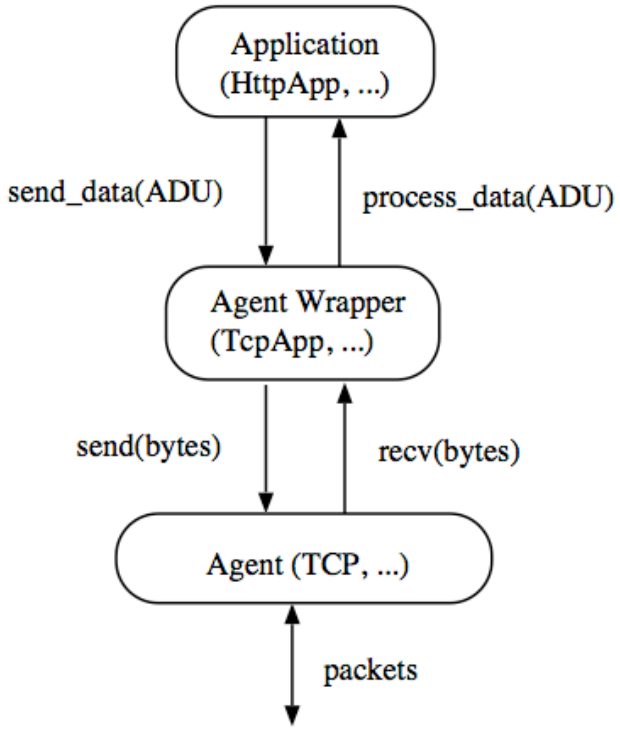
Traffic Trace

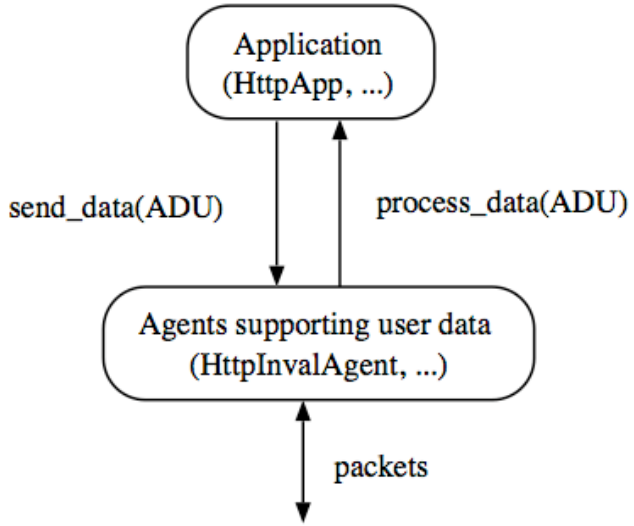
- A Traffic Trace object is instantiated by the OTcl class Application/Traffic/Trace.
  - The associated class Tracefile is used to enable multiple Traffic/Trace objects to be associated with a single trace file.
  - The Traffic/Trace class uses the method attach-tracefile to associate a Traffic/Trace object with a particular Tracefile object.
  - The method filename of the Tracefile class associates a trace file with the Tracefile object.
- The following example shows how to create two Ap- plication/Traffic/Trace objects, each associated with the same trace file (called "example-trace" in this example).
- To avoid synchronization of the traffic generated, random starting places within the trace file are chosen for eachTraffic/Trace object.

```
set tfile [new Tracefile]
$tfile filename example-trace

set t1 [new Application/Traffic/Trace]
$t1 attach-tracefile $tfile
set t2 [new Application/Traffic/Trace]
$t2 attach-tracefile $tfile
```

Web traffic (Web cache model)





- There are three major classes related to web cache, as it is in the real world:
  - client (browser),
  - server, and
  - cache

Web Page and PagePool

- Web Page
  - name, size, modification time, age
- Page Pool
  - Math
    - one page, whose **size can be generated by a given random variable**.
    - **Page modification sequence** and **request sequence** are generated using two given random variables.
  - CompMath
    - introduces **a compound page model**.
    - a compound page = main page + several component objects.
    - All component objects have the same size;
  - ProxyTrace
    - [<serverID> <URL_ID> <PageSize> <AccessCount>]
  - WebTraffic (to generate traffic only and not header info)

Esempio

- simple-webcache.tcl

Internet topologies

- To effectively engineer the Internet, crucial issues such as the large scale structure of its underlying physical topology,
  - its time evolution and the contribution of its individual components to its overall function need to be well understood.
- During the **design phase of an Internet-based technology**, extensive simulations are usually performed to assess its feasibility, in terms of efficiency and performance.
- In general, **Internet studies and simulations assume certain topological properties or use synthetically generated topologies.**
- If such studies are to give accurate guidance as to Internet-wide behavior of the protocols and algorithms being studied, **the chosen topologies must exhibit fundamental properties or invariants empirically found in the actual extant structure of the Internet.** Otherwise, correct conclusions cannot be drawn.

BRITE topology generator

- BRITE: Boston university Representative Internet Topology gEnerator
- http://www.cs.bu.edu/brite/

- Router topology model
- AS topology model
- Hierarchical
  - Top-down topology model
  - Bottom-up topology model

Router Waxman

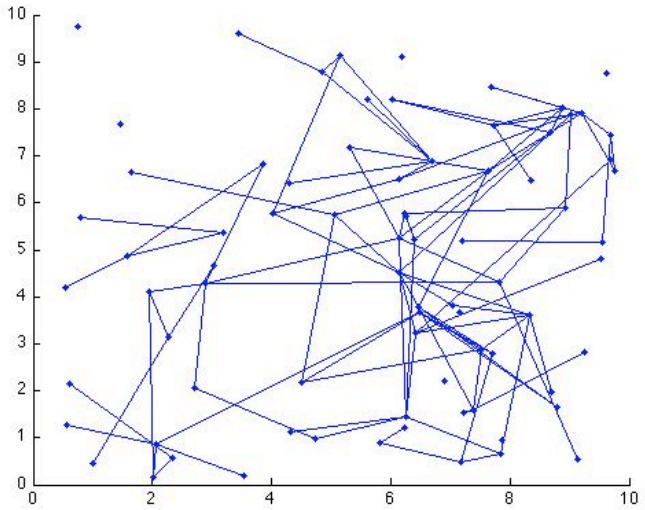
- Basically refers to a generation model for a random topology using Waxman's probability model for interconnecting the nodes of the topology, which is given by:

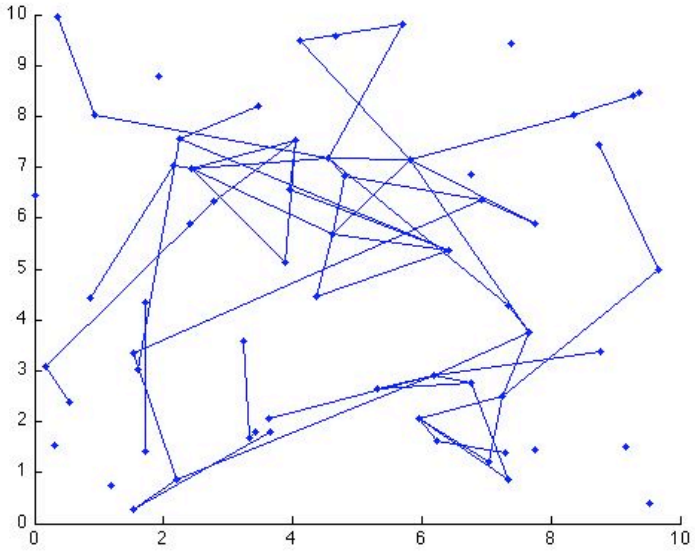
$$P(u,v) = a*e^{-d/(b*L)}$$

where $0 < a,b <= 1$,

d is the Euclidean distance from node ***u*** to node ***v***, and

L is the maximum distance between any two nodes.





Emiliano Casa
emiliano.casalicchio@uniroma2.it

Route Barabasi-Albert topology

- RouterBarabasiAlbert model that implements a model proposed by Barabási and Albert
- This model suggests two possible causes for the emergence of a power law in the frequency of outdegrees in network topologies: incremental growth and preferential connectivity.
  - **Incremental growth** refers to growing networks that are formed by the continual addition of new nodes, and thus the gradual increase in the size of the network.
  - **Preferential connectivity** refers to the tendency of a new node to connect to existing nodes that are highly connected or popular.
- RouterBarabasiAlbert interconnects the nodes according to **the incremental growth approach**. When a node ***i*** joins the network, the probability that it connects to a node ***j*** already belonging to the network is given by:

$$P(i,j) = d_j / S$$

where d_j is the degree of the target node and S is the sum of outdegrees of all nodes that previously joined the network.

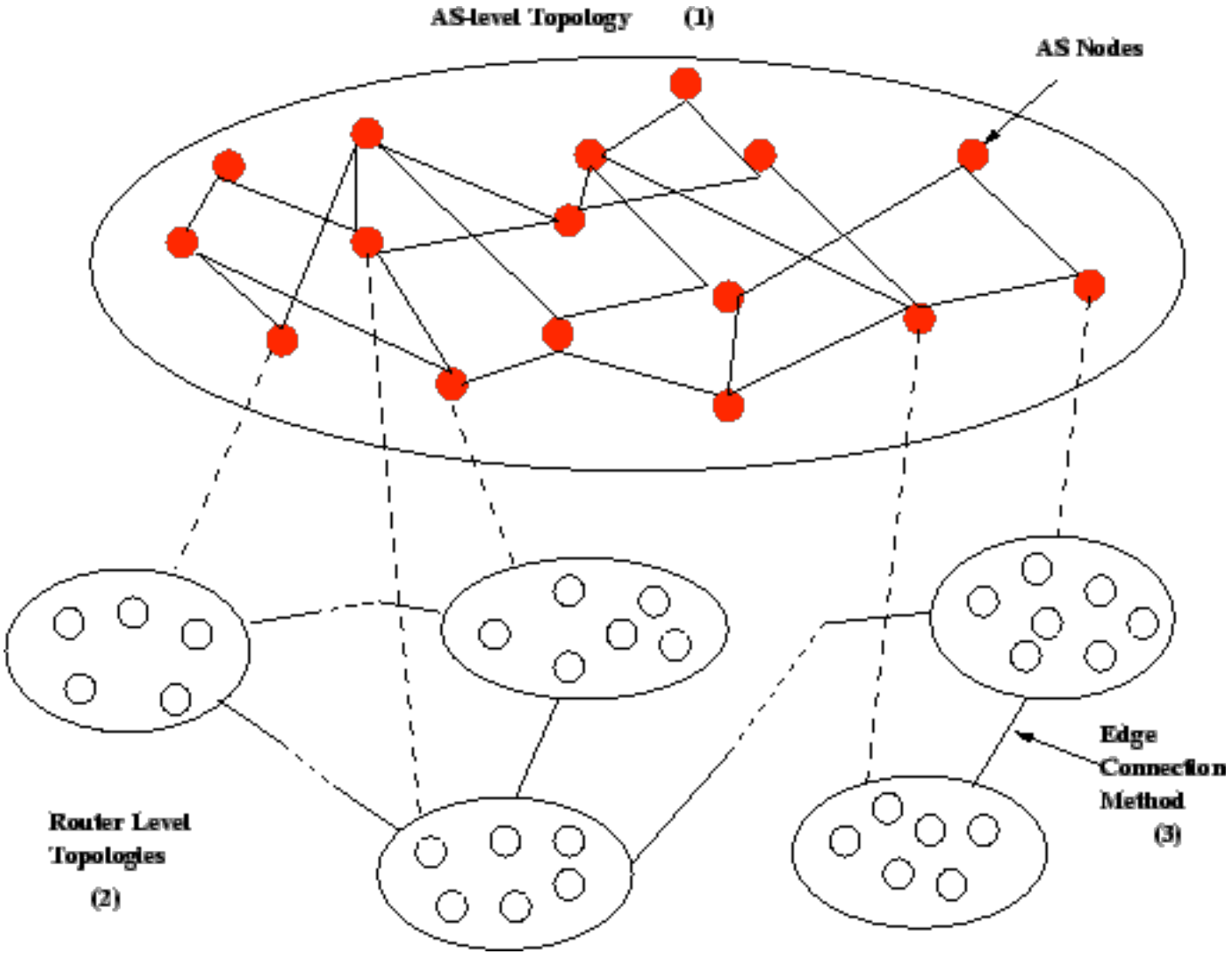
Emiliano Casalicchio -
emiliano.casalicchio@uniroma2.it

AS-level topology

- The provided AS-level models are very similar to the models provided for generating router-level topologies.
- The main difference between these router-level and AS-level models is the fact that AS models place AS nodes in the plane and these have the capability of containing associated topologies.
- Note that this does not mean that there are no AS-level and router-level models that differ substantially from each other. The idea of separating router-level from AS-level from the beginning is to allow for the flexibility of developing independent models for each scenario.
- The two AS-level models provided with the initial distribution of BRITE are **Waxman** and **BarabasiAlbert**

Top-down topology

- Top-down means that BRITE generates first an AS-level topology (1) according to one of the available flat AS-level models (e.g. Waxman, Imported File, etc.).
- Next, for each node in the AS-level topology BRITE will generate a router-level topology (2) using a different generation model from the available flat models that can be used at the router-level.
- (3) BRITE uses an edge connection mechanism to interconnect router-level topologies as dictated by the connectivity of the AS-level topology.
  - Performing this interconnection of router-level topologies in a representative way is an open research question.



Emiliano Casalicchio -
emiliano.casalicchio@uniroma2.it