Lezione n.9
SIMULAZIONE DI RETI P2P: PEERSIM
27/3/2009
Tutorials sul sito
http://peersim.sourceforge.net/

SIMULAZIONE DI SISTEMI P2P

- Un sistema P2P è, in generale, composto da milioni di peers
- La valutazione di un nuovo protocollo in un ambiente reale è in generale di difficile realizzazione
  - necessità di disporre di un alto numero di hosts
  - problemi: presenza di NATs, firewalls che ostacolano la comunicazione
  - definizione di ambienti che consentano la gestione di un programma distribuito su larga scala
  - dinamicità
- Ambienti attualmente disponibili per la valutazione di sistemi P2P
  - Planet Lab
  - GRID 5000
- Alternativa: utilizzare simulatori altamente scalabili

PEERSIM: INTRODUZIONE

- Simulatore open source, basato su JAVA ,sviluppato all'Università di Bologna

  http://peersim.sourceforge.net/

- Caratteristiche principali:
  - Scalabilità (maggiore con modalità cycle-based...)
  - Configurabilità
  - Plug in Simulation. La simulazione viene definita specificando un insieme di componenti che possono essere 'inserite' nel simulatore in modo da sfruttare le classi 'core' della simulazione
  - Simulazione eseguita su un unico host. Il simulatore non è multithreaded, non si possono sfruttare architetture multi-core
- Documentazione
  - JavaDOC ed un insieme di tutorial on line (presenti sul sito)
  - Mailing lists su sourceforge
  - Articoli con proposte di nuovi overlay valutati mediante Peersim

PEERSIM: INTRODUZIONE

- Struttura del simulatore
  - un nucleo minimo che realizza le funzionalità di base della simulazione
  - l'utente può definire nuove componenti
  - l'implementazione di una componente può essere facilmente rimpiazzata con un'implementazione diversa
  - i riferimenti alle componenti definite dall'utente sono raccolti in un file F di configurazione
  - possibilità di caricare dinamicamente nel simulatore le componenti definite in F

- Modalità di simulazione
  - cycle-driven
  - event-driven

PEERSIM: MODALITA' DI SIMULAZIONE

Caratteristiche della simulazione Cycle-Driven

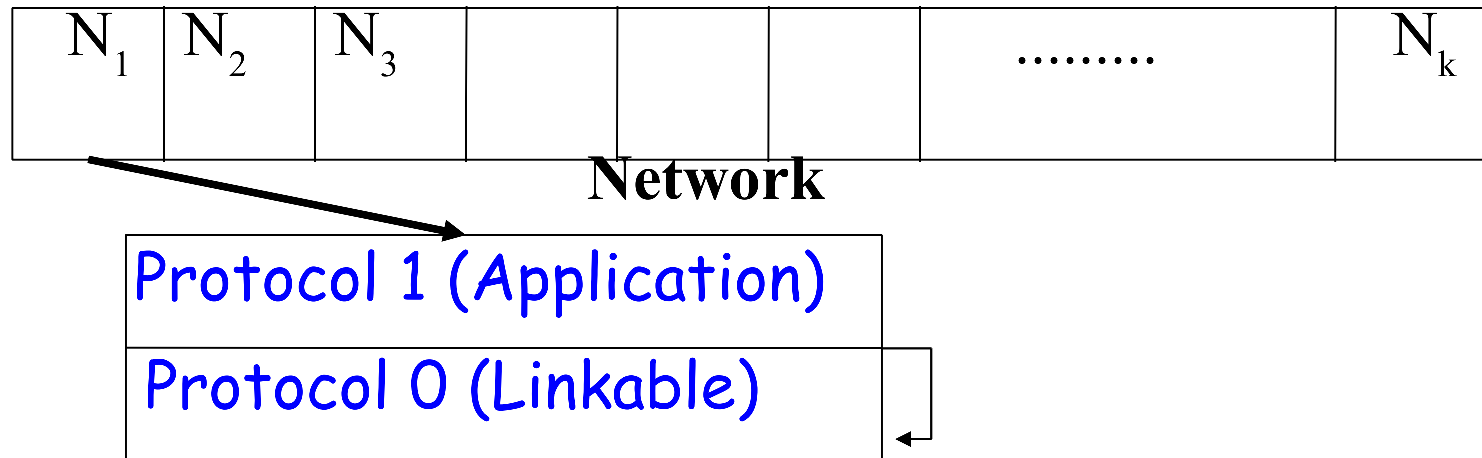
- modella la rete P2P come un insieme di nodi
- ogni nodo è un 'contenitore' di protocolli
- la simulazione consiste nell' esecuzione sequenziale di tutti i protocolli di ogni nodo
- scambio di messaggi tra nodi simulato mediante l'esecuzione di metodi su oggetti che modellano i protocolli
- non modella il livello trasporto dell'overlay (latenze, perdite di messaggi,....)
- utilizzato per analizzare proprietà dell'overlay indipendenti dal livello trasporto
  - crescita del numero di vicini di un nodo all'aumentare del numero di nodi
  - convergenza di algoritmi di gossiping
- testato fino a 10^7 nodi

PEERSIM: MODALITA' DI SIMULAZIONE

Caratteristiche della simulazione Event-Driven

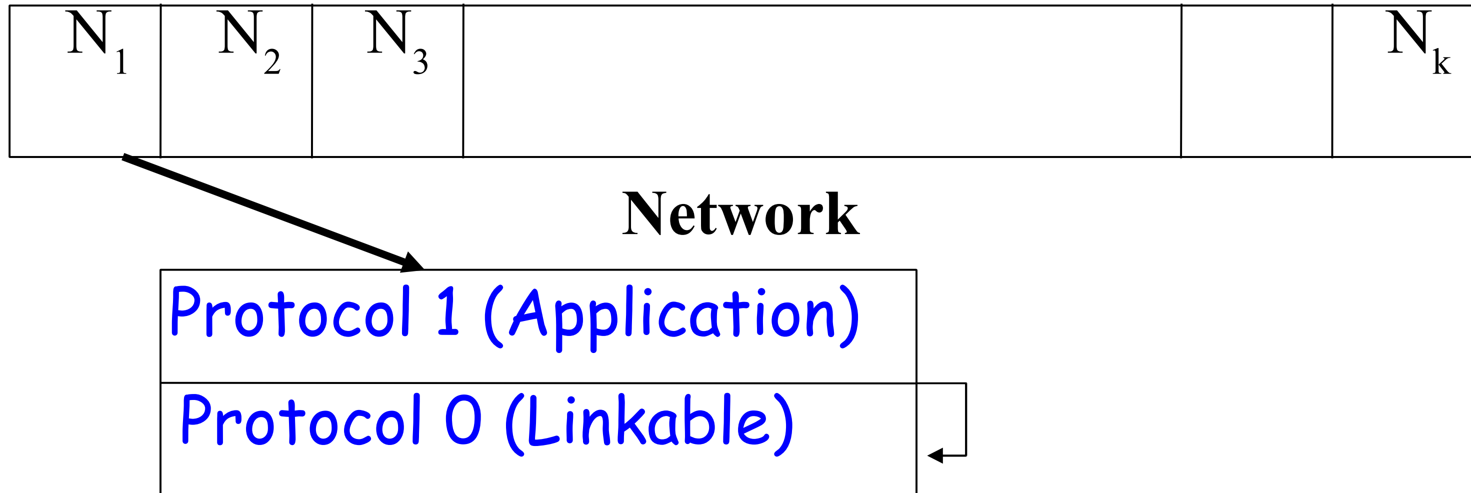
- possibilità di modellare lo scambio di messaggi tra nodi
- eventi= schedulazione dei messaggi ricevuti dai nodi
- l'esecuzione dei protocolli presenti su ogni nodo è guidata dagli eventi
- modellazione del sistema più realistica
- può essere utilizzata congiuntamente con il simulatore cycle-driven
- testato fino a $2.5 \cdot 10^5$ nodi

PEERSIM: L'ARCHITETTURA



- Network = Vettore contenente riferimenti a tutti i nodi della rete. Viene creata clonando un nodo prototipo per k volte (k dimensione della rete)
- Nodo
  - identificato da un PID
  - contiene uno stack di k protocolli, inizialmente assumeremo k=2
  - Peersim fornisce un'implementazione standard delle funzionalità di un nodo

PEERSIM: L'ARCHITETTURA



- Protocolli: specificano il comportamento del nodo, tutti i nodi eseguono gli stessi protocolli. Peersim offre un insieme di protocolli predefiniti
- una semplice implementazione può includere, ad esempio:
  - il protocollo Application definito dall'utente (es:simula il routing)
  - il protocollo Linkable = Contenitore di Nodi, non eseguibile, rappresenta la visione della rete posseduta da un nodo (contatti con i nodi vicini)

PEERSIM.CORE.NETWORK

```
public class Network {
    /** Numero di nodi appartenenti alla rete
    public static int size ( ) { return len;   }
    /** Restituisce un nodo dato indicizzato da index
    public static Node get (int index) {....}
    /**Aggiunge un nodo  dalla rete
    public static void add (Node n) {....}
    /** Rimuove un nodo dalla rete
    public static void remove( ) {....}
    ...........
```

- PeerSim implementa la classe Network (vedi documentazione)

PEERSIM.CORE.NODE

```
public interface Node extends Cloneable {
    /** restituisce l'i-simo protocollo del nodo
    public Protocol getProtocol (int i)
    /** restitisce il numero di protocolli inclusi nel nodo
    public int protocolSize( );
    /** restituisce l'identificatore unico del nodo
    public long getID( );
    /** per definire più istanze del nodo mediante clonazione
    public Object clone();
```

- PeerSim definisce la classe GeneralNode.java che implementa l'interfaccia Node. E' possibile utilizzare quella classe oppure definire una propria implementazione dell'interfaccia Node

PEERSIM.CORE.NODE

- Il metodo clone( ) viene invocato automaticamente da Peersim quando alloca i nodi che partecipano alla simulazione
- Supponiamo che ogni nodo utilizzi alcune strutture dati
- Se si vuole evitare che tutti i nodi facciano riferimento alla stessa struttura dati, occorre effettuare un overriding del metodo clone() in modo da allocare una copia diversa delle strutture dati per ogni nodo
- In questo modo una copia delle strutture dati del nodo viene allocata per ogni nodo automaticamente quando Peersim invoca ilmetodo clone() per quel nodo

PEERSIM.CORE.PROTOCOL

```
public interface Protocol extends Cloneable
    {/** Restituisce un clone del protocollo. Utilizzato per
      replicare il protocollo su diversi nodi
      public Object clone( );
      }
```

- L'interfaccia è molto generale. Vengono poi definite interfacce che estendono Protocol e contengono segnature più specifiche di metodi
- Come per I nodi, il metodo clone() deve essere riscritto dall'utente in modo da allocare copie diverse delle strutture dati accedute dal protocollo
- Una copia diversa per ogni istanza del protocollo in esecuzione su un nodo diverso

CYCLE DRIVEN PROTOCOLS:DEFINIZIONE

/* questa classe definisce il corpo dei protocolli che definiscono una simulazione cycle-driven, cioè di quei protocolli che vengono invocati ad intervalli di tempo regolari

```
public interface CDProtocol extends Protocol{
```

/* metodo che definisce il corpo del protocollo. Il metodo viene invocato ad ogni ciclo della simulazione per ogni nodo della rete e per ogni protocollo definito

```
public void nextCycle (Node node, int pid)

    {....}   }
```

NOTA BENE: Per definire una simulazione cycle-driven,l'utente deve

- implementare l'interfaccia CDProtocol
- specificare il corpo del metodo nextCycle( )

PEERSIM: L'ARCHITETTURA

- Controlli
  - definiscono operazioni che richiedono la conoscenza dell'intera rete
  - vengono eseguiti all'inizio della simulazione o periodicamente
  - esistono alcuni controlli predefiniti, mail loro comportamento può essere specificato dall'utente
- Un controllo può essere di uno dei seguenti tipi
  - Initializers eseguiti all'inizio della simulazione per definire
    - la topologia iniziale dell'overlay
    - lo stato iniziale dei nodi
  - Dynamics eseguiti periodicamente durante la simulazione per
    - aggiungere nodi all'overlay
    - rimuovere nodi dall'overlay
    - ....
  - Observers eseguiti periodicamente durante la simulazione
    - analisi statistica
    - ...

PEERSIM.CORE.CONTROL

/*interfaccia generica per le classi responsabili di monitorare la simulazione oppure di modificare lo stato della rete

```
public interface Control
```

/* il metodo execute restituisce **true** se la simulazione può essere terminata, **false** altrimenti

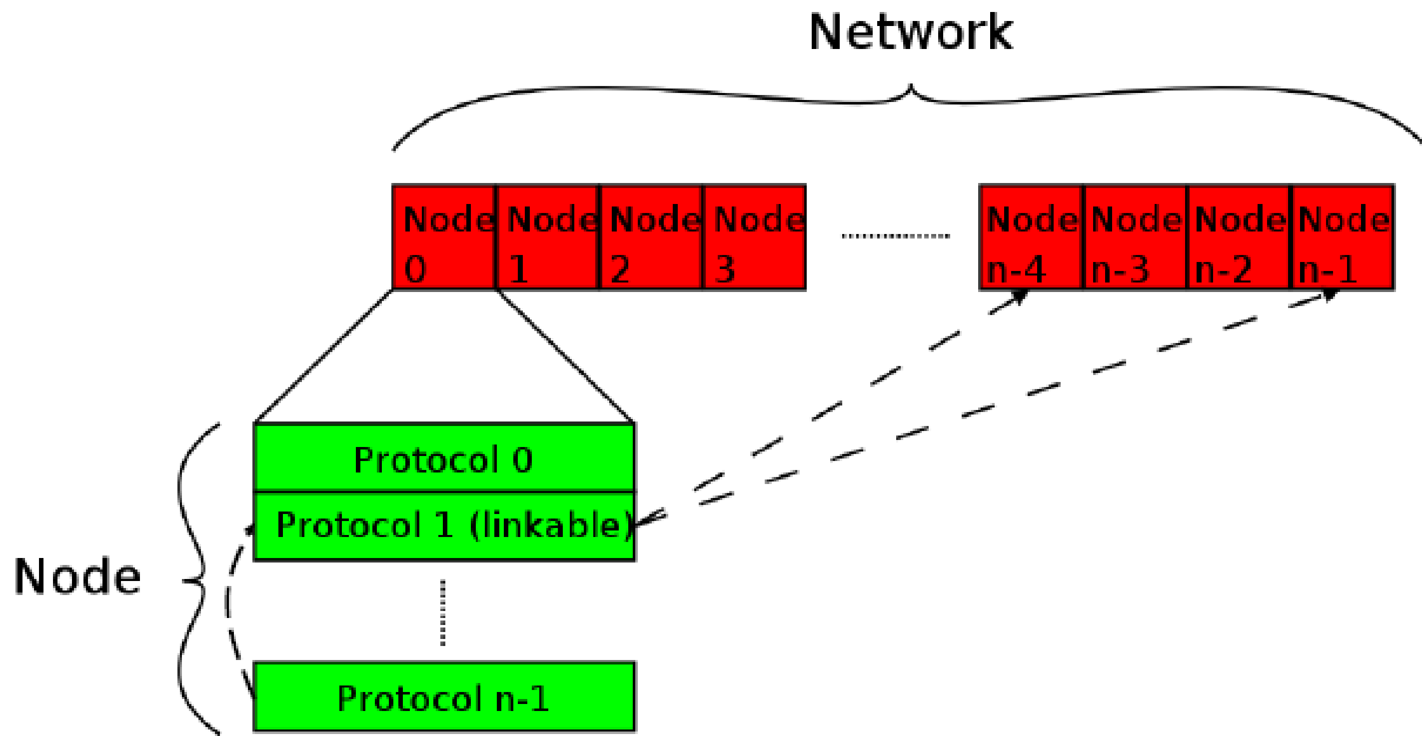
```
public boolen execute():
}
```

PEERSIM.CORE.LINKABLE

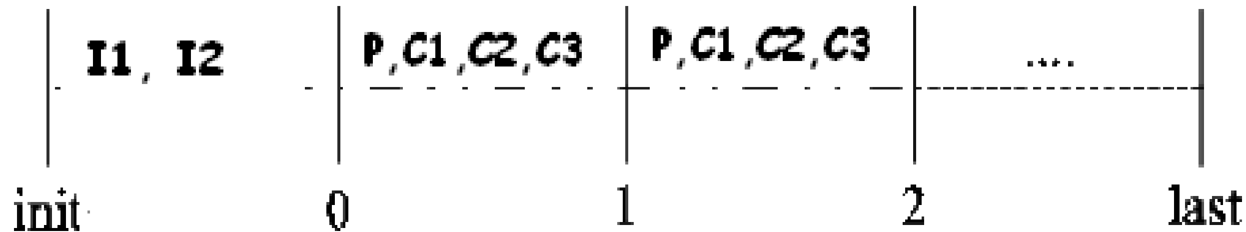
- Ogni nodo possiede una visione locale dell'overlay
- Linkable = Contenitore di Nodi, rappresenta la conoscenza locale di un nodo rappresenta i links di un nodo con i nodi vicini

```
public interface Linkable extends Cleanable {
    /* restituisce la dimensione della lista dei vicini
    public int degree( );
    /* restituisce il vicino caratterizzato da un certo indice
    public Node getNeighbour (int i);
    /* aggiunge un nodo alla lista dei vicini
    public boolean addNeighbour (Node neighbour);
    /*Restituisce vero se un nodo fa parte della lista dei vicini
    public boolean contains (Node neighbour);
```

PEERSIM.CORE.LINKABLE



PEERSIM: CYCLE-BASED SCHEDULING



Cycle-Based Simulation : supponiamo che sia stato definito

- due inizializzatori I1, I2
- un protocollo P
- tre diversi controlli periodici C1, C2, C3

- Dopo la fase di inizializzazione, il simulatore
  - esegue n cicli di simulazione (n specificato dall'utente)
  - ad ogni ciclo vengono eseguiti tutti i protocolli (nel nostro caso solo P) e quindi tutti i controlli periodici (nel nostro caso C1,C2, C3) su tutti i nodi della rete
  - E' possibile configurare il simulatore in modo da stabilire l'ordine con cui vengono eseguiti i protocolli/controlli

PEERSIM: IL FILE DI CONFIGURAZIONE

File di testo in cui l'utente specifica

- Parametri Globali: la dimensione della rete, il numero di cicli della simulazione,....
- Protocolli e Componenti che devono essere inseriti dinamicamente nel simulatore. Per ogni componente:
  - riferimento alla classe definita dall'utente o predefinita che implementa la componente
  - parametri per la configurazione della componente

PEERSIM: UN ESEMPIO

Esempio: Simulazione di un algoritmo distribuito di aggregazione basato su gossiping tra nodi

- Dimensione della rete: 50000 nodi
- Stato iniziale dei nodi: valore intero nell'intevallo (0-100)
- Protocollo:
  - Calcolo distribuito di una funzione (media) calcolata sull'insieme di valori memorizzati nei nodi della rete
  - Gossip Based Aggregation: ogni nodo seleziona periodicamente un vicino e scambia con esso l'approssimazione del valore calcolato. Entrambe i nodi aggiornano l'approssimazione corrente
- Topologia: Connessione casuale tra i nodi

PEERSIM: PROPRIETA' GLOBALI

```
01 # PEERSIM GOSSIP AGGREGATION
02
03 random.seed 1234567890
04 simulation cycles 30
05 control.shf Shuffle
06 network.size 50000
```

Proprietà globali

- simulation cycles cicli di simulazione
- network size dimensione della rete
- shuffle cambia l'ordine con cui i nodi vengono considerati ad ogni ciclo della simulazione
- random seed parametro utilizzato per replicare esattamente i risultati della simulazione basandosi su un comportamento pseudo-random

PEERSIM: I PROTOCOLLI

```
07   protocol.lnk IdleProtocol
08
09   protocol.avg example.aggregation.AverageFunction
10   protocol.avg.linkable lnk
```

Dichiarazione di Protocolli

`protocol.string_id [full_path]classname`

- assegna l'identificatore string_id al protocollo definito nella classe classname
- protocol.lnk IdleProtocol assegna il nome lnk al protocollo IdleProtocol
- IdleProtocol
  - protocollo predefinito da PeerSIM,implementa la interfaccia Linkable
  - È un contenitore di links
  - non è eseguibile (non implemanta CDProtocol)

PEERSIM: I PROTOCOLLI

```
07 protocol.lnk IdleProtocol
08
09 protocol.avg example.aggregation.AverageFunction
10 protocol.avg.linkable lnk
```

Dichiarazione di protocolli:

```
protocol.string_id [full_path]classname
```

- assegna l'identificatore string_id al protocollo definito nella classe classname
- example.aggregation.AverageFunction
  - protocollo definito dall'utente
  - implementa un protocollo di aggregazione basato su gossiping
  - Viene eseguito ad ogni ciclo

PEERSIM: I PROTOCOLLI

```
07 protocol.lnk IdleProtocol
08
09 protocol.avg example.aggregation.AverageFunction
10 protocol.avg.linkable lnk
```

Configurazione dei Protocolli:

- protocol.string_id.xxx definisce il parametro xxx del protocollo identificato da string
- protocol.avg.linkable lnk configura il protocollo avg in modo che utilizzi protocollo lnk, cioè l'IdleProtocol, per rappresentare la visione locale che ogni nodo possiede della rete
- il protocolo avg utilizza come overlay la topologia rappresentata in IdleProtocol

PEERSIM: INIZIALIZZATORI

```
11 init.rnd WireKOut
12 init.rnd.protocol lnk
13 init.rnd.k 20
```

Definizione delle componenti che devono essere eseguite solo una volta per inizializzare la simulazione

- init.rnd WireKOut associa al protocollo WireKOut (predefinito) l'identificatore rnd
- WireKOut inizializza l'overlay mediante connessione casuale dei nodi ed utilizza come protocollo contenitore il protocollo lnk, cioè l'IdleProtocol.
- init.rnd.k Il parametro k definisce il grado massimo dei nodi dell'overlay. E' un parametro utilizzato da WireKOut

PEERSIM: ACQUISZIONE DEI PARAMETRI

All'interno della classe WireKOut occorre acquisire i parametri

```
private final int k;
......
private static final String PAR_DEGREE = "k";
.....
k = Configuration.getInt(prefix + "." + PAR_DEGREE);
.....
public WireKOut (String prefix)
    { super(prefix);
    k = Configuration.getInt(prefix + "." + PAR_DEGREE);    }
```

- Prefix : identificatore associato a WireKOut nel file di configurazione
- Le classi Configuration e FastConfiguration contengono i metodi per acquisire i parametri dal file di configurazione(vedere documentazione)

PEERSIM: I CONTROLLI

```
18 init.lin LinearDistribution
19 init.lin.protocol avg
20 init.lin.max 100
21 init.lin.min 1
```

Definizione delle componenti che devono essere eseguite solo una volta per inizializzare la simulazione

- LinerDistribution Libreria che definisce una distribuzione lineare crescente.I valori considerati sono compresi tra 1 e 100 (parametri init.lin.min 1, init.lin.max 100)

- init.lin.protocol avg definisce il protocollo che utilizza la distribuzione lineare dei valori

PEERSIM:I CONTROLLi

```
22 control.avgo example.aggregation.AverageObserver
23 control.avgo.protocol avg
```

- Definizione delle componenti che devono essere schedulate periodicamente
- Le componenti seguono il monitoraggio dell'overlay
- AverageObserver: Componente predefinito che produce statistiche sui valori osservati sui nodi dell'overlay (vedere documentazione)

PEERSIM: FILE DI CONFIGURAZIONE

- E' possibile stabilire nel file di configurazione l'ordine con cui vengono eseguiti i protocolli
- se non esiste alcuna specifica, i protocolli vengono eseguiti in ordine alfabetico
- altrimenti:
  - con la clausola : order.protocol P1 P2 P3

    i protocolli vengono eseguiti in questo ordine P1 P2 e P3. I protocolli non specificati vengono eseguiti in ordine alfabetico
  - con la clusola: include.protocol P1 P2 P3

    eseguo P1 P2 e P3 in questo ordine, ma gli altri protocolli non vengono eseguiti
  - con la clausola: control.shf Shuffle

    posso cambiare ad ogni ciclo di simulazione l'ordine con cui il simulatore sceglie i nodi per l'esecuzione dei protocolli
- i controlli vengono eseguiti comunque dopo i protocolli

PEERSIM: ESEGUIRE IL SIMULATORE

JAVA -cp <class-path> peersim.Simulator example1.txt

example1.txt è il nome del file di configurazione

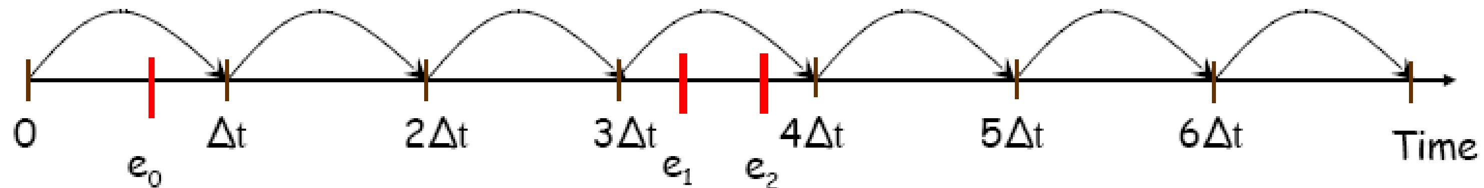
DISCRETE EVENT SIMULATION (DES)

- Sistema: insieme di entità interagenti che cooperano per fornire un insieme di funzionalità
  - esempio:
    - determinare quante classe veloci sono necessarie in un supermercato per assicurare un servizio 'veloce' a clienti con meno di 10 oggetti
    - entità: casse, clienti con meno di 10 oggetti
- Stato del Sistema :
  - insieme di variabili il cui valore caratterizza il sistema in un certo istante di tempo
  - esempio: numero di casse veloci, tempo di arrivo dei clienti con meno di 10 oggetti, numero di clienti in coda,...
- Evento: occorrenza istantanea di una qualsisi azione che modifica lo stato del sistema
  - esempio: arrivo di un nuovo cliente, inizio del servizio da parte di una cassa, fine del servizio

DISCRETE EVENT SIMULATION

- Simulation clock: variabile che registra lo scorrere del tempo nella simulazione
  - E' necessario stabilire una relazione tra il tempo simulato ed il tempo reale (es: 1 tick= 200 ms)
  - in genere non esiste una relazione tra il tempo simulato ed il tempo necessario per l'esecuzione della simulazione
- Due approcci per l'avanzamento del tempo simulato:
  - ad incrementi costanti
  - Basato sull'occorrenza di eventi (discret event simulation)

AVANZAMENTO AD INCREMENTI COSTANTI

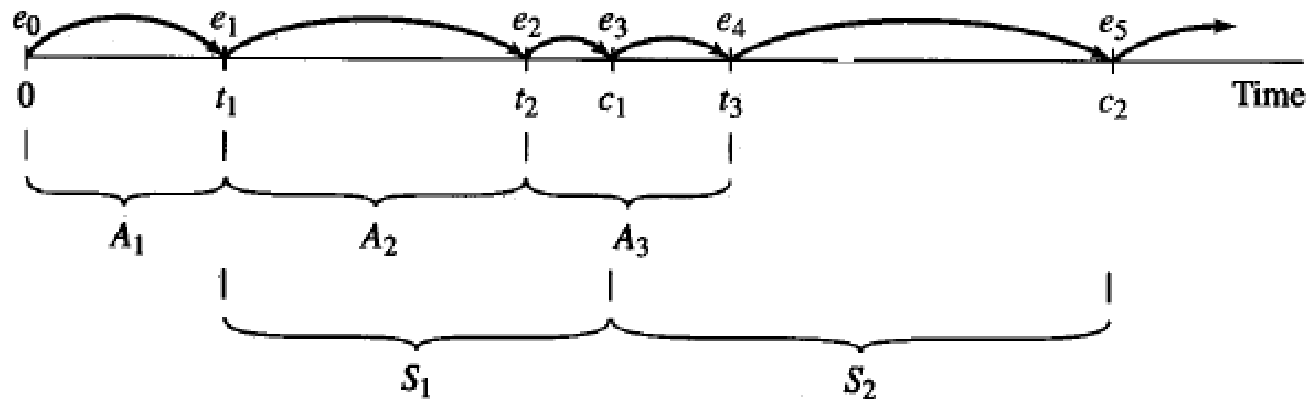


- L'avanzamento del tempo di simulazione avviene mediante incrementi costanti (time step)
- L'occorrenza di un qualsiasi evento viene fatta coincidere con l'inizio di un time step
- Gli eventi che avvengono durante un timestep vengono spostati e fatti coincidere con l'inizio del time step successivo
- Simulazione semplice da implementare,mapoco accurata

DISCRETE EVENT SIMULATION

- il tempo simulato viene inizializzato a 0
- si individano gli eventi che possono modificare lo stato del sistema
- si associa ad ogni evento il suo tempo di occorrenza (timestamp)
- il simulatore mantiene una lista degli eventi ordinata per valori crescenti di event time step
- L'avanzamento del tempo viene guidato dai timestamps associati agli eventi
- Prossimo evento= Evento con timestamp minimo
- ad ogni passo di simulazione il tempo assume il valore del timestamp del prossimo evento
- il tempo 'salta' da un timestamp all'altro e 'non esiste' nell'intervallo tra due eventi successivi.
- i periodi in cui non avviene nessun evento rilevante per la simulazione vengono ignorati

DISCRETE EVENT SIMULATION



Esempio: Casse veloci e clienti

- t_i = tempo di arrivo dell'i-esimo cliente (t_0=0)
- $A_i = t_i - t_{i-1}$ = tempo di interarrivo tra due clienti
- S_i = tempo di servizio dell'i-esimo cliente
- c_i = istante di tempo in cui il cliente lascia il supermercato perchè ha ricevuto il servizio

EVENT DRIVEN PEERSIM

- Peersim definisce un insieme di interfacce/classi per il supporto della simulazione basata su eventi
- Peersim discretizza il tempo in istanti virtuali o tick
- Tick = unità di misura del tempo
- In base ai tick si misura
  - la durata dell'intera simulazione
  - il tempo a cui deve avvenire qualsiasi evento
- Tipi di eventi in Peersim
  - Eventi periodici
    - spedizione del proprio stato ad un vicino
  - Eventi asincroni
    - ricezione di un messaggio da un vicino

SCHEDULAZIONE DI EVENTI PERIODICI

Eventi periodici

- come nella simulazione cycle based, vengono schedulati periodicamente
- poichè non esiste il concetto di ciclo, occorre
  - specificare l'istante temporale (tick) in cui l'evento viene schedulato per la prima volta
  - l'intervallo di tempo (STEP) che intercorre tra due successive generazioni dell'evento.
- Esempio:
  - un protocollo P può essere schedulato periodicamente, anche se si usa la modalità event based
  - si specifica il tick temporale in cui avviene la prima esecuzione di P e l'intervallo che intercorre tra una esecuzione di P e la successiva
  - come nella simulazione cycle-based, il codice del protocollo va comunque specificato mediante l'implementazione del metodo nextcycle( ) dell'interfaccia CDProtocol

ESECUZIONE DI UN PROTOCOLLO PERIODICO

- Si può utilizzare uno schedulatore predefinito, il CDScheduler, per inserire nella coda degli eventi la prima esecuzione di un protocollo periodico
- CDSCheduler implementato in una classe predefinita di Peersim
- Al termine della esecuzione, il protocollo stesso si schedula per l'esecuzione successiva
- Il tick a cui avviene l'esecuzione successiva viene stabilito mediante lo step specificato nel file di configurazione
- Quando viene schedulato l'evento corrispondente al protocollo periodico, viene eseguito il corpo della nextcycle( ) di quel protocollo
- nextcycle(Node, ProtocolID) alla nextcycle vengono passati i riferimenti a
  - Il nodo della rete su cui si esegue il protocollo
  - L'identificatore attribuito al protocollo nel file di configurazione. Questo identificatore viene utilizzato per identificare i parametri del protocollo nel file di configurazione

EVENT DRIVEN PEERSIM

Per la definizione degli eventi periodici, occorre

- implementare l'interfaccia CDProtocol, definendo il corpo del metodo nextCycle, come nell'approccio cycle based
- fornire uno schedulatore di eventi periodici che
  - associ un time stamp alla prima occorrenza dell'evento
  - definisca la frequenza di schedulazione dell'evento
- Peersim fornisce uno schedulatore predefinito (CDScheduler)
- è possibile implementare schedulatori ad hoc

Per definire un event handler occorre

- implementare l'interfaccia EDProtocol, definendo il corpo del metodo processEvent.
- il metodo processEvent( ) viene invocato su un nodo, per un certo protocollo e contiene il codice da eseguire al momento della ricezione di evento (handler dell'evento)

FILE DI CONFIGURAZIONE

```
01 N 4560
02 M 20
03 simulation.endtime N
04 protocol.c1 AverageProtocol
05 protocol.c1.step M
06 init.schedulatore CDScheduler
07 init.schedulatore.protocol c1
```

- clausola 06: Si attribuisce il nome schedulatore allo schedulatore standard di Peersim, contenuto nella classe CDScheduler
- clausola 07: Si indica che lo schedulatore deve schedulare la prima esecuzione del protocollo il cui nome simbolico attribuito nel file di configurazione è c1
- clausola 05: si indica che quel protocollo deve essere schedulato ogni 20 tick() del tempo virtuale

EVENT BASED PEERSIM

Per generare un nuovo evento occorre:

- definire l'evento specificando
  - il delay dell'evento, cioè l'intervallo di tempo (rispetto al tempo attuale) che deve trascorrere prima che l'evento venga schedulato
  - l'oggetto che descrive l'evento
  - il nodo n a cui l'evento è destinato
  - il protocollo in esecuzione su n che riceverà l'evento

- inserire l'evento nella coda degli eventi gestita dal simulatore
  - metodo add della classe EDSimulator

- definire l'handler dell'evento

EVENT BASED PEERSIM

- L'invio/ricezione dei messaggi è modellato mediante eventi
- Ogni nodo può
  - spedire messaggi ad un altro nodo.
    - un messaggio è spedito ad un protocollo in esecuzione su un nodo
    - ad ogni messaggio viene associato un timestamp che indica l'istante temporale in cui tale messaggio dovrà essere ricevuto
    - il timestamp può essere utilizzato per modellare le latenze della rete
  - ricevere messaggi.
- I protocolli che ricevono messaggi devono implementare un handler che deve essere eseguito al momento della ricezione del messaggio
  - Handler= implementazione del medodo processevent( ) della classe EDSimulator

IL LIVELLO DI TRASPORTO

- La spedizione del messaggio viene gestita da protocolli associati ai nodi che simulano il livello di trasporto della rete
- Peersim offre diversi livelli di trasporto implementati da protocolli predefiniti
  - UniformRandomTransport. Servizio di trasporto affidabile, associa latenze variabili alla trasmissione dei messaggi
  - UnreliableTrasport. Latenze variabili + Perdita di messaggi con una data probabilità
  - altri protocolli.....
  - inoltre è possibile definire nuovi servizi di trasporto implementando l'interfaccia Transport

IL LIVELLO DI TRASPORTO

Esempio:

il protocollo Example esegue delle send e della receive e quindi utilizza un certo livello di trasporto

```
protocol.urt UniformRandomTransport
protocol.urt.mindelay MIND
protocol.urt.maxdelay MAXD
protocol.transp UnreliableTransport
protocol.transp.transport urt
protocol.transp.drop DROP
protocol example Example
protocol.example transp
```

IL LIVELLO DI TRASPORTO

Esempio:

il protocollo Example esegue delle send e della receive e quindi utilizza un certo livello di trasporto

```
protocol.urt UniformRandomTransport
protocol.urt.mindelay MIND
protocol.urt.maxdelay MAXD
```

con queste clausole si definisce

- il protocollo urt che è implementato mediante la classe predefinita da Peersim UniformRandomTransport
- UniformRandomTransport introduce latenze variabili nell'invio dei messaggi
- Occorre specificare i parametri MIND a MAXD che indiacno la latenza minima e massima

IL LIVELLO DI TRASPORTO

Esempio:

il protocollo Example esegue delle send e della receive e quindi utilizza un certo livello di trasporto

```
protocol.transp UnreliableTransport
protocol.transp.transport urt
protocol.transp.drop DROP
```

con queste clausole si definisce

- Il protocollo UnreliableTransport che implementa un livello di trasporto non affidabile
- UnreliableTransport introduce perdita di messaggi
- Occorre specificare i paramentri urt, drop. Il primo specifica che il protocollo UnrelaibleTrasport è costruito sul protocollo urt, il secondo indica la probabilita' di perdita di messaggi

IL LIVELLO DI TRASPORTO

Esempio:

il protocollo Example esegue delle send e della receive e quindi utilizza un certo livello di trasporto

```
protocol.example Example
protocol.example.transport transp
```

- con la seconda clausola si specifica che il protocollo example, implementato nella classe Example, utilizza il protocollo transp, come protocollo per il livello di trasporto

- Il protocollo specificato (transp) verrà utilizzato ogni volta che si effettua una send() (modella latenza e perdita di messaggi)

EVENT DRIVEN PEERSIM

Spedizione di messaggi

```
Send( M, D, Mes, Prot)
```

- M: Nodo Mittente
- D: Nodo Destinatario
- Mes: Messaggio
- Prot: Il messaggio viene consegnato al protocollo P in esecuzione sul nodo D

Ricezione di messaggi

- basata sulla definizione di message handler
- il protocollo che intende ricevere il messaggio deve definire un handler che si incarica di ricevere il messaggio e di elaborarlo
- simile agli event handler di JAVA (gestione di eventi generati dall'interfaccia,...)

GOSSIP BASED AVERAGE

Calcolo della media basato su tecniche di gossiping

- un valore intero V per ogni nodo della rete
- V inizializzato in modo casuale
- algoritmo eseguto da ogni nodo
  - scelta casuale di un vicino NB e scambio dei valori
  - aggiornamento del valore V con la media tra V ed il valore ricevuto dal vicino

La simulazione in Peersim richiede:

- la definizione di un protocollo schedulato periodicamente che contenga un metodo che implementi la scelta casuale di NB e l'invio di V ad NB
- la definizione di un handler che implementi:
  - la ricezione dei messaggi dai nodi vicini
  - l'eventuale invio del proprio valore al nodo vicino
  - aggiornamento della media.

EVENT DRIVEN SIMULATION

- Definizione del messaggio scambiato tra i nodi

```
class AverageMessage {
    final double value;
    public AverageMessage (double value, Node sender)
        {
        this.value = value;
        this.sender = sender
        }}
```

- Definizione di una classe che implementi sia CDProtocol che EDProtocol
  - Schedulazione periodica della send
    - Definizione della nextcycle( )
  - Definizione dell'handler associato alla ricezione dell'evento
    - Definizione della processevent( )

GOSSIP BASED AVERAGE: EVENTI PERIODICI

```
public Class AverageED implements CDProtocol, EDProtocol

    public AverageED {....}

    public void nextCycle (Node node, int pid)
        { Linkable linkable = (Linkable) node.getProtocol
                                  (FastConfig.getLinkable(pid));
        if (linkable.degree( ) > 0
           { Node peern = <scelta casuale di un nodo vicino>
           Transport t = (Transport)
                  node.getProtocol(FastConfig.getTransport(pid))
           t.send(node, peern, new AverageMessage(value, node),pid)
        }
```

GOSSIP BASED AVERAGE: MESSAGE HANDLER

```
public void processEvent (Node node, int pid, Object event)

   {  AverageMessage aem = (AverageMessage) event;
   // Questo metodo viene schedulato quando viene ricevuto un
     messaggio
   // sul nodo Node, diretto al protocollo pid
   // l'istruzione precedente equivale ad una  receive

   if (aem.sender != null)
   Transport t = ((Transport)
     node.getProtocol(FastConfig.getTransport(pid))
   t.send (node, aem.sender, newAverageMeassage(value,null),pid);
   value = (value + aem.value) / 2
   }
```

GOSSIP BASED AVERAGE: MESSAGE HANDLER

- il metodo processEvent gestisce i messaggi ricevuti da un protocollo associato ad un nodo
- Mittente = null utilizzato per implementare lo scambio di valori
  - se mittante= null, non si invia una risposta, perchè il messaggio ricevuto è già un messaggio di risposta
  - la risposta viene inviata accedendo al servizio di trasporto
- ad ogni protocollo che utilizza l'invio di messaggi deve essere associato un protocollo di trasporto
- FastConfig.getTransport (pid)) restituisce un riferimento al protocollo di trasporto associato al protocollo pid
- L'associazione viene specificata nel file di configurazione

GOSSIP BASED AVERAGE: IL FILE DI CONFIGURAZIONE

Il file di configurazione dece contenere

- parametri globali: dimensione della rete, fine della simulazione,....
- definizione del protocollo che definisce l'overlay
- definizione dei protocolli di trasporto associati ai protocolli che utilizzano scambio di messaggi
- politica utilizzata per la schedulazione degli eventi periodici
- ....

CONFIGURAZIONE DELLA SIMULAZIONE

```
01 # PEERSIM EVENT DRIVEN AGGREGATION
02 SIZE 10^3
03 network.size SIZE
04 random.seed 1234567890
05 simulation.endtime 10^6
06 network.node peersim.core.GeneralNode
```

Proprietà globali

- network.size dimensione della rete
- network.node classe che implementa l'oggetto nodo

La simulazione termina quando

- la coda degli eventi è vuota
- tutti gli eventi nella coda sono caratterizzati da un

time-stamp > simulation.endtime

CONFIGURAZIONE DELLA SIMULAZIONE

```
07  #protocols
08 CYCLE 1000
09 protocol.avg AverageED
10 protocol.avg.linkable link
11 protocol.avg.step CYCLE
12 protocol.avg.transport tr
```

Proprietà dei protocolli

- al protocollo implementato nella classe AverageED viene assegnato il nome avg
- il protocollo utilizzato da avg per la gestione dell'overlay è link
- il protocollo utilizzato da avg per l'invio dei messaggi è tr
- il protocollo avg deve essere eseguito con frequenza CYCLE
- il protocollo tr deve essere opportunamente configurato (vedi pagina successiva)

CONFIGURAZIONE DELLA SIMULAZIONE

```
13 MINDELAY 10
14 MAXDELAY 400
15 DROP 20
16 protocol.urt UniformRandomTransport
17 protocol.urt.mindelay (CYCLE*MINDELAY)/100
18 protocol.urt.maxdelay (CYCLE*MAXDELAY)/100
19 protocol.tr UnreliableTransport
20 protocol.tr.transport urt
21 protocol.tr drop DROP
```

Configurazione dei protocolli di trasporto

- si definisce il protocollo urt che assegna ai messaggi un valore casuale della latenza variabile tra mindelay e maxdelay
- si definisce un protocollo wrapper di urt, il protocollo tr, che aggiunge alle funzionalità di urt lo scarto di messaggi con probabilità DROP

CONFIGURAZIONE DELLA SIMULAZIONE

```
21 init.sch CDScheduler
22 init.sch.protocol avg
23 init.sch.randstart
```

Definizione di uno schedulatore per gli eventi periodici

- Il protocollo avg implementa il metodo nextCycle e contiene quindi un evento che deve essere schedulato periodicamente
- Occorre definire un componente che effettui lo scheduling periodico del protocol
- Il componente scelto è uno schedulatore predefinito di Peersim: CDScheduler
- CDScheduler viene associato al protocollo avg
- La prima invocazione del protocollo nextCycle avviene in un istante di tempo casuale tra 0 e CYCLE
- Successivamente il metodo viene invocato ogni CYCLE time steps