UNIVERSITÀ DEGLI STUDI DI PAVIA
FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica
Sede di Mantova

Simulazione del comportamento prestazionale di una rete IP in presenza di traffico UDP di videostreaming

Relatore:
Prof. GIUSEPPE FEDERICO ROSSI

Tesi di Laurea di:
Marco Bianchera

Matricola:
292463 / 17

Anno Accademico 2004-2005

Indice

Qualche idea…

La seguente tesi esaminerà la simulazione ad eventi discreti testata tramite un simulatore, nel nostro caso useremo Network Simulator 2.

Inizialmente l'idea era quella di poter misurare l'affidabilità dei dati in output della simulazione confrontandoli con dati teorici, ma la mancanza di parametri per i livelli 1 e 2 (fisico e DLC nella scala ISO/OSI) non hanno consentito di eseguire misure "esatte" con i dati forniti dalla simulazione fatta su supporto cartaceo.
Si è scelto quindi di optare per una simulazione meno vincolata a questi "difetti" presenti nel simulatore, dedicando più attenzione ai due principali protocolli presenti nelle reti IP:
TCP e UDP.

Abbiamo pensato ad una simulazione orientata ad applicazioni di nuova generazione, come ad esempio il traffico di Streaming, cercando di sfruttare al meglio il protocollo messo a disposizione da NS2: RTP (in realtà sono due, dato che esiste anche FTP, ma per usarlo occorre implementare nuovi metodi scritti in C++).

RTP si può tranquillamente dichiarare sottoclasse del protocollo UDP, in quanto, nella gerarchia di NS2 (come nella realtà) viene rappresentato come "successore" del protocollo UDP.

- Ma durante le fasi di sperimentazione dell' RTP abbiamo notato una seconda anomalia presente in NS2: le persone che leggeranno questa tesi dovrebbero aver già chiare alcune idee sui protocolli di telecomunicazione e quindi conosceranno la fondamentale differenza tra TCP e UDP; il protocollo TCP fornisce un servizio di affidabilità della trasmissione dei dati, quindi, grazie all'uso di traffico di feedback per la ritrasmissione di PDU-Dati non andate a buon fine (vedi schemi ARQ) e il supporto della consegna in sequenza delle PDU-Dati riesce a garantire una connessione molto stabile e sicura; invece il protocollo UDP viene solitamente usato per traffici che non necessitano di molti controlli e che non prevedono l'uso di feedback (come ad esempio lo streaming).

E’ proprio su questa differenza di comportamento che nel nostro simulatore avviene una cosa strana, infatti, NS2 prevede la numerazione delle PDU anche per il protocollo UDP, non è da considerarsi una numerazione per la consegna in sequenza dei pacchetti come avviene in TCP, ma nella realtà UDP non numera mai le proprie PDU-dati.

Ora che abbiamo chiarito un pò le cose, possiamo entrare nella parte importante e tecnica della tesi.

Parleremo di NS2, dell’importanza della simulazione ad eventi discreti, dei principali comandi di NS2, della scala ISO/OSI, dei protocolli TCP e UDP, dell’RTP e per finire, parleremo della simulazione del traffico di videostreaming.

Il manuale e tutto il materiale riguardante NS è disponibile sul sito ufficiale:

www.isi.edu/nsnam

Introduzione

Network Simulator nasce come simulatori di reti basate su protocolli Internet o, come nel gergo informatico, si potrebbe definire come simulatore IP-Based (basato su IP) che trova le sue radici nello sviluppo del progetto REAL (altro simulatore) e nasce come variante open-source dello stesso.

NS venne usato per la prima volta all'università di Berkeley nel 1989 e dopo numerose modifiche e accorgimenti (data la sua natura open-source) il suo evolversi convinse anche numerose associazioni tra cui DARPA (collaborazione tra USC/ISI, Xerox PARC, LBNL e soprattutto l'università di Berkeley) che lo incluse nel progetto VINT (progetto tuttora TOP-SECRET Virtual InterNetwork testbed) e molte fondazioni come ACIRI, la UCB Daedalus e la famosissima SUN Microsystem che a loro volta portarono numerose modifiche in modo da poter un giorno garantire la completa stabilità e attendibilità di questo prodotto.

Network Simulator è un simulatore organizzato e gestito ad eventi discreti che permette all'utente finale di testare il comportamento di una grande varietà di reti basate sull'IP, è in grado di simulare protocolli di livello 4 (riferito alla scala ISO/OSI) come TCP e UDP, utilizzare una politica di gestione delle congestioni come RAP e TFRC, gestire le code nei router attraverso tecniche come DropTail (simile alla FIFO), RED e CBQ, sfruttare numerosi algoritmi di routing, simulare trasmissioni Unicast e multicast, simulare i nuovissimi sistemi di rete WIRELESS ed infine ricreare protocolli basilari per la creazione di LAN.

Capitolo 1
Installazione

Ora esamineremo la parte pratica del lavoro: l'installazione.
Come ben saprete, molto spesso i progetti open-source risultano instabili e poco pratici da maneggiare, dato che solitamente vengono creati per essere ulteriormente sviluppati e quindi mirati ad una clientela di "esperti" nel settore.
NS fa esattamente parte di questa categoria di programmi.
Un'altra grande caratteristica degli open-source, è quella di essere sempre (o quasi) testati su un altro prodotto poco gradevole da usare: LINUX.
Questa accoppiata può creare subito moltissimi grattacapi, prendiamo un esempio che renda bene l'idea di queste due problematiche: partiamo dal fatto che gran parte degli utenti non sappia nemmeno avviare Linux (o una delle sue release) e che per forza di cose disponga solamente di un banalissimo Win9x o addirittura di un WinXP, solo con questa premessa abbiamo già un grossissimo problema: trovare una modalità d'uso sotto ambiente Windows che sia perlomeno stabile. La prima soluzione è quella di scaricarsi un compilatore in C++, imparare a maneggiare il linguaggio e installare i file Binari di NS; oppure, qui di seguito, vi illustrerò come fare per risparmiarvi un sacco di problemi.

Come già detto, NS funziona abbastanza bene su LINUX (molti utenti di internet vi consiglieranno di installarlo sotto il "vecchio" Mandrake 7.2) usando il pacchetto che contiene tutte le librerie chiamato ALL-IN-ONE che troverete sul sito:

http://www.isi.edu/nsnam

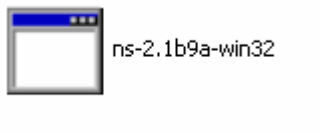
il file può essere scaricato in codice sorgente o già compilato per Win9x, Win2k o XP.
Ho provato ad installare manualmente il file ALL-IN-ONE sul nuovissimo Fedora Core e più precisamente sul FC4 che, in ordine cronologico, sarebbe l'ultimo nato in casa LINUX.
Il FC4 doveva risultare molto più completo e stabile del vecchio Mandrake 7, ma non è così.
Dopo due giorni di prove e tentativi vani per installare tutto l'occorrente per NS sono riuscito a farlo partire con la strabiliante sorpresa che a fine compilazione il NAM (Network Animator) non riusciva a partire a causa della mancanza di "alcune" librerie non presenti sul sistema operativo Fedora.

Dopo questa brutta esperienza, ho deciso di dedicare una parte della mia tesi su come far funzionare NS sotto il S.O. WinXP.

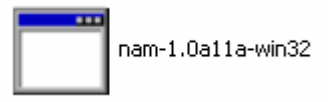
Seguite questi passi e non sbaglierete!

1 – cercate il file .exe del Tcl832.exe, installatelo (è autorun) e riavviate XP;

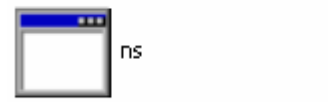
2 – prendete il file .exe del NS, fatene una copia e mettetela sotto la directory principale (c:\);



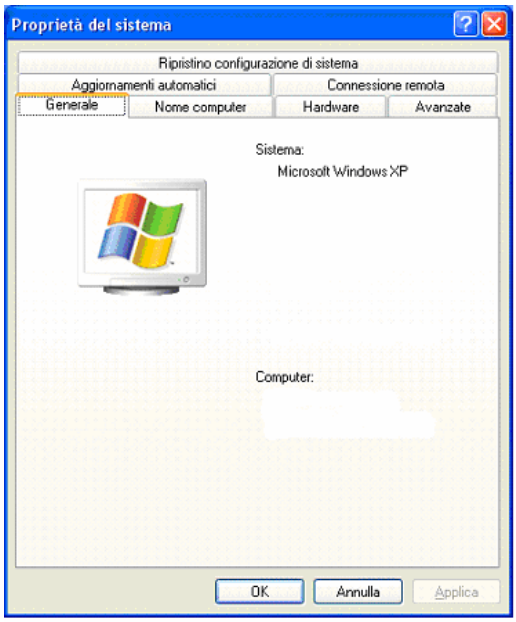
e fate la stessa cosa con il NAM ;



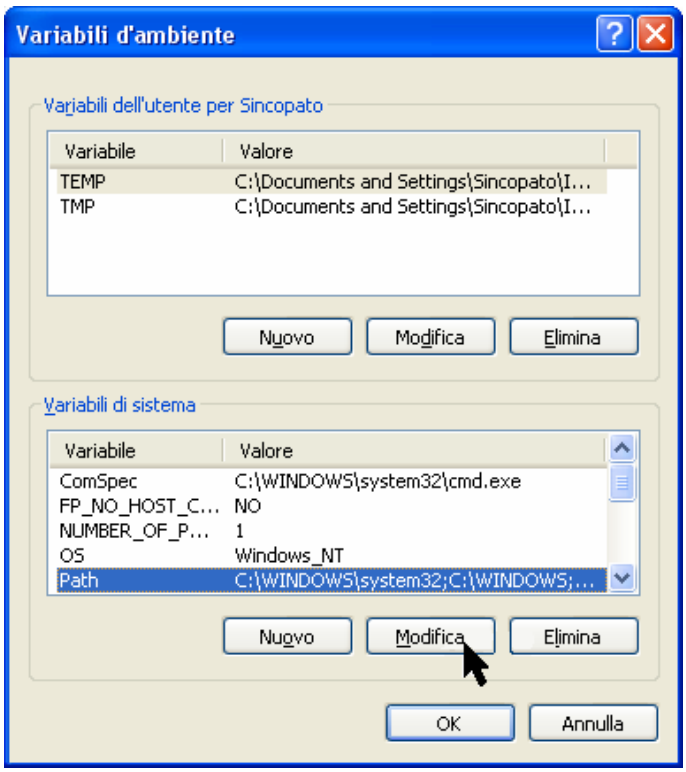
N.B.:è bene che rinominiate i due file eseguibili in **ns.exe** e **nam.exe** dato che altrimenti ogni volta dovrete scrivere tutta la pappardella nel prompt dei comandi di DOS!



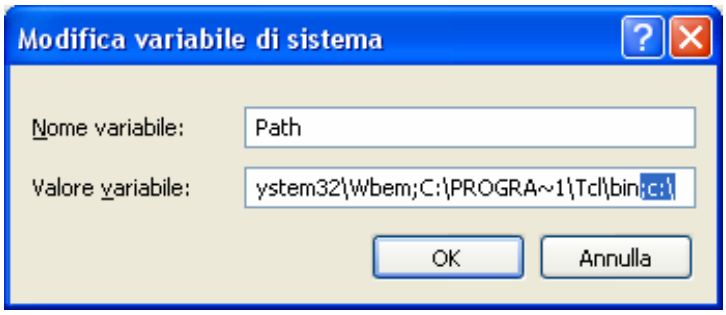
3 – Andate in Proprietà dalla cartella Risorse del computer, ciccate su avanzate e cercate tra le variabili d'ambiente il Path (variabile di Win), modificatela e introducete nel suo percorso il nuovo percorso dove avete copiato i file ns e nam (ricordatevi del " ; " per chiudere la fine della stringa del path!).



avanzate in variabili d'ambiente



modificate aggiungendo il percorso dei due nuovi file ns.exe e nam.exe;



arrivati a questo punto, salvate le modifiche e riavviate Windows.

Per verificare che la nuova installazione funzioni correttamente, aprite il prompt di DOS, digitate il comando “cd” e andate nella cartella dove sono presenti i due file .exe :

per eseguire un file di tipo .Tcl

ns nomefile.tcl

per eseguire il nam di un file .Tcl

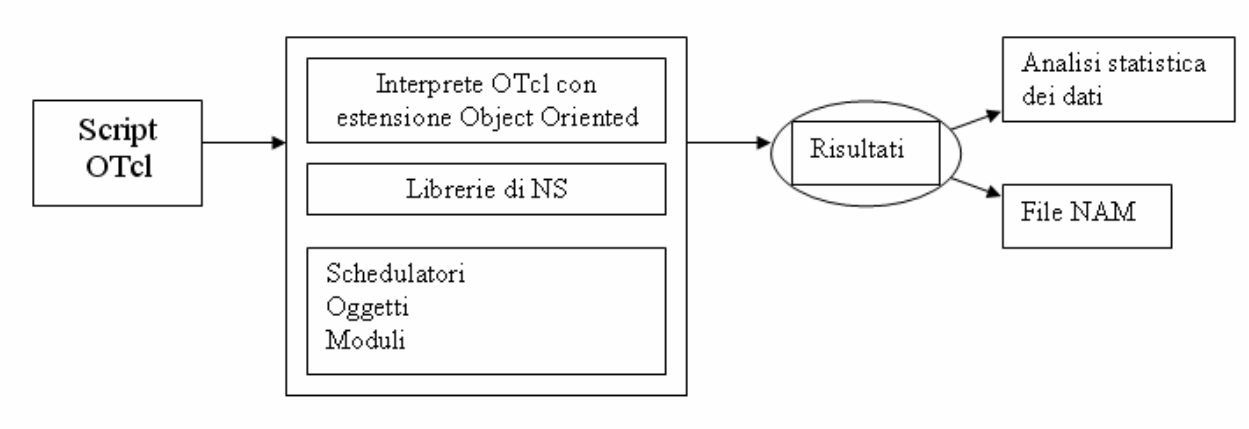
nam nomefile.nam (dipende poi se il nam è autoeseguibile dopo la compilazione e dipende anche dal nome con cui viene salvato OUT del nam)

Vi ricordo che attualmente è disponibile la versione 2.26 di NS2.

Capitolo 2

Struttura di NS

Ora andremo ad analizzare la struttura esterna di Network Simulator mettendoci nei panni dell'utente finale utilizzatore del programma:



Analizziamo le fasi dello schema:

PRIMA FASE

Script OTcl ⟶ Network Simulator utilizza come linguaggio di programmazione della parte dedicata agli script il Tcl (Tool Command Lenguage) per descrivere tutte le operazioni necessarie alla definizione degli elementi e delle proprietà di una simulazione.

SECONDA FASE

La seconda parte è composta da:

1. schedulazione degli eventi, cioè creare una "tabella" dove vengono messe in ordine cronologico tutti gli eventi che andranno ad influire sulla simulazione, come ad esempio, l'inizio e la fine di un flusso dati da parte di una sorgente, l'acquisizione dei parametri riguardanti code, capacità di trasferimento, ritardi, caduta di nodi, ecc..;

2. creazione della topologia della rete tramite gli oggetti OTcl presenti nello script e appartenenti ad eventuali classi o sottoclassi di NS (nodi, link, agenti, ecc..);

3. Utilizzo di eventuali librerie per il setup della rete, in modo da poter collegare tra loro (o nel caso di un nodo Null, di fargli scartare i pacchetti) gli oggetti presenti nello script;

TERZA FASE

La terza ed ultima fase riguarda la creazione di file di output da parte di Network Simulator (che devono essere dichiarati all'interno dello script della simulazione e devono contenere i relativi "campi di appartenenza" dei dati presi in esame) che potranno essere sottoposti a due tipi di analisi:

1. Analisi statistica e prestazionale della simulazione, in pratica, si vanno a prendere i file di output che contengono i dati relativi alla simulazione, come ad esempio, la capacità media della rete, i tempi d'impiego della trasmissione o eventuali "inconvenienti" provocati volontariamente per vedere le prestazioni in caso di interruzioni momentanee o cambiamento di tecniche di routine;

2. Il file di output viene "passato" al NAM (direttamente tramite comando OTcl o indirettamente aprendo il file .nam) per verificare se effettivamente i comandi scritti nello script rispecchiano la tipologia e le modalità della rete desiderata.
   (N.B.: esiste anche un altro programma per l'animazione delle reti: xGraph)

   Il NAM nacque nel 1990 (un anno dopo NS) e le sue animazioni vengono prodotte da un file denominato NamTrace (dichiarato nello script) che viene prodotto dallo stesso NS durante l'esecuzione dello script e che, alla fine della compilazione, apre automaticamente una finestra nella quale da vita ad un'animazione della simulazione: mostra la tipologia della rete, i flussi dei pacchetti provenienti dalle varie sorgenti (potendo anche distinguerli grazie a colori pre-dichiarati), i pacchetti in coda nei buffer (in certi casi di dimensioni "infinite"), gli eventuali link che cadono e si ripristinano e tante altre informazioni dichiarate nel file di script.
   Tutte queste attività, vengono mostrate tramite una semplice interfaccia grafica che permette all'utente di farsi un'idea (se pur relativamente ristretta) del traffico che avviene costantemente ed incessantemente in tutte le parti del mondo sulla rete di Internet.

In Ns la realtà viene modellata come un'insieme di eventi ristretti e voluti dall'utente, ora vedremo come: uno schedulatore di eventi memorizza il tempo corrente di simulazione per svolgere tutte le attività della lista degli eventi associati ad essa e fa in modo di creare una "relazione" tra gli oggetti appartenenti ad un dato evento e l'azione appropriata.
Come è facile intuire, si tratta di istanti di tempo "virtuali" che inglobano determinati eventi che influiranno direttamente sulla simulazione, non avendo nessuna relazione con il tempo effettivo (reale) consumato dalla simulazione stessa.
Gli oggetti di rete che devono simulare l'impiego di un certo tempo nel manipolare i pacchetti vengono guidati dallo schedulatore che li mette in attesa di esecuzione fino al dato istante di ritardo, in pratica, se nello script il pacchetto viene impostato con un ritardo di 0,05 secondi, lo schedulatore aspetterà 0,05 secondi prima di dare l'impulso al nodo sorgente.
Oppure, un altro esempio si può facilmente trovare nel tempo di propagazione su un determinato link, in cui la componente di rete che processa il pacchetto utilizza lo scheduler per simulare il ritardo durante l'attraversamento, da parte del pacchetto, di tutti i link presenti nella rete.

Capitolo 3
I Linguaggi

Network Simulator contiene due linguaggi di programmazione che, se pur molto diversi tra loro, riescono a convivere su questo simulatore grazie ad una corrispondenza interna. E più precisamente parleremo di:

- C++, che rappresenta il "cuore" del simulatore dove vengono impostati o aggiunti nuovi metodi o nuove classi e dove sono poste le librerie. C++, le sue librerie e le sue classi, in NS prendono il nome di GERARCHIA COMPILATA;

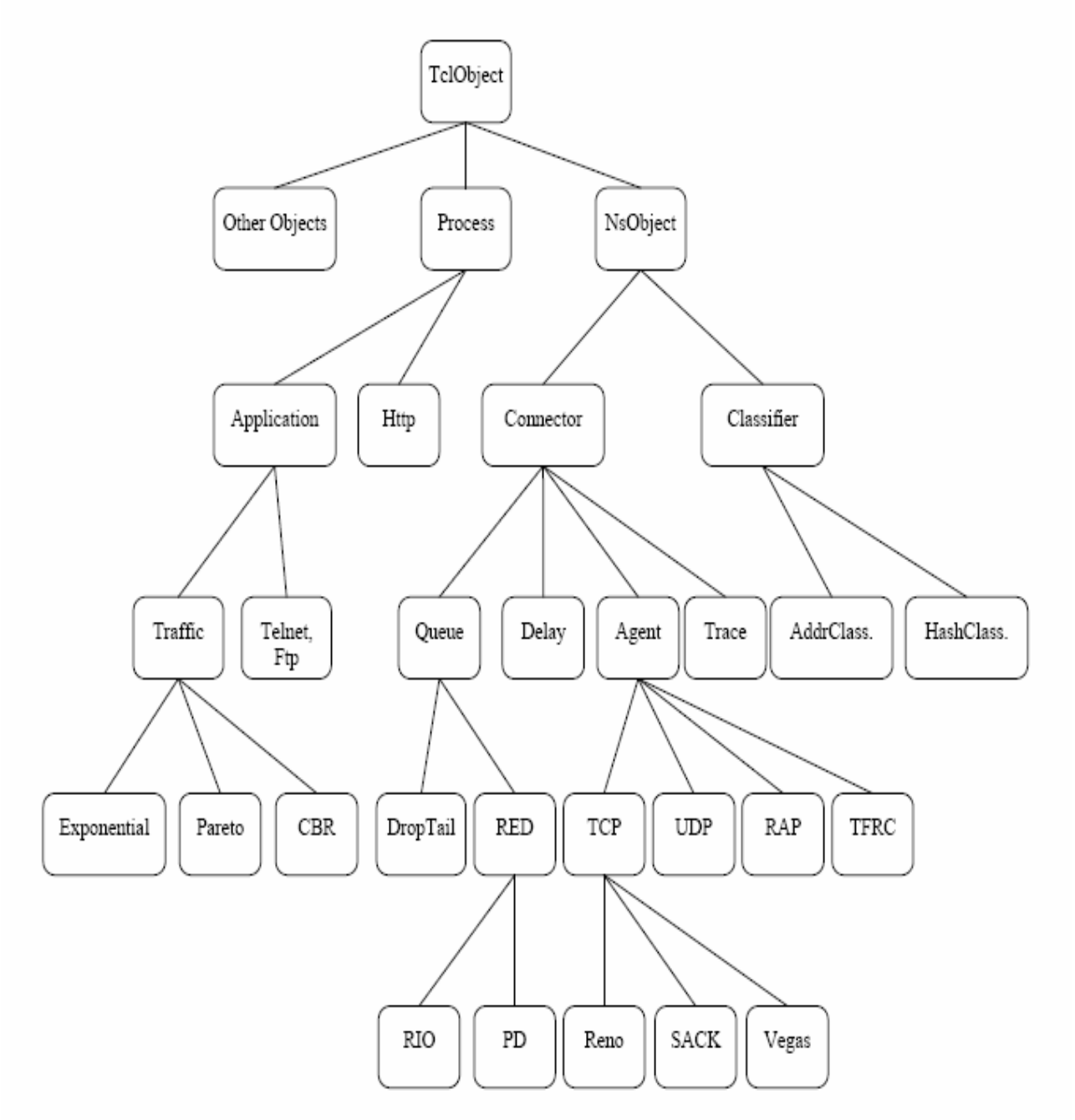
- OTcl rappresenta il secondo linguaggio presente sul simulatore e fa parte della GERARCHIA INTERPRETATA;

Il motivo di questa "convivenza" tra i due linguaggi risiede essenzialmente in questioni di efficienza, dato che in NS le funzioni di implementazione e le funzioni che riguardano la parte simulativa del software vengono nettamente distinte ma in qualche modo associate.
In poche parole, la parte implementativa viene messa da una parte e le funzioni che gestiscono il percorso durante lo svolgimento della simulazione, i controlli e il setup della rete vengono messi dall' altra.
E' per questo motivo che per riprodurre il tempo di processo dei pacchetti e degli eventi legati alle funzioni dei protocolli, agli oggetti componenti di rete e agli schedulatori di eventi relativi al percorso dati vengono implementati in C++, mentre per il setup della rete ed il controllo dei parametri durante il tempo di simulazione viene utilizzato OTcl che risulta molto più semplice (e indipendente, dato che è object oriented) e rapido per la stesura degli script.
Ritornando al discorso dell'aspetto open-source di questo prodotto, possiamo dire che ogni versione installata sul nostro pc è totalmente modificabile e personalizzabile, rendendo possibile la modifica o l'aggiunta di nuovi metodi scritti e compilati in C++ e successivamente resi disponibili con collegamenti ad OTcl.
Tra C++ e OTcl, come detto in precedenza, esiste una stretta corrispondenza è possibile definire come un collegamento che crea per ogni oggetto in C++ un corrispondente in OTcl e che traduce le funzioni membro e le variabili specificate dagli oggetti in C++ come funzioni membro e

variabili specificate del corrispondente oggetto OTcl, in modo da creare una dipendenza di OTcl a C++.

Riassumendo si potrebbe dire che , se durante la scrittura in OTcl venisse definita un'istanza (un oggetto) di una classe, nel medesimo istante, si creerebbe un'istanza dell'oggetto corrispondente alla classe compilata in C++ attraverso metodi della classe OTcl Object (si duplicherebbe da OTcl a C++ creando un collegamento).
Quindi, l'utente che ha come scopo la "semplice" simulazione, è libero di imparare solamente il Tcl senza la necessità di imparare il C++.
Mentre, lo sviluppatore utilizzerà il C++ con il quale potrà implementare gli eventi e la maggior parte delle componenti di rete relativi al percorso dati.

Ora daremo un'occhiata alla gerarchia delle classi di OTcl per poterci rendere conto come effettivamente è strutturato lo "scheletro" di OTcl:



- **Tcl Object:** radice dell'albero, dalla Tcl Object derivano tutti gli oggetti di NS;
- **Process:** classe da cui derivano tutte le entità capaci di processare, consegnare o richiedere dati da altre entità (praticamente, tutte le funzioni di livello applicativo);
- **NSObject:** classe che raggruppa tutti i componenti fondamentali della rete;
- **Connector:** sottoclasse di NSObject alla quale fanno parte i componenti con 1 solo processo di uscita nel cammino dei pacchetti;
- **Classifier:** sottoclasse di NSObject alla quale fanno parte i componenti che hanno la possibilità di scegliere tra più percorsi d'uscita nel cammino dei pacchetti.

Esamina il pacchetto in arrivo e controlla la destinazione prima di instradarlo verso il successivo;

- **Queue:** sottoclasse di connector, gestisce la politica delle code;
- **Delay:** sottoclasse di connector, l'istante successivo all'uscita del pacchetto dal nodo, delay lo trattiene nella sua classe simulando il ritardo di propagazione;
- **Agent:** classe dalla quale derivano tutti i protocollo di trasporto, di congestione e di routing;
- **Trace:** classe che "traccia" il percorso di tutti gli Agent presenti nella simulazione e quindi dichiarati nello script (in realtà tiene traccia del percorso, dato che l'instradazione avviene tramite appositi protocolli);
- **Application & Process:** coppia di classi legate alle sorgenti di traffico.

Capitolo 4
Livelli di NS

Application: la classe Application rappresenta il livello applicativo di Network Simulator, da qui nascono le applicazioni simulate ed i generatori di traffico. I generatori di traffico sono derivati dalla classe Traffic Generator a sua volta derivata dalla Application con il nome di Application/Traffic. Le sorgenti CBR (costant bit rate), la distribuzione esponenziale e la Paretiana derivano anche loro dalla classe Application/Traffic e che troveremo in NS come:
Application/Traffic/CBR;
Application/Traffic/Exponential;
Application/Traffic/Pareto.

Transport: la classe Agent è la directory principale per la creazione dei diversi protocolli di trasporto presenti nel simulatore, tra i quali possiamo contare le 3 principali tipologie di TCP:

TCP	=	Agent/TCP, si riferisce alla prima versione di TCP detta Tahoe
TCP Reno	=	Agent/TCP/Reno, seconda versione detta Reno
TCP New Reno	=	Agent/TCP/NewReno, terza versione migliorata

Poi abbiamo il protocollo UDP e i suoi derivati:

UDP	=	Agent/UDP, riferito al protocollo UDP, ma con numerazione dei pacchetti
RAP	=	Agent/RAP
RTP	=	Agent/RTP, Real Time Protocol usato per lo streaming Audio/video
FTP	=	Agent/FTP, File Transfer Protocol usato per i download e gli upload
TFRC	=	Agent/TFRC, TCP Friendly Rate Control for VOIP (Voices On IP)

Notare che TFRC e RAP vengono solitamente usati in simultanea con UDP per risolvere i problemi di congestione dato che il protocollo è molto più veloce di TCP ma completamente inaffidabile.

Network: la base degli oggetti "coda" viene definita tramite Queue, dalla quale nascono altre classi che, ereditando metodi e attributi, ne ri-definiscono di nuove creando così diverse tipologie di politica delle code adattate ad ogni caso preso in esame.

Tra le gestioni troviamo la DropTail, definita in NS con il comando Queue/DropTail, che gestisce le code con una politica di tipo FIFO (il primo pacchetto che arriva è il primo ad uscire) con eventuale scarto dei pacchetti che sopraggiungono quando ormai le risorse disponibili sono esaurite (Buffer Saturo), poi abbiamo la Queue/RED che implementa il metodo Random Early Detection, la Queue/CBQ che gestisce la coda assegnando diversi livelli di priorità ad ogni pacchetto e tante altre.

Midle…

Prima di passare all'analisi completa degli elementi base di NS, daremo un'occhiata molto veloce al concetto di SIMULAZIONE, e quale sia lo scopo di simulare ad eventi discreti.

Capitolo 5
La Simulazione

Def.: La simulazione è l'imitazione del funzionamento di un sistema reale per lo studio di particolari aspetti di un determinato problema preso in esame.
Essa si ottiene riproducendo l'evoluzione temporale del processo in questione, generando un'evoluzione artificiale.

Per simulare un evento esistono vari metodi, ma la cosa più importante risiede nel Modello di simulazione che consiste in una rappresentazione del sistema da analizzare formata da un'insieme di "parti" espresse sotto forma di relazioni matematiche, logiche o simboliche fra entità del sistema stesso.

Esistono diverse motivazioni che possono spingere alla simulazione di un evento:

- Lo studio e la sperimentazione delle iterazioni interne di un sistema complesso;
- Una migliore conoscenza del sistema reale come conseguenza della creazione del modello di simulazione;
- La valutazione What-IF (cosa succederebbe se..) senza interruzione del sistema di funzionamento o, nel nostro caso, senza interruzione del sistema operativo;
- La valutazione del funzionamento e delle prestazioni di un sistema prima della costruzione di un prototipo reale;
- La possibilità di accelerare o rallentare il tempo di un processo per poter così studiare i dettagli del fenomeno preso in esame;
- La verifica e la valutazione di soluzioni analitiche;
- Verificare la presenza di eventuali punti deboli nel sistema e la sua flessibilità nel risolverli o nell'adattarsi, come ad esempio, nel nostro caso potremmo notare la soluzione di una congestione in presenza di un protocollo TCP.

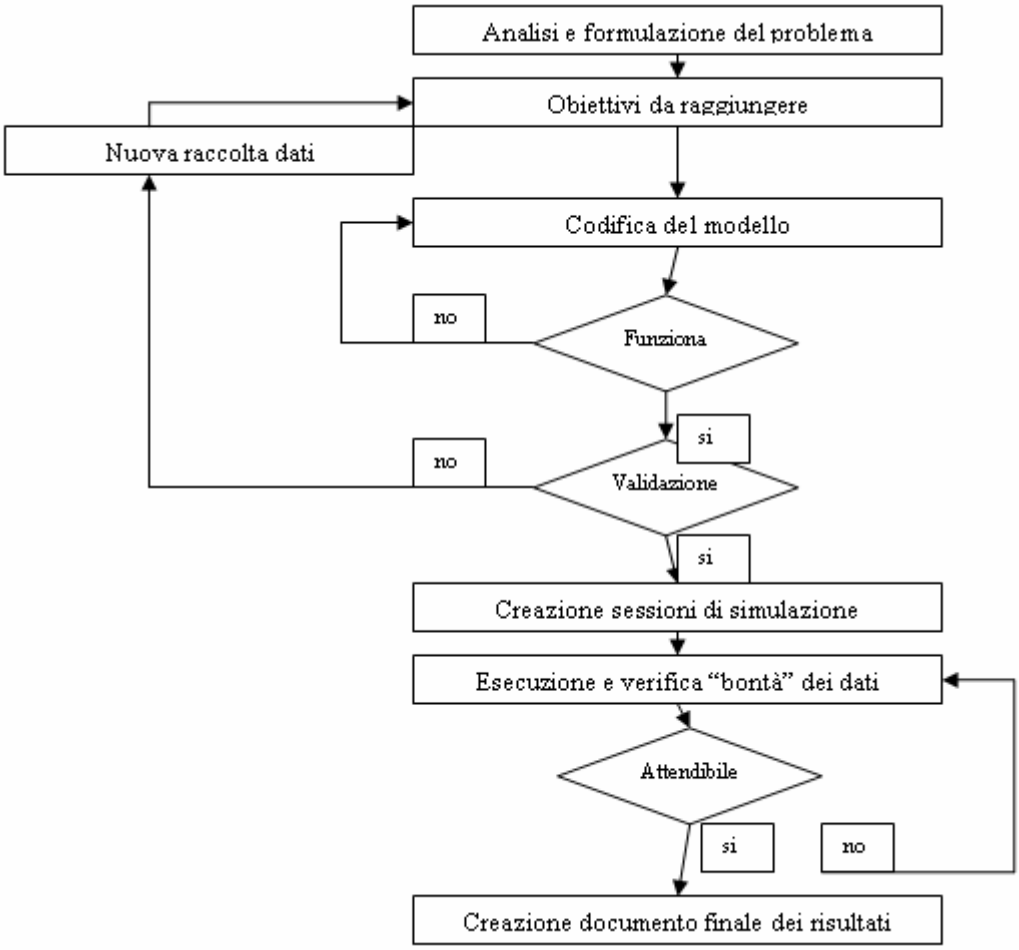
Ma esistono anche diversi aspetti negativi nell'applicare la simulazione per un evento:

- Il problema più grande risiede nel Tempo, dato che solitamente la simulazione si usa per modelli complessi, il tempo per cercare di ricreare un modello fedele sono lunghissimi;
- I risultati della simulazione potrebbero essere inaffidabili a causa della mancanza di alcune variabili ritenute poco importanti, ma che ai fini pratici negano i valori teorici;
- In caso di lavori su commissione, il guadagno è proporzionale al tempo, quindi meno tempo = più guadagno, ma si va in contro al primo punto dichiarato.

La simulazione, solitamente, si basa su di uno schema composto da quattro punti essenziali e non trascurabili:

1. formulazione del problema con definizione dettagliata degli obbiettivi da raggiungere ed eventuale pianificazione del progetto;
2. progettazione del modello con acquisizione di TUTTI i dati di input ed output del sistema REALE preso in esame. Successivamente si implementerà, si testerà e si approverà il modello;
3. definizione rigida delle variabili, studio e progettazione di tutte le parti della simulazione, analisi e test dei risultati finali e pianificazione di nuove ed eventuali simulazioni;
4. Al raggiungimento di dati ritenuti attendibili si potrà redarre il documento finale riguardante il modello e le tutte le parti da cui è composto, fornendo una lista dei risultati finali dello studio.

Vedi diagramma a blocchi del modello a 4 parti:



Capitolo 5.1

Simulazione ad eventi discreti & Object Programming

Iniziamo il discorso partendo dal concetto di variabile di stato: sono particolari variabili che caratterizzano in modo completo lo stato del sistema e con la loro dinamica riescono a definire interamente l'intero evolversi del sistema.

Parlando dei sistemi Discreti, le variabili di stato sono in grado di mutare lo stato del processo solamente in istanti di tempo appartenenti ad un insieme Discreto di possibilità, inteso come sott'insieme di tutti gli eventi possibili nella realtà ma che, ovviamente, impossibile da ricreare all'interno di una simulazione.

Risulta quindi facile intuire che, l'evoluzione di un sistema di questo tipo, avviene solo in istanti di tempo scelti dall'utente che ne fa uso e racchiusi in un insieme ristretto di casi.

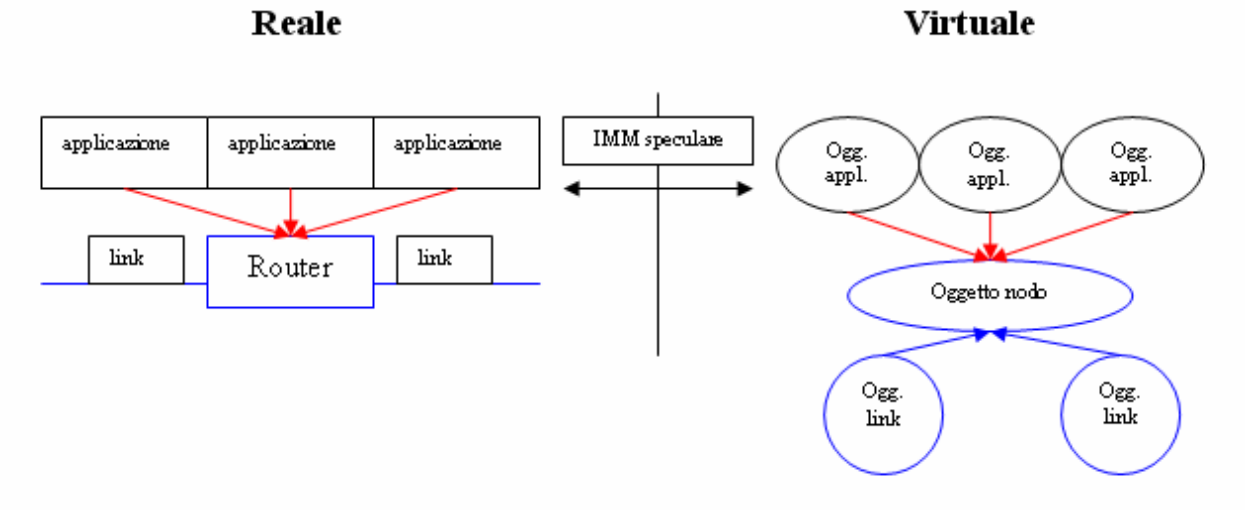
Lo studio simulativo procede (come detto in precedenza) per passi ben precisi, inizia con la scelta dello stato di partenza e le eventuali modifiche da apportare alle variabili di stato per ogni evento influente sul processo di simulazione per poi creare un "calendario degli eventi" sulla base dell'istante di occorrenza.
Arrivati a questo punto, la simulazione vera e propria consisterà nell'andare a prendere il "calendario degli eventi" e farlo scorrere evento per evento, eseguire costantemente l'aggiornamento delle variabili di stato in base agli eventi verificatosi ed, in fine, effettuare le opportune verifiche avvenute sulle variabili di stato in uscita dal processo simulativo.

Capitolo 5.2
Object programming

Immaginando una rete di telecomunicazioni, ci risulterebbe molto difficile intuirla in un unico blocco complesso e ricco di componenti, ma, se utilizzassimo il concetto di OGGETTO e più precisamente di programmazione ad oggetti, ci risulterebbe più semplice scomporre la rete in tanti blocchi (chiamati appunto oggetti) e unirli tra di loro come un grafico. Questi oggetti, di diversa natura (router, bridge, link, nodi ecc..) riescono a comunicare e interagire tra di loro per offrire all'utente numerosi servizi che, in qualche modo, risultino essere sicuri e soddisfacenti.
Questo primo discorso ci aiuta a comprendere la natura di OTcl (e quindi di Network Simulator) dove, sia la struttura interna sia l'interfaccia del simulatore vengono composte da elementi base che, collegati tra di loro cerano una rete "virtuale" di telecomunicazione: infatti, in ambiente di sviluppo della rete (lo script) si può associare ad ogni componente reale un oggetto virtuale che ne specifica (più o meno fedelmente) le caratteristiche strutturali e comportamentali.
Grazie a questa "flessibilità" degli oggetti virtuali, oggi è possibile creare modelli estremamente manipolabili che riescono ad evolversi di pari passo con l'avanzare della tecnologia delle reti di telecomunicazioni che ogni giorno migliora e si espande verso nuovi ambienti e tecnologie.

Ad ogni oggetto facente parte di una rete Reale di telecomunicazione, se ne crea uno Virtuale in grado di ricreare la struttura interna e il "comportamento" dello stesso. Questa è in sintesi la programmazione ad oggetti, cioè suddividere il problema in tante piccole parti Virtuali che ricreano le parti del problema Reale.

Un piccolo schema ci può far capire il funzionamento:



ora vediamo i vantaggi dell' Object - programming:

- Il codice flessibile del simulatore è strutturato coerentemente con la realtà e quindi è in grado di adattarsi ad ogni tipologia di rete;
- È più facile da sviluppare grazie alla sua flessibilità;
- Creare una simulazione diventa più intuitivo, infatti, basta osservare la struttura reale e simularla connettendo oggetto per oggetto.

Capitolo 5.3

Simulazione ad eventi discreti su Network Simulator

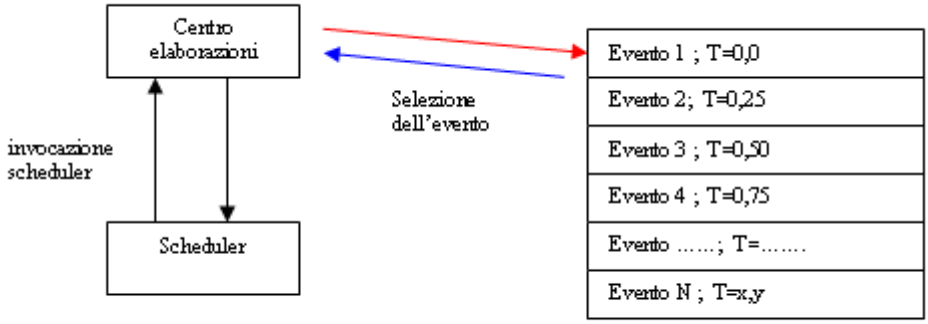
Ora assoceremo NS al discorso precedente sulla programmazione ad oggetti cercando di capire alcune proprietà del nostro simulatore.
NS appartiene alla categoria di simulatori ad eventi discreti (event-driven simulator) nei quali vengono eseguite azioni SOLO se presenti nel "calendario degli eventi" che dichiara passo per passo ogni istante di azione sul simulatore.
NS, come ogni simulatore ad eventi discreti, possiede un "centro di elaborazione" che accede automaticamente al "calendario degli eventi" attraverso uno schedulatore (scheduler) in grado di gestire le operazioni di inserimento ed estrazione degli eventi dal "calendario" in base ad un determinato criterio impostato di default: ad ogni passo, il centro di elaborazione seleziona dalla lista l'evento indicato dallo scheduler ed esegue le operazioni associate a tale evento, generando di volta in volta, nuovi eventi che verranno passati allo scheduler il quale li aggiungerà al "nuovo calendario eventi".
Tutto il meccanismo viene avviato dall'utente, la classe OTcl implementa lo scheduler che viene a sua volta creato grazie ad un'istanza della classe simulator presente su NS .
Nella versione attuale di NS (2.26) esistono due sottoclassi di scheduler:

1. **Scheduler list**: è lo schedulatore reimpostato su NS, è composto da una lista concatenata e ordinata cronologicamente (per questo motivo chiamata "calendario degli eventi"); il pregio di questo metodo sta nella semplicità della creazione della lista, ma il difetto è che una volta creata la lista degli eventi, risulta molto difficile rimuovere ed inserire eventuali eventi che per distrazione ci si è dimenticati, il problema di fondo sta nella modalità di lettura sequenziale, in poche parole, l'inserimento di un nuovo evento comporta lo scorrimento di tutta la lista;

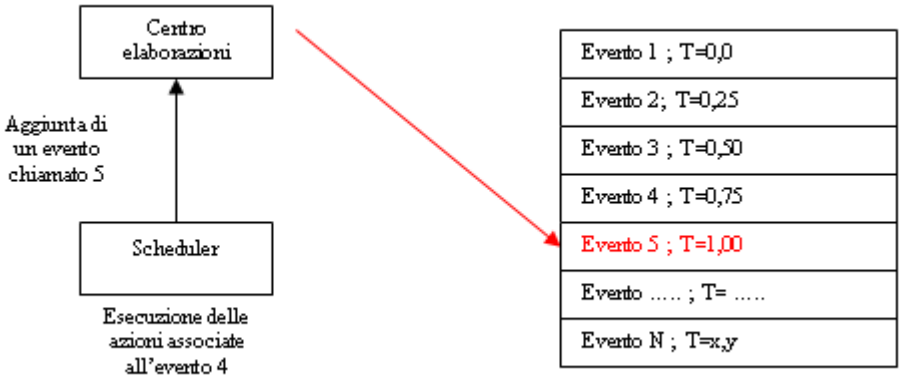
2. **Scheduler Real Time**: è in grado di sincronizzare gli eventi simulati con il tempo reale di scorrimento ; questa proprietà permette al simulatore di emulare reti a bassa velocità ricopiando esattamente le reti Reali.

Vediamo lo schema della struttura di elaborazione:

1) scelta dell'evento



2) aggiornamento in presenza di un nuovo evento



dopo questi cenni sulla simulazione, sugli eventi discreti e sulla programmazione ad oggetti, il prossimo passo sarà l'introduzione a NS ed ai suoi script.

Capitolo 6
Introduzione agli SCRIPT di NS2

Elementi base:

Creazione di un oggetto simulator

Il primo passo della creazione di uno script risiede nella classe Simulator di NS dato che l'oggetto chiamato Simulator è necessario per dichiarare una simulazione, ossia un'istanza della simulazione stessa richiamata dall'istruzione OTcl:

```
set ns [new Simulator]
```

La variabile ns viene creata tramite questa istruzione e dichiarata come istanza alla quale si potrà fare riferimento durante la stesura dello script.

Capitolo 6.1

I NODI

In ordine di apparizione nella composizione di uno script compaiono i nodi, le istruzioni per crearne uno sono dichiarate in OTcl come:

```
set n1 [$ns node]
```

ad ogni nodo viene assegnato per default un indirizzo IPv4.
L'assegnazione dell'indirizzo viene fatta nel seguente modo:
la lunghezza standard dell'indirizzo è di 16 bit, dei quali, gli 8 bit più significativi (più a sinistra) definiscono l'identificatore del nodo, mentre gli 8 bit meno significativi (più a destra) servono per distinguere i vari agenti passanti sul nodo.
Dunque, facendo due conti, la rete più grande realizzabile sarà formata da 2^8=256 nodi!
In realtà, esiste il comando "node expandaddr" che mi aumenta il numero di bit dell'indirizzo e li porta da 16 bit a 30 bit, e la nuova composizione sarà composta da: i primi 22 bit

identificheranno il nodo mentre gli ultimi 8 bit mi definiranno gli utenti. Sempre parlando di numeri, si passerà dai 256 nodi ai 4.194.304 nodi.
Passiamo al tipo di rete, di default i nodi di NS vengono creati per supportare simulazioni di tipo Unicast, ciò nonostante, si potrà mutare la forma di comunicazione in Multicast settando la variabile EnableMcast_ e portandola al valore1 prima della creazione dei nodi con il seguente comando OTcl:

Simulator set EnableMcast_1

Riassumendo, nel caso di trasmissioni multicast il bit più significativo di ogni indirizzo mi dirà se quel determinato indirizzo appartiene ad una comunicazione di tipo Unicast o multicast, se il bit più significativo risulterà essere uguale a 0 l'indirizzo sarà di tipo unicast, se risulterà essere settato a 1 sarà di tipo multicast. Questo bit, toglie un altro spazio per i il numero totale di nodi, quindi, una rete multicast con indirizzi a 16 bit avrà solo 128 nodi, dato che dagli 8 bit precedenti uno verrà usato per distinguere le unicast dalle multicast.

Per la creazione standard di nodi si può ricorrere ad un "trucco" chiamato ciclo FOR (uguale a quello presente in altri linguaggi di programmazione) in grado di generare un array di nodi. Vediamo l'istruzione in OTcl:

```
for {set i 0} {$i < N} {incr i}
    { set n($i) [$ns node] }
```

ricordando che: INCR incrementa solo di 1 unità per ciclo
N è il numero totale di nodi che voglio creare nella mia rete
I è il numero dal quale ha inizio la numerazione dei nodi

esiste anche un comando che consente di colorare ogni nodo tramite il comando color, ad esempio, se si vuole che nel NAM il nodo n1 sia distinto dal colore rosso si dovrà scrivere:

$n1 color "red"

ricordandosi che per default tutti i nodi sono neri.
Volendo rimarcare che un determinato nodo fa parte di un gruppo multicast o di un particolare gruppo di nodi, si potrà creare un secondo contorno di colore diverso (o uguale) al primo tramite il comando:

$n1 add-mark “green”

in questo caso il colore del secondo cerchio sarà verde, come dichiarato tra apici.

Capitolo 6.2

I LINKS

Come nella realtà, due oggetti di tipo nodo dovranno comunicare tramite un collegamento chiamato Link, nella programmazione ad oggetti si chiamerà oggetto link.
Esistono fondamentalmente due tipi di Link:

1- link unidirezionali detti “Simplex-link” (simplex)
2- link bidirezionali detti “Duplex-link” (full-duplex)

al contrario della realtà, non esiste un oggetto che rappresenti gli half-duplex;
la sintassi per la creazione di link è la seguente:

$ns simplex $<node0> $<node1> <bandwidth> <delay> <queue-tipe>

analizzando le componenti troveremo:
creazione di un link dal nodo <node0> al nodo <node1> ;
con capacità di banda <bandwidth> espressa per default in bps ;
con un ritardo di propagazione <delay> espresso in secondi ;
ed una politica di gestione delle code >queue-tipe> da dichiarare.

Ad esempio:

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

$ns simplex-link $n0 $n1 1,5Mb 100ms DropTail
oppure, in caso di link bidirezionali

$ns duplex-link $n0 $n1 1,5Mb 100ms DropTail

nel dettaglio:

Capitolo 6.3

la BANDA

<bandwidth> è espressa in bps (bit per second) , ma esistono numerosi suffissi da poter applicare e che ne cambiano la capacità, come ad esempio il "K" che esprime la capacità in migliaia di bps - Kbps oppure il suffisso "M" che la esprime in milioni di bps - Mbps.
Usando la "B" si esprimono i Byte invece che i bit e quindi saranno Bps.
Una lista di scritture equivalenti tra loro:

1.5 M
1.5 Mb
1500 K
1500 Kb
0.1875 MB
187.5 KB

il valore di default della banda è di 1.5 Mbits/sec.

Capitolo 6.4

il RITARDO

è espresso solitamente in secondi, ma con il suffisso "m" lo si può esprimere in millesimi di secondo, con "n" in nanosecondi e con "p" in picosecondi.
Una lista di scritture equivalenti tra loro:

1500 m
1.5
1.5 e9ns (1500000000 ns)

1500 e9p (1500000000000 ps)

il valore di default del ritardo è 100 ms.

La gestione della CODA <queue-tipe> può essere principalmente di tre tipologie:

- DropTail = disciplina di tipo FIFO;
- SFQ = Static Fair Queueing;
- Custom = creata dall'utente;

inoltre, l'utente può anche stabilire il limite massimo di pacchetti da poter mettere in coda tramite la <queue-limit> su di un link scegliendo la direzione, ad esempio un limite dal <node0> al <node1> sarà dato dall'istruzione:

$ns queue-limit <node0> <node1> <queue-limit>

ed è anche possibile creare una latenza pari a <time-interval> di percorrenza scegliendo, anche in questo caso, il verso di percorrenza come ad esempio dal <node0> al <node1> con l'istruzione:

$ns delay <node0> <node1> <time-interval>

Capitolo 6.5

oppure assegnare un **COSTO** ad ogni link di percorrenza dato da <cost-val> scegliendo da dove a dove, come ad esempio, dal <node0> al <node1> con l'istruzione:

$ns cost <node0> <node1> <cost-val>
$ns cost $n0 $n1 10
$ns cost $n1 $n0 7
sapere il costo del tragitto mi può servire per calcolare il percorso minimo tramite alcuni protocolli di routing come ad esempio Bellman-ford o Djikstra; <cost-val> può essere dichiarato solo con numeri INTERI.
Per default il costo di ogni link è pari ad 1.

Se, dovessimo legare tutti i nodi tra loro, potremmo usare ancora il ciclo FOR anche per i link, andando a creare una rete a tipologia circolare:

un esempio a 10 nodi:

```
for {set i 0} {$i < 10} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%10]) 1Mb 20ms DropTail
    }
```

l'operatore % serve a connettere l'ultimo nodo al primo, quindi sarebbe possibile anche non chiudere l'anello.

Parlando di "grafica", alla fine dello script si potrà decidere se:

a) far partire manualmente il file .nam generato dalla compilazione del file .tcl;
b) far partire automaticamente il file .nam dopo la compilazione del .tcl tramite il comando inserito nello script: **exec nam NomeFile.nam** dove NomeFile sarà il nome del file di output dichiarato ad inizio script.

Una volta partita l'animazione, ci troveremo un grafico sicuramente in "disordine", ma per riordinarlo basterà ciccare sul pulsante LAYOUT dell'apposta interfaccia, oppure, per evitare da subito il disordine, si possono dichiarare le posizioni dei nodi direttamente dalla script tramite i seguenti comandi:

```
$ns duplex-link-op <node0> <node1> orient right-up
$ns duplex-link-op <node0> <node1> orient right-down
$ns duplex-link-op <node0> <node1> orient right
$ns duplex-link-op <node0> <node1> orient left-up
$ns duplex-link-op <node0> <node1> orient left-down
$ns duplex-link-op <node0> <node1> orient left
```

Capitolo 6.6

Gli AGENTS

Gli agents sono oggetti che vengono collocati su ogni nodo ed hanno il compito si "guidare" attivamente la simulazione in corso: in pratica, sono costituiti dai processi e dalle entità di trasposto che si possono trovare su host e router.

Tutti i comandi essenziali per creare un agent fanno parte della sottoclasse Agent/<Type>.
Ad esempio, per creare un agent di tipo UDP i userà il seguente comando:

set udp1 [new Agent/UDP]

una volta creato l'agente, si dovrà collocarlo su di un nodo tramite il comando OTcl:

$ns attach-agent $n1 $udp1
questo comando assegna l'agent al numero di posto del nodo.
Tutti gli agent dispongono di varie funzioni regolabili tramite la configurazione dei loro parametri, ad esempio, si potrebbero assegnare le dimensioni dei pacchetti, usando agent di tipo TCP:

$tcp1 packetSize_500

o dimensionare la finestra:

$tcp1 set window_25

se inizialmente pongo che:

$tcp1 set window_30

la mia finestra di congestione sarà impostata a 30 pacchetti per tutti gli agent di tipo tcp, è come se dichiarassi che il default dello script in uso fosse settato a 30.

Diamo un'occhiata ai vari tipi di agent che compongono la lista di NS (solo i più usati):

1. TCP: agente di tipo tcp che attua il controllo della congestione ed una stima dell'RTT con sorgenti di traffico di tipo "TAHOE";
   La finestra di congestione viene aumentata di 1 segment per ogni PDU-ACK ricevuto durante SLOW-START, poi, superata una certa soglia di traffico, viene aumentata di 1/(dimensione corrente della finestra);

2. TCP/sack1: è la versione tcp con selective repeat;
3. TCP/Sink: messo sul nodo destinatario del traffico di rete, emette solo PDU-ACK per ogni PDU-DATI ricevuta;
4. TCP/Sink/DelAck: posto sul nodo destinatario, trasmette una PDU-ACK comulativa per ogni sessione di traffico ricevuta;
5. TCP/FullTcp: apre la trasmissione scambiando pacchetti SYN/FIN , avviene una trasmissione bidirezionali dei dati e numera i Byte invece di numerare i segment. NON è compatibile con nessuna delle altre versioni di TCP;
6. UDP: differisce dal reale protocollo UDP in quanto esegue la numerazione dei pacchetti inviati;
7. Null: agent usato solitamente per la simulazione della capacità di un buffer, solitamente viene posto sul nodo destinatario e il suo compito è quello di scartare tutti i pacchetti che gli arrivano, la scelta del pacchetto da scartare NON segue regola di gestione;
8. CBR: costant bit rate, posto su sorgenti di traffico, genera pacchetti di dimensione costante con bit rate d'invio costante (di default o reimpostato nello script);

Solitamente in una sessione di simulazione vengono creati uno o più agenti, ma che appartengono a due categorie principali, ovvero, possono essere:

1 – sorgente <src>
2 – destinatario <dest>

ed hanno bisogno, ovviamente, di una connessione tra di loro per potersi scambiare pacchetti ed il comando OTcl è il seguente:
$ns connect <src> <dest>

vediamo ora un esempio reale per fissare le idee, creiamo un agent di tipo UDP chiamato udp1 posto sul nodo n1 ed un agent di tipo Null chiamato null0 posto sul nodo n0:

$ns connect $udp1 $null0

la connessione tra sorgente e destinatario è SEMPRE unica!

Immaginiamo, per esempio, di voler trasmettere con due sorgenti di traffico su di un unico nodo destinatario: dovremmo creare su questo nodo due diversi destinatari std1 e std2 e connettere ognuno di essi con una delle due sorgenti di traffico. Se si richiede una connessione su di un destinatario che ne ha già una stabilita un'altra, la seconda ad arrivare avrà la precedenza sulla prima, e così via, pacchetto dopo pacchetto. Questo non significa che ogni volta il TCP deve ristabilire una nuova connessine, perché quella presente non cade, ma viene messa in pausa.

Per il calcolo dei pacchetti appartenenti ad un flusso di un determinato agent bisogna eseguire i seguenti comandi:
$ns color 1 blue
$ns color 2 red

$tcp1 ste fid_1
$tcp2 fid_2

in questo modo potrò distinguere i diversi flussi di dati grazie al colore differente dei 2 agent e potrò andare a calcolare i pacchetti di uno e i pacchetti dell'altro.

Capitolo 6.7

Gli EVENTS

Gli ultimi oggetti citati per la creazione di uno script di base sono gli events, in pratica tutte le voci presenti nel "calendario degli eventi" sopra citato.
Ricordarsi SEMPRE che di un event va dichiarato obbligatoriamente sia l'inizio che la fine, altrimenti la simulazione non avviene in modo corretto o agisce in modo non desiderato.
Ad esempio, si deve dichiarare in quale istante una sorgente deve iniziare a generare traffico e quando deve interrompere il traffico, dichiarare l'istante di un'eventuale caduta di un link e l'istante del suo ripristino, si devono dichiarare le connessioni tra agent, si devono dichiarare gli elenchi di nodi appartenenti a gruppi multicast, si deve obbligatoriamente dichiarare l'INIZIO e la FINE della sessione di simulazione.

In NS tutti gli eventi appartengono alla classe "AT-EVENT" e sono procedure la cui esecuzione DEVE avvenire in istanti precisi di tempo.

Per default, la simulazione inizia a 0,0 secondi.

Tra gli events esistono anche procedure personalizzate che possono essere richiamate tramite il comando PROC e per farle entrare in esecuzione si deve digitare la seguente riga di comando:

$ns at time “PROC”
le procedure personalizzate seguono la regola generale della dichiarazione dell’istante di partenza, quindi non va mai omesso l’istante “time”
nell’insieme delle procedure risiedono tutti i metodi, tra i quali:

attach-agent, detach-agent, cost,
connect, disconnect, set, color,
join, add-mark,
e molte altre.

Solitamente, ogni utente crea la propria procedura chiamata FINISH che, come dice il nome, pone fine alla simulazione con la chiusura del trafficfile (file su cui vengono “tracciate” tutte le informazioni appartenenti alla simulazione) e la creazione del file di NAM che parte manualmente o automaticamente dopo la compilazione del file.tcl.

Elenchiamo alcuni esempi di events:

- All’istante time, la sorgente src inizierà a spedire pacchetti

  $ns at time “<src> start”

- All’istante time, la sorgete src bloccherà l’ivio di pacchetti

  $ns at time “<src> stop”

- Caduta del link tra <node1> e <node2>

  $ns rtmodel-at time down <node1> <node2>

- Ripristino del link caduto in precedenza

  $ns rtmodel-at time up <node1> <node2>

e molti altri esempi li potrete trovare negli script delle ultime pagine.

Capitolo 6.8

ROUTING

Ora daremmo un breve cenno sulle tecniche di routing presenti in NS .
In NS sono state implementate diverse tipologie di routing che sostanzialmente si dividono in due grandi categorie (come nella realtà):

Statico
Dinamico

Il comando per la dichiarazione del tipo di strategia desiderata ed il protocollo di routing specifico è unico, anche perché alcune strategie di routing vengono associate ad un determinato protocollo venendo così considerate esse stesse dei veri e propri protocolli di routing.
Il comando è il seguente:

$ns rtproto <routing-proto>

dove al posto della parola <routing-proto> verrà inserita la tipologia del protocollo di routing che si intende utilizzare nella sessione di simulazione (static, session, LS, DV, manual ecc.)
Per default, una simulazione di tipo unicast mantiene un routing di tipo statico nel quale il percorso dei pacchetti per ogni coppia src/dest viene fissato una volta per tutte sulla base del costo dei vari link che, come anticipato, possono avere costi diversi.
In pratica, viene usato un criterio del costo minimo per determinare il valore del tracciato tra sorgente e destinatario questo significa che i pacchetti scelgono da subito la via meno costosa.

Abbandoniamo il discorso NS per parlare nuovamente di concetti teorici legati a quanto detto fino ad ora.

Per prima cosa vedremo alcuni aspetti della tabella ISO/OSI, poi daremo qualche cenno sui protocolli protagonisti nell'IP che sono il TCP e l'UDP ed in fine volgeremo l'attenzione sullo Streaming (più precisamente sul protocollo RTP) e su come si possa simulare su NS cercando di vedere un pò di valori numerici.

Capitolo 7
ISO/OSI

Iniziamo il discorso distinguendo le due principali categorie che suddividono il mondo delle reti:

1. OSI Reference Model , il modello base su cui si creano i protocolli;
2. Internet Protocol Suite, nel linguaggio comune viene chiamata architettura TCP/IP o anche TCP/IP reference model;

distinguiamo ora le due parti:

MODELLO di RIFERIMENTO: struttura in grado di definire il numero, le relazioni e le caratteristiche funzionali dei livelli ma non dichiara esplicitamente il protocollo da usare.

ARCHITETTURA di RETE: definisce esplicitamente livello per livello i protocolli effettivi da utilizzare.

PARTE 1

Capitolo 7.1
Il modello OSI

L'OSI (Open Systems Interconnection) appartiene alla categoria dei Reference Model è il frutto del lavoro della ISO (International Standard Organization), ed ha lo scopo di:

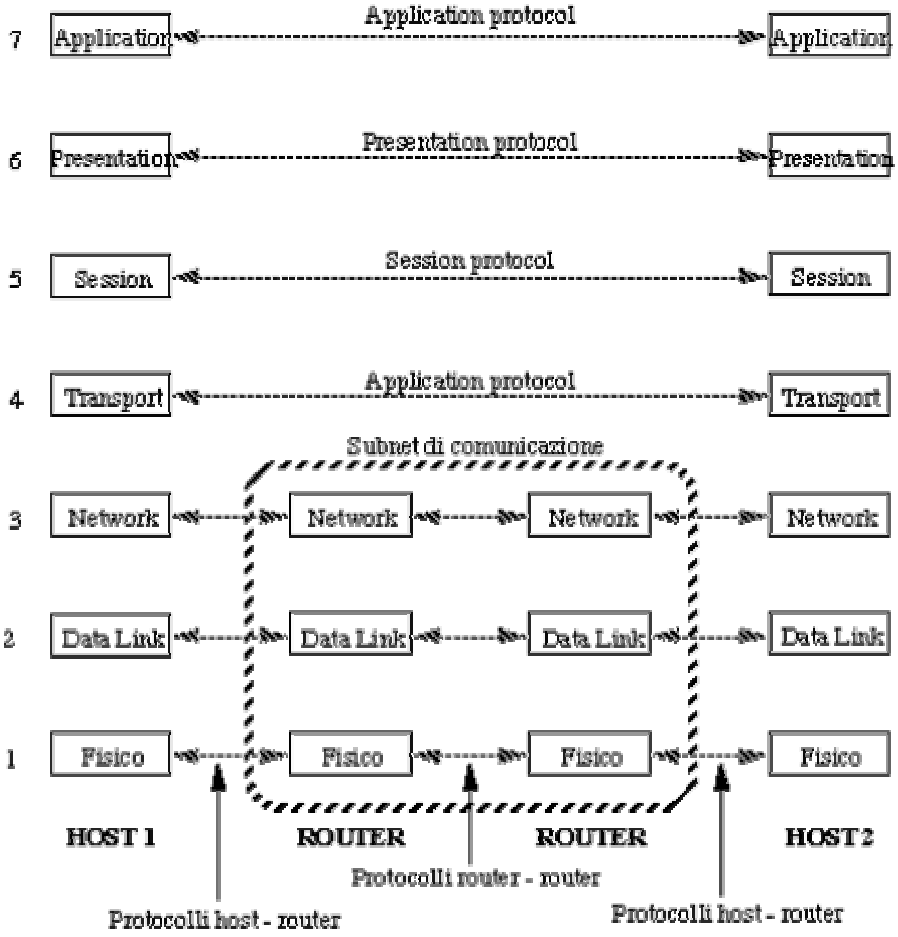
- fornire una base comune per lo sviluppo di standard per l'interconnessione di sistemi;
- fornire un modello rispetto a cui confrontare le varie architetture di rete.
- fornire uno standard per la connessione di sistemi aperti, cioé in grado di colloquiare gli uni con gli altri;

Bisogna tener presente, come detto in precedenza, che l'OSI non dichiara l'uso di protocolli specifici dato che per natura rappresenta solamente un MODELLO.

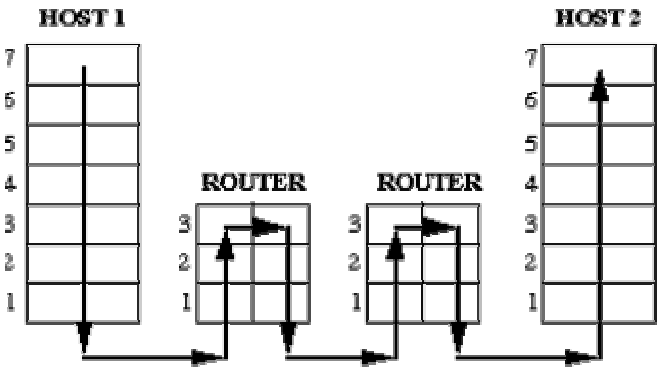
Le sue caratteristiche sono:

- ogni livello deve avere un diverso livello di astrazione;
- ogni livello deve avere una funzione ben definita;
- ogni livello deve minimizzare il passaggio delle informazioni fra livelli, non deve avere troppi livelli interni o troppe funzioni di livello;

Il modello OSI si raggruppa in 7 livelli:



e per la visualizzazione delle funzioni svolte dai diversi computer su di un tratto di rete (cammino), si fa spesso uso di diagrammi simili a questo:



N.B.: in caso di esercizi su architetture TCP/IP i livelli degli host vengono ridotti a 4, parlando di questa architettura, vedremo anche il perché di questa modifica.

Si noti che il modello OSI non è un'architettura di rete, perché dice solo i compiti dei livelli, ma non definisce né i servizi né i protocolli. Per questo ci sono separati documenti di definizione degli standard.

Ora entreremo nel dettaglio dei vari livelli che compongono l'OSI:

Livello 1: Fisico

Gestisce la trasmissione dei bit sottoforma di segnali analogici su di un canale di telecomunicazione.
Questo livello dovrebbe garantire che, se da una sorgente di traffico venisse spedito un bit con valore 1, ne arrivi uno al destinatario con valore 1 e non con valore 0.
Diciamo che il livello fisico rappresenta le parti meccaniche, elettriche e procedurali delle interfacce di rete (tutte le componenti che permettono al PC di connettersi alla rete) e le caratteristiche specifiche del mezzo utilizzato.
I punti principali studiati per il livello fisico sono:

- Le tensioni per rappresentare gli 0 e gli 1;
- La durata di un singolo bit;
- trasmissione half o full duplex;
- forma dei connettori;

Livello 2: Data Link

Lo scopo di questo livello è far si che un mezzo fisico trasmissivo appaia, al livello superiore, come una linea di trasmissione esente da errori di trasmissione non rilevati.

Normalmente funziona così:

- spezzetta i dati provenienti dal livello superiore in frame (da qualche centinaia a qualche migliaia di byte);
- invia i frame in sequenza;
- aspetta un acknowledgement frame (ack) per ogni frame inviato.

Ha il compito di:

- aggiunta di delimitatori (framing) all'inizio ed alla fine del frame;
- gestione di errori di trasmissione come ad esempio, errori in ricezione, perdita di frame, o duplicazione di frame (da perdita di ack);
- regolazione del traffico (per impedire che il ricevente sia "sommerso" di dati);
- meccanismi per l'invio degli ack: frame separati (che però competono col regolare traffico nella stessa direzione),o gestione del piggybacking (da pickaback, cioé attaccare PDU-ACK su PDU-DATI).

E problemi anche sul tipo di trasmissione utilizzata, ad esempio: il controllo dell'accesso al canale trasmissivo, che è condiviso. Per questo hanno uno speciale sottolivello del livello data link, il sottolivello MAC (Medium Access Control).

Livello 3: Network

Lo scopo di questo livello è quello di controllare la rete di comunicazione;

Inizialmente tale livello offriva solamente servizi connection oriented (come il protocollo TCP); successivamente fu aggiunta la modalità connectionless (come il protocollo UDP).

Ha il compito di:

- routing, cioé scelta del cammino da utilizzare che può essere:
    - statico (fissato ogni tanto e raramente variabile);
    - dinamico (continuamente aggiornato, anche da un pacchetto all'altro);
- gestire la congestione: a volte troppi pacchetti arrivano ad un solo router;
- convertire i dati nel passaggio fra una rete ed un'altra:
    - indirizzi da rimappare;
    - pacchetti da frammentare;
    - protocolli diversi da gestire.

Livello 4: Transport

Lo scopo di questo livello è accettare dati dal livello superiore, spezzettarli in pacchetti, passarli al livello network ed assicurarsi che arrivino alla peer entity che si trova all'altra estremità della connessione. In più, fare ciò efficientemente, isolando i livelli superiori dai cambiamenti della tecnologia di rete sottostante.

Il livello transport è il primo livello realmente end-to-end, cioé da host sorgente a host destinatario: le peer entity di questo livello portano avanti una conversazione senza intermediari.

Si noterà che certe problematiche sono, in ambito end-to-end, le stesse che il livello data link ha nell'ambito di una singola linea di comunicazione; le soluzioni però sono alquanto diverse per la presenza della subnet di comunicazione.

Ha il compito di:

- creare le connessioni di livello network (attraverso i servizi del livello network) per ogni connessione di livello transport richiesta:
    - normalmente, una connessione network per ciascuna connessione transport;
    - ottenere un alto throughput (nella medesima connessione(multiplazione)): molte connessioni network per una singola connessione transport;
- offrire i vari servizi al livello superiore:
    - canale punto a punto affidabile, che consegna i dati in ordine e senza errori (il servizio più diffuso, connection oriented);
    - invio di messaggi isolati, con o senza garanzia di consegna (connectionless);
    - broadcasting di messaggi a molti destinatari (connectionless).

I prossimi 3 livelli sono poco importanti ai fini pratici, ma è importante accennarli data la loro importanza nello sviluppo delle reti.

Livello 5: Session

Ha a che fare con servizi più raffinati che non quelli del transport layer, ad esempio:

- il token management: autorizza le due parti, a turno, alla trasmissione.

Livello 6: Presentation

E' interessato alla sintassi ed alla semantica delle informazioni da trasferire. Ad esempio, si occupa di convertire tipi di dati standard (caratteri o interi) da come vengono espressi dalla sorgente a come verranno consegnati al destinatario.

Livello 7: Application

Prevede che qui risieda tutta la varietà di protocolli che sono necessari per offrire i vari servizi agli utenti, quali ad esempio:

- **terminale virtuale**;
- **trasferimento file**;
- **posta elettronica**.

Attraverso l'uso di questi protocolli si possono scrivere applicazioni che offrono i suddetti servizi agli utenti finali.

PARTE 2

Capitolo 7.2
Internet Protocol Suite

Oggi, possiamo facilmente affermare che la "madre di tutte le reti" fu Arpanet, originata da un progetto di ricerca finanziato dal DoD (Department of Defense) americano.

Lo scopo iniziale era quello di creare una rete estremamente affidabile che, anche in caso di catastrofi o (più probabilmente) eventi bellici, riuscisse a rimanere intatta e salvare le informazioni da essa contenute.

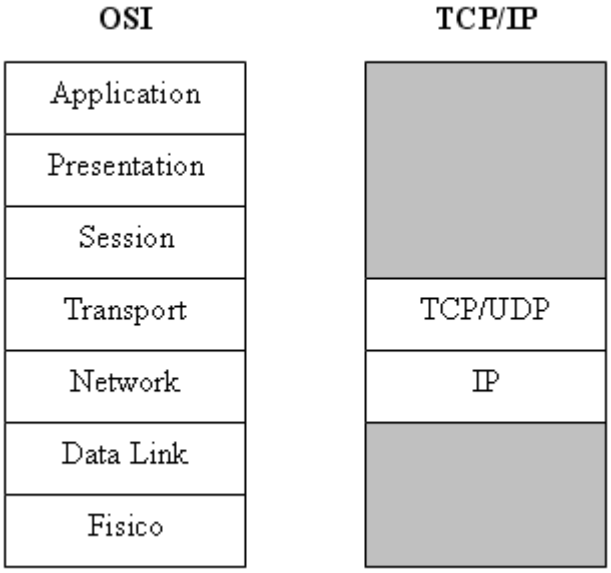
Arpanet, attraverso varie evoluzioni, ha dato origine alla attuale Internet.

Nel corso dello sviluppo, per integrare i vari tipi di reti, si vide la necessità di una nuova architettura, mirata fin dall'inizio, a consentire l'interconnessione di molteplici reti (internetwork).

L'architettura divenne, più tardi, nota coi nomi di Internet Protocol Suite, architettura TCP/IP e TCP/IP reference model (dal nome dei suoi due protocolli principali).

Essa non è un modello nel senso stretto del termine, in quanto include i protocolli effettivi che sono specificati per mezzo di speciali documenti detti RFC (Request For Comments).

Ora vedremo ed analizzeremo i 4 livelli del TCP/IP confrontandoli,da subito, con la scala OSI:



I requisiti del progetto stabiliti fin dall'inizio erano:

1. estrema affidabilità;
2. tolleranza ai guasti;
3. possibilità di interconnessione di più reti

che portarono alla scelta di una rete con:

- packet-switched (passaggio di pacchetti);
- basata su un livello connectionless di internetwork (senza aperture di sessioni di connessione).

Livello 1 - 2

Sono i livelli più bassi e non specificati nell'architettura, dato che prevedono l'utilizzo dei protocolli disponibili per le varie apparecchiature disposte sulla rete in grado di "adattarsi" alla spedizione/ricezione di pacchetti.

In pratica, i livelli 1 e 2 riguardano tutto ciò che assume la capacità da parte dell'host di inviare/ricevere dei pacchetti sulla rete.

Livelli IP e TCP

Internet Protocol

L'Internet Protocol (IP) è un protocollo di rete a commutazione di pacchetto;

Secondo la classificazione ISO/OSI è di livello network (3).

La versione correntemente usata del protocollo IP è detta anche IPv4 per distinguerla dalla più recente IPv6, nata dall'esigenza di gestire meglio il crescente numero di computer connessi ad Internet.

IP è un protocollo di interconnessione di reti (Inter-Networking Protocol), nato per interconnettere reti eterogenee per tecnologia, prestazioni, gestione.

I protocolli di trasporto utilizzati su IP sono soprattutto TCP e UDP.

Indirizzo IP

All'interno di una rete ad ogni interfaccia connessa alla rete fisica viene assegnato un indirizzo univoco, in modo da rendere possibili le comunicazioni tra un computer e l'altro.

Va considerato, infatti, che non è l'host ad essere connesso ma è l'interfaccia fisica (ad esempio una scheda di rete);

Indirizzi IP ed indirizzi MAC

I protocolli di collegamento, al livello 2 del modello ISO/OSI, indirizzano i calcolatori utilizzando il MAC address.

Quando su una rete locale si utilizza IP, ad ogni calcolatore deve essere assegnato anche un indirizzo IP, per permettergli di comunicare con i calcolatori al di fuori della sua rete locale.

L'assegnazione di un indirizzo IP ad un calcolatore può essere manuale, o automatizzata da protocolli come DHCP o dai meno conosciuti BOOTP e RARP..

DHCP

Il DHCP (Dinamic Host Configuration Protocol) è un protocollo di rete che permette ai dispositivi che ne facciano richiesta di essere automaticamente configurati per entrare a far parte della LAN. Tramite questo protocollo, non è necessario specificare manualmente nessun parametro di rete, in quanto il DHCP-server fornisce tutte le informazioni necessarie al client nel momento stesso in cui questo effettua il boot.

Ovviamente si tratta di una soluzione molto comoda per tutti coloro che usano computer portatili

in quanto tutte le volte che si muovono da una LAN all'altra, non devono manualmente specificare nulla: basta che inseriscano la presa di rete in una porta attiva, e il DHCP-server farà il resto.

IPv4

L'indirizzo IPv4 è formato da 32 bit;

viene descritto con 4 numeri decimali rappresentati su 1 byte (quindi ogni numero varia tra 0 e 255) separati dal simbolo "punto";

un esempio di indirizzo IPv4 è 192.0.34.166.

Tali numeri individuano la rete e l'host mediante algoritmi e tecniche particolarmente complesse.

Tale standard si sta rivelando limitato rispetto alla crescita di Internet per cui si è pensato di rivederlo introducendo IPv6.

Il numero di indirizzi univoci disponibili in IPv6 è di circa 3,4 x 10^38 , ma anche di questi alcuni sono riservati.

IPv6

L'indirizzo IPv6 è costituito da 128 bit; viene descritto da 8 numeri esadecimali rappresentati su 8 byte (quindi ogni numero varia tra 0 e 65535) separati dal simbolo "due punti".

Un esempio di indirizzo IPv6 è 2001:0DB8:0000:0000:0000:0000:0000:0001,

che può essere abbreviato in 2001:DB8::1 (i due punti doppi stanno a sostituire la parte dell'indirizzo che è composta di soli zeri consecutivi. Si può usare una sola volta, per cui se un indirizzo ha due parti composte di zeri la più breve andrà scritta per esteso).

Al pari del livello Transport, troviamo nel TCP/IP i protocolli più famosi dell' Internet Protocol Suite:

- **TCP (Transmission Control Protocol)**:
    1. è un protocollo connection oriented e ordered deliveri ossia tutti i pacchetti arrivano e nel giusto ordine di trasmissione;
    2. Frammenta il flusso in arrivo dal livello superiore in tanti “messaggini” separati che verranno poi passati al livello3 Internet; In arrivo, i pacchetti verranno riassemblati in un flusso di output per il livello superiore.
    3. Utilizza traffico di feedback con protocolli a finestra chiamati ARQ (Ask Reponds when reQuest) ai quali appartengono 3 principali tecniche di trasmissione:

1. S&W: stop and waith;
2. GB'n: go-back-n;
3. SR: selective repeat

4. Utilizza PDU-ACK per la conferma della ricezione di pacchetti o la richiesta di trasmissione;
5. Controlla il flusso ed eventuali errori;

- **UDP (User Datagram Protocol)**:

  è un protocollo non connesso e **soprattutto** non affidabile, i pacchetti possono arrivare in ordine diverso o non arrivare affatto ma la sorgente non può accorgersi del traffico mancante dato che UDP non usa traffico di feedback.

Nell'architettura TCP/IP non ci sono i livelli session e presentation (non furono ritenuti necessari; l'esperienza avuta con il modello OSI ha mostrato che questa visione è condivisibile).

Sopra il livello transport c'è direttamente il livello che contiene tutti i protocolli di alto livello che vengono usati dalle applicazioni reali presenti sulle varie macchine.

I primi protocolli furono:

- Telnet: terminale virtuale;
- FTP (File Transfer Protocol): file transfer (Up e Down load);
- SMTP (Simple Mail Transfer Protocol) e POP (Post Office Protocol): posta elettronica.

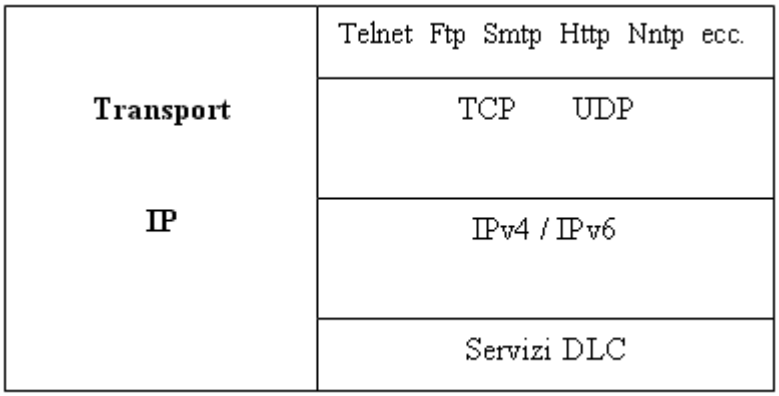
Successivamente se ne aggiunsero altri, tra i quali possiamo elencare:

DNS (Domain Name Service): mapping fra nomi di host e indirizzi IP (ovviamente dichiarati in formato numerico);

NNTP (Network News Transfer Protocol): trasferimento di articoli per i newsgroup;

HTTP (HyperText Transfer Protocol): alla base del Word Wide Web.

I vari protocolli nell'architettura TCP/IP si collocano come segue:



Ora, proviamo a dare un'occhiata alle differenze a elle somiglianze tra il modello OSI e l'architettura TCP/IP.

Somiglianze:

- basati entrambi sul concetto di pila di protocolli indipendenti;
- funzionalità simili in entrambi per i vari livelli.

Differenze:

- OSI nasce come modello di riferimento (utilissimo per le discussioni generali), i protocolli vengono solo successivamente;
- TCP/IP nasce con i protocolli mentre il suo modello di riferimento viene a posteriori.

Conseguenze:

essendo il modello OSI nato prima dei relativi protocolli, avvenne che:

- il modello era (**ed è tuttora**) molto generale (punto a favore);
- vi era insufficiente esperienza nella progettazione dei livelli (punto a sfavore). Ad esempio:
    - il livello data-link (pensato all'origine per linee punto-punto) ha dovuto essere sdoppiato per gestire reti broadcast;

- manca del tutto l'idea di internetworking: si pensava ad una rete separata, gestita dallo stato, per ogni nazione.

I protocolli dell'architettura TCP/IP sono invece il punto di partenza del progetto, per cui:

- l'architettura è molto efficiente (punto a favore);
- il reference model non è generale, in quanto descrive solo questa particolare architettura (punto a sfavore);
- è difficile rimpiazzare i protocolli se necessario (punto a sfavore).

I protocolli OSI non sono riusciti ad affermarsi sul mercato per una serie di ragioni:

- innanzi tutto, la scelta del periodo di divulgazione: la definizione dei protocolli è arrivata troppo tardi, quando cioé quelli TCP/IP si erano già considerevolmente diffusi. Le aziende non se la sono sentite di investire risorse nello sviluppo di una ulteriore architettura di rete;

- Le scelte tecnologiche: i sette livelli (e i relativi protocolli) sono stati dettati in realtà dalla architettura SNA dell' IBM, più che da considerazioni di progetto. Per cui il progetto soffre di vari difetti:
  - grande complessità e conseguente difficoltà di implementazione;
  - inutili i livelli session e presentation;
  - non ottimali attribuzioni di funzioni ai vari livelli:
    - alcune funzioni appaiono in molti livelli (es. controllo errore e flusso in tutti i livelli);
    - altre funzioni mancano del tutto (ad es. sicurezza e gestione rete);
- e per finire, l'implementazione: le prime realizzazioni erano lente ed inefficienti, mentre contemporaneamente TCP/IP era molto ben implementato (e per di open source e quindi gratis!).

In effetti i protocolli dell'architettura TCP/IP invece sono stati implementati efficientemente fin dall'inizio, per cui si sono affermati sempre più, e quindi hanno goduto di un crescente supporto che li ha resi ancora migliori.

Bisogna però ricordare che nemmeno l'architettura TCP/IP è priva di problemi:

- non ha utilità come modello, dato che non è generica come l'OSI e che non serve ad altro che a descrivere se stessa;
- non c'è una chiara distinzione fra protocolli, servizi e interfacce, il che rende più difficile l'evoluzione dell'architettura;
- In IPv4 c'è il problema degli indirizzi a 32 bit, troppo pochi (anche se la soluzione l'ha dichiarata l' IPv6).

In conclusione:

- OSI è ottimo come modello, mentre i suoi protocolli hanno avuto poco successo;
- TCP/IP è ottima come architettura di rete, ma inutile come modello.

Capitolo 7.3

Un breve cenno a SNA.

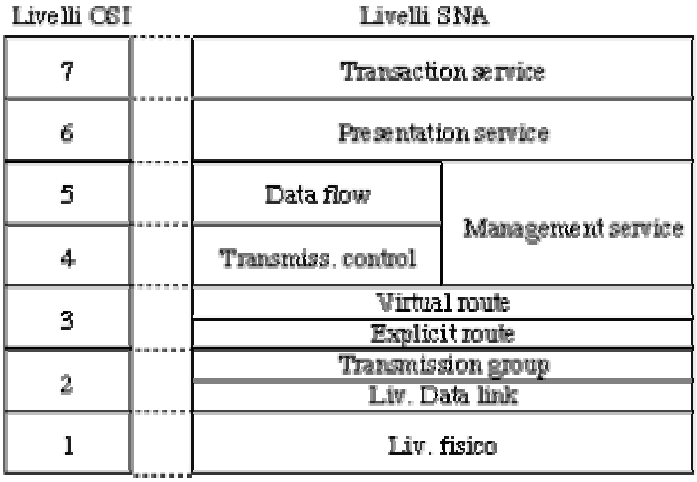
SNA (IBM)

SNA (System Network Architecture) è un'architettura di rete ancora utilizzata nel mondo mainframe, sopratutto nelle grandi aziende dotate di sistemi informativi IBM. Nacque a metà degli anni '70, periodo in cui i sistemi informativi erano basati sull'uso di mainframe e terminali.

Fu pensata per connettere fra loro più mainframe, ai quali dovevano poter essere connessi moltissimi terminali, anche geograficamente lontani.

E' un'architettura estremamente complessa e poco adatta all'attuale impostazione dei sistemi di calcolo (reti locali e applicazioni tipo client-server) per cui IBM sta migrando verso una strategia di networking più ampia, nella quale viene fornito il supporto ad SNA, APPN (successore di SNA: architettura proprietaria ma di dominio pubblico), OSI, TCP/IP ed altri

L'architettura SNA è la seguente:



Capitolo8

ELENCO DEI FONDATORI DEGLI STANDARD NEL MONDO:

PTT (Post, Telephone and Telegraph): amministrazione statale che gestisce i servizi trasmissivi (in Italia è il Ministero delle Poste);

CCITT (Comité Consultatif International de Telegraphie et Telephonie): organismo internazionale che emette le specifiche tecniche che devono essere adottate dalle PTT. E' entrato da poco a far parte dell'**ITU (International Telecomunication Union)**;

ISO (International Standard Organization): il principale ente di standardizzazione internazionale, che si occupa fra l'altro anche di reti;

ANSI (American National Standards Institution): rappresentante USA nell' ISO;

UNINFO: rappresentante italiano, per le reti, nell'ISO;

IEEE (Institute of Electrical and Electronic Engineers): organizzazione professionale mondiale degli ingegneri elettrici ed elettronici; ha gruppi di standardizzazione sulle reti;

IRTF (Internet Research Task Force): comitato rivolto agli aspetti di ricerca a lungo termine in merito alla rete Internet;

IETF (Internet Engineering Task Force): comitato rivolto agli aspetti di "ingegnerizzazione" a breve termine della rete Internet;

IAB (Internet Architecture Board): comitato che prende le decisioni finali su nuovi standard da adottare per Internet, di solito proposti da IETF o IRTF.

Capitolo 9
Streaming e RTP

In alcuni casi come:

- la trasmissione contemporanea di più flussi di dati real-time;
- la sincronizzazione fra più flussi che, in alcuni casi, corrispondono a flussi di media diversi;
- la necessità di gestire più partecipanti nella stessa sessione;

sono richieste a cui i protocolli di trasporto classici TCP e UDP non possono dare una risposta soddisfacente.

Nasce quindi l'esigenza di un protocollo specifico che estenda quelli preesistenti con l'aggiunta di funzionalità adatte al trasporto di flussi multimediali real-time.

Ripetiamo i concetti riguardanti i protocolli "classici":

TCP

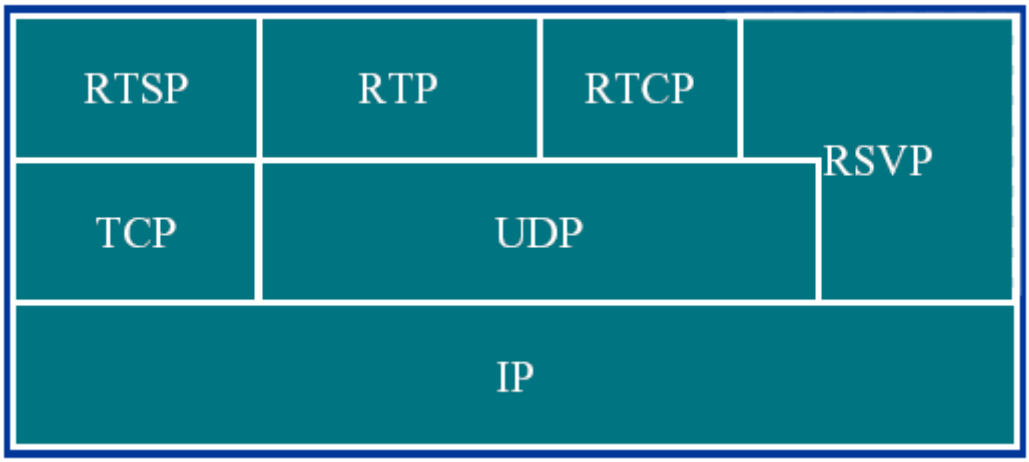
Garantisce un traffico affidabile sulla rete (tutti i pacchetti arrivano e sono nell'ordine corretto) attraverso controlli di congestione e ritrasmissione.
E' utilizzato tipicamente per trasferire dati il cui utilizzo avviene al termine della ricezione: file, posta ecc.
Durante la trasmissione viene utilizzata tutta la banda disponibile.
Non è adatto al multicast in quanto è un protocollo connesso per il quale i meccanismi di ack e ritrasmissione dovrebbero gestire individualmente le connessioni con tutti i numerosi destinatari della trasmissione.
Quando utilizzato per lo streaming, permette in realtà un "progressive download".

UDP

E' un protocollo connectionless, dove cioè non viene stabilita una connessione tra server e client (come invece avviene in TCP) ed è perciò adatto, per la sua leggerezza, al traffico realtime e per contenuti già fruibili mentre vengono ricevuti (streaming).

Non garantisce che la trasmissione avvenga in un certo intervallo di tempo e nemmeno che il pacchetto giunga a destinazione.
Non si effettuano ritrasmissioni.
Per il trasferimento non viene usata tutta la banda disponibile ma si ha un controllo del bitrate a livello di applicazione.

Per dati real-time l'affidabilità non è importante in quanto i tempi di consegna rendono le tecniche di ritrasmissione del TCP deleterie; ad esempio, in caso di congestione della rete, le tecniche del TCP potrebbero compromettere ulteriormente la situazione e rivelarsi immotivate data la presenza pesante del ritardo accumulatosi sulla rete.
Al protocollo di trasmissione si richiede che renda possibile la gestione dei differenti ritardi di trasmissione dei pacchetti nonché il riordino degli stessi, ma si cerca anche di fargli tollerare la perdita di alcuni permettendo però la riproduzione del flusso nel miglior modo possibile anche se incompleto.



Una volta scartato il TCP a causa della sua “accuratezza” nel trattare i flussi, l protocollo preesistente più adatto e su cui appoggiarsi è pertanto l'UDP (RTP è l’estensione perfezionata dell’UDP).

Capitolo 9.1

RTP – Real Time Protocol

IETF Proposed Standard:

RFC 3550 (sostituisce RFC1889), RTP: A Transport Protocol for Real-Time Applications
http://www.ietf.org/rfc/rfc3550.txt

RFC 3551 (sostituisce RFC1890), RTP Profile for Audio and Video Conferences with Minimal Control
http://www.ietf.org/rfc/rfc3551.txt

Per maggiori informazioni su RTP/RTCP:

-"About RTP and the Audio-Video Transport Working Group", Henning Schulzrinne http://www.cs.columbia.edu/~hgs/rtp/

-IETF Audio/Visual Transport Charter http://www.ietf.org/html.charters/avt-charter.html

RFC è' il formato di trasmissione standard per le applicazioni multimediali realtime in Internet.

In pratica si occupa del trasporto "end-to-end" di dati audio/video che necessitano di essere consegnati in tempo reale, inoltre è adatto sia a trasmissioni unicast che multicast.

E' affiancato da un protocollo di controllo detto RTCP (definito all'interno dello stesso standard RTP) ma il cui utilizzo non è obbligatorio ma solo di supporto al trasporto dati vero e proprio e per questo non sempre implementato nelle applicazioni.

Altri protocolli strettamente correlati all'RTP sono RTSP e RSVP.

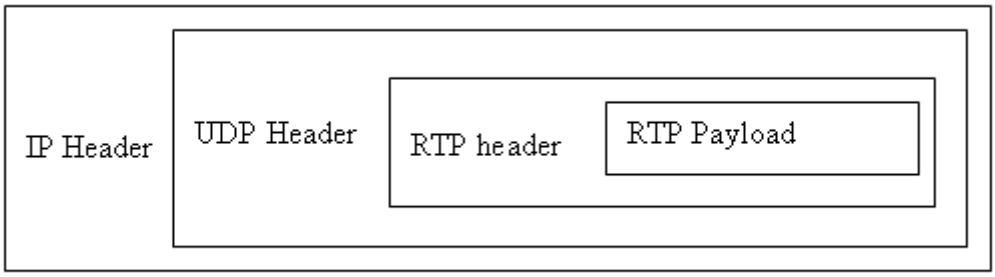
Nel modello OSI si colloca tra il livello 4 (transport) e il livello 3 (network), appoggiandosi ad UDP.

Anche se ufficialmente si tratta di un protocollo di trasporto, è spesso classificato al livello di applicazione dato che lo sviluppatore deve implementare l'RTP nell'applicazione stessa per gestire l'incapsulamento dei pacchetti RTP nelle header dell'UDP.

Quindi, sia in trasmissione che in ricezione, i pacchetti RTP sono visti dall'applicazione attraverso l'interfaccia UDP;

il programmatore deve scrivere il codice in grado di estrarre dai pacchetti UDP le informazioni aggiuntive dell'RTP e per la successiva decodifica dei dati multimediali.

Vediamo ora una Header + Payload di RTP:



Il compito fondamentale del programma client è di ricostruire la sequenza esatta ed individuare eventuali perdite sfruttando le informazioni contenute nell'intestazione di pacchetto.

Per ogni sessione viene trasmesso un unico flusso multimediale;

Sull' IP del destinatario dovranno essere utilizzate due porte una assegnata al flusso video e una per quello audio.
Una delle ragioni di questa separazione riguarda la maggior flessibilità data dal fatto di permettere ai partecipanti di ricevere, su richiesta, un solo flusso di dati.

Questa limitazione non è però rigida in quanto ad es. nella RFC 2250 (http://www.ietf.org/rfc/rfc2250.txt) si stabilisce come incapsulare pacchetti TS MPEG2 (multiplex di più flussi elementari) in pacchetti RTP per trasmettere tale flusso in un'unica sessione.

Nota: Transport Stream (TS)

Il Transport Stream (TS) rappresenta le caratteristiche di robustezza agli errori e alla gestione della multiplazione, caratteristiche fondamentali per trasmissioni di ogni genere (soprattutto usato per quelle di tipo satellitare)..

Il TS è lo stream complessivo che trasporta tutti i dati dei vari programmi trasmessi da un certo operatore.

Per le sua struttura riesce a garantire il minimo carico computazionale necessario per:

-estrarre e decodificare anche solo parte dei dati dal flusso complessivo.

-estrarre i pacchetti di uno o più programmi, anche da TS differenti e reinserirli in un TS nuovo

Caratteristiche RTP

L'RTP si appoggia al protocollo UDP e definisce le informazioni addizionali richieste quali ad es. numeri di sequenza e timestamps (I Timestamps vengono usati per impedire ai segmenti duplicati o vecchi la corruzione di un collegamento attivo e quindi permettono, pur non essendo previsto alcun meccanismo per assicurare il tempo di consegna massimo, la ricostruzione dell'esatto ordine con cui il mittente ha spedito i suoi pacchetti, la rivelazione di eventuali perdite e la sincronizzazione dei media trasportati (eliminazione dei jitter)

Nota: Jitter

Che cosa è il jitter?

Con il termine clock in elettronica digitale si intende un segnale estremamente semplice costituito da una sequenza di zeri e di uno alternati, spesso con la durata dello zero uguale a quella dell'uno.

Questo segnale così semplice trasferisce solo un tipo di informazione: una temporizzazione.

In pratica rappresenta una semplicissima misura del tempo, come un metronomo, e come un metronomo è semplicemente utilizzata da circuiti più o meno complessi per cambiare il proprio stato (circuiti sincroni). Un esempio tipico è un orologio al quarzo.

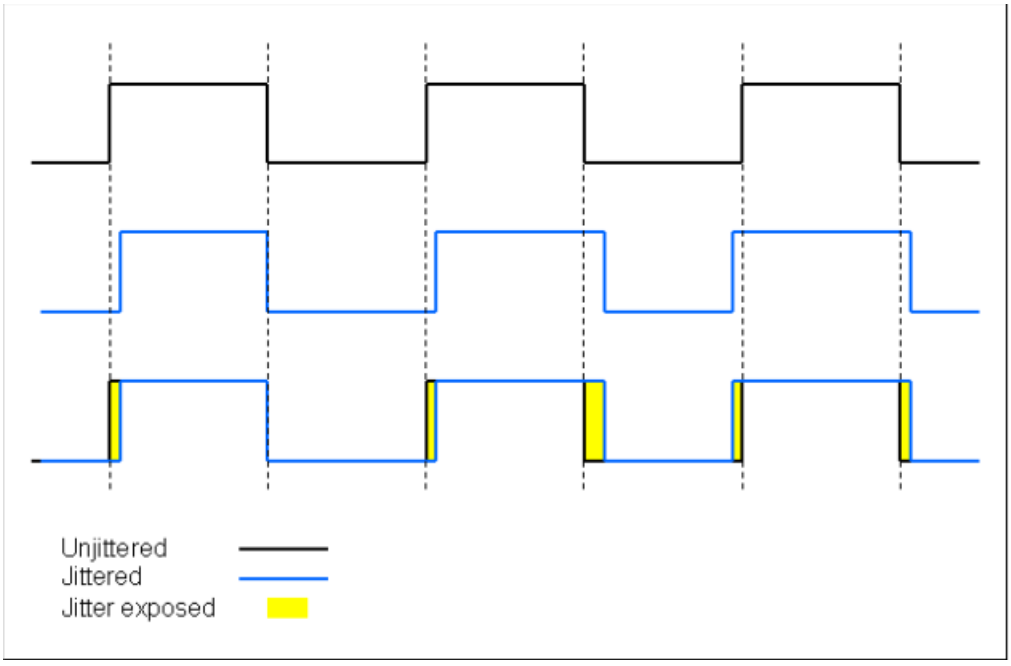
Ogni clock nel mondo reale è affetto da un problema di precisione ed accuratezza. La precisione e l'accuratezza assoluta non esistono nella realtà. E' più facile vedere la differenza fra le due in prospettive temporali diverse, nel lungo e nel breve termine.

Continuando con l'esempio dell'orologio al quarzo, nel lungo termine si è abituati a convivere col fatto che l'orologio corra o resti indietro, cioè che la frequenza media del suo clock non sia esattamente quella nominale: questa è un problema di precisione.

Chiaramente, qualsiasi orologio deve avere una buona precisione per essere utilizzabile. A breve termine, si può star sicuri che ciascun periodo di un secondo misurato dall'orologio è (leggermente) diverso dal periodo precedente o seguente, anche se ciò è impossibile da rilevare per noi.

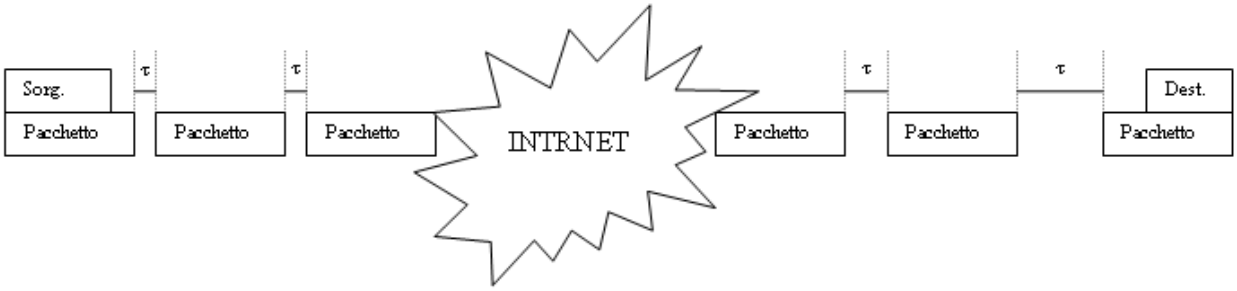
Questo significa che l'orologio non è accurato.

Lo stesso avviene con qualsiasi clock: l'errore di accuratezza da cui ogni transizione ed ogni ciclo del clock sono affetti è il jitter del clock.



Come abbiamo detto, ogni sistema sincrono è controllato da un clock che, come qualsiasi clock nel mondo reale, è affetto da jitter.

Nel nostro caso il jitter potrebbe influire sulla trasmissione dei dati, dato che nello streaming si cerca sempre di ridurre i tempi e di tenere il flusso il più costante possibile.
Esempio jitter nella trasmissione di pacchetti:



Dalla sorgente i pacchetti vengono spediti con un intervallo τ costante, mentre al destinatario arriveranno con un τ non costante a causa dei vari ritardi accumulati passando dalla rete (questo viene visto come Jitter nella spedizione di pacchetti).
Nello streaming, si cerca di tenere sotto controllo questo ritardo usando protocolli in grado di esaminare il ritardo di ogni singolo pacchetto ed avvisare la sorgente in caso di anomalie.

L'utilizzo del protocollo RTP/RTCP è utile per garantire l'interoperabilità fra applicazioni di produttori diversi, poichè trasporta sia i dati che le informazioni necessarie ad identificare il tipo di flusso e la loro codifica.
Questo avviene attraverso la creazione di profili che adattano il protocollo RTP ad una specifica applicazione e che definiscono un identificativo "payload type identifier" nell'header del pacchetto.
E' un valore di 7 bit, quindi sono definibili 128 possibili tipi di payload trasportati.

Esempi di payload video:

Payload Type Number	**Video Format**
26	Motion JPEG
31	H.261
32	MPEG1 video
33	MPEG2 video

L'elenco completo ufficiale presso IANA: http://www.iana.org/assignments/rtp-parameters

Header RTP

Payload Tipe	Sequence number	Timestamp	Synchronization Source Identifier	Miscellaneous

Descrizione delle parti:

Payload Type: 7 bits,
rendendo possibile l'identificazione del contenuto del pacchetto.
Il payload del pacchetto RTP può contenere più campioni audio o frame video o essere viceversa, una parte di un solo fotogramma, a seconda delle codifiche, non ha nemmeno una lunghezza definita, ma anche questa può dipendere dalla codifica usata per i campioni del media dove la quantità dei dati può variare da campione a campione ed è quindi possibile che pacchetti provenienti dalla stessa sorgente contengano payload di differente lunghezza. L'unica limitazione alla lunghezza del pacchetto è data dal protocollo sottostante.

Sequence Number: 16 bits;
per riordinare i pacchetti e controllare un'eventuale perdita.

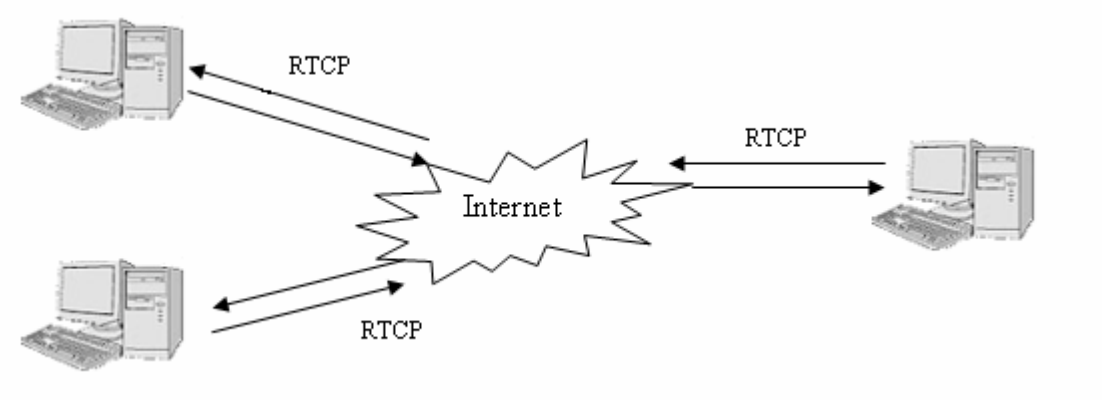
Timestamp: 32 bits;
istante di campionamento del primo byte (audio/video) nel payload del pacchetto.
Servono a eliminare i jitter introdotti dalla rete.
E' derivato da un clock che incrementa linearmente nel tempo per permettere i controlli di sincronizzazione La frequenza del clock dipende dal payload type trasportato (può essere richiesta una precisione più o meno accurata). Pacchetti RTP consecutivi possono avere gli stessi timestamp se sono generati nello stesso istante (ad es. appartenenti allo stesso frame video).

Synchronization Source identifier (SSRC): 32 bits;
identificatore della sorgente dello stream; è assegnato casualmente dalla sorgente.
Se un'entità genera più flussi cioè più stream RTP, ciascuno dovrà essere indicato da un differente SSRC.

Miscellaneous: sono presenti poi altri campi che indicano ad esempio la versione del protocollo (attualmente è la 2, la 0 e 1 sono obsolete), la lista delle sorgenti che contribuiscono al flusso di cui fa parte il pacchetto (CSRC list) nel caso sia stato generato da un RTP mixer, ed anche un campo di extension a volte utilizzato dalle applicazioni ed altri.

Capitolo 9.1.A

RTCP, Real-Time Control Protocol



RTCP è uno speciale protocollo composto da un set di messaggi;
Complementare al RTP (cioè definito nello stesso standard) svolge funzioni di monitoraggio della trasmissione mediante lo scambio fra server e clients di report contenenti le statistiche sulla qualità di trasmissione (QoS feedback) quali numero di pacchetti persi, Round Trip Time (RTT) e Jitter.

Grazie a queste informazioni, le applicazioni riescono a risolvere la congestione della rete cambiando la codifica del flusso per diminuire la banda trasmissiva richiesta.

A ciascuna sessione RTP corrisponde un sessione RTCP (aperta su una porta UDP dedicata)

E' da notare che l'RTP può essere utilizzato anche senza RTCP.

I pacchetti RTCP definiscono 2 tipi di report e i loro rispettivi pacchetti:

1. SR = sender reports: statistiche e informazioni sulla sorgente;
2. RR = reception reports: statistiche del client;

Esistono poi i pacchetti:

SDES: Source Description (per l'identificazione della sorgente di un flusso di dati (un SSRC))
BYE : fine della partecipazione a una trasmissione
APP : funzioni specifiche dell'applicazione

Capitolo 9.1.B

RTSP (Real-Time Streaming Protocol)

E’ un protocollo che nel modello OSI si colloca al livello applicazione (non gestisce la trasmissione dei dati di cui si occupano i protocolli di livello inferiore: RTP e RTCP).

Nello specifico, le sue informazioni di controllo sono trasportate indifferentemente o da TCP o da UDP a seconda delle impostazioni con un collegamento su una porta separata (usa 2 porte separate, 1 TCP e 1 UDP).
Stabilisce e controlla sessioni RTP fornendo comandi per la gestione della riproduzione dello stream:start, play, pause, resume...

E un protocollo testuale molto simile all'HTTP.

Descrive le caratteristiche di un flusso attraverso il protocollo SDP (session decription protocol)

Capitolo 9.1.C

RSVP (Resource ReSerVation Protocol)

L'RTP non si occupa del problema dell'allocazione delle risorse (banda e buffers) o del controllo della qualità di servizio ;

Per queste funzioni si appoggia a uno specifico protocollo di resource reservation: RSVP.

Nel modello OSI si colloca a livello transport.

Garantisce un certo specifico QoS (Quality of Session) nella tratta fra server e client.

Dopo questo elenco generale sui vari protocolli usati nella trasmissione dei dati, passiamo alla parte finale della tesi: lo Streaming.

Capitolo 9.2
Streaming

Con il termine streaming si indica un metodo di trasmissione di file audiovisivi in tempo reale su Internet.

I file streaming sono immediatamente fruibili on line dall'utente senza la necessità di scaricarli prima sul PC, simulando così la trasmissione di programmi radiofonici e televisivi.

Questo metodo di trasmissione istantanea di materiale multimediale può essere sia un flusso live, che un flusso file.

Mentre il primo permette di diffondere all'utenza una trasmissione in diretta (Web Radio), il secondo converte al volo un file in un formato di dimensioni adatte alla connessione del richiedente, per poi successivamente inviarlo a destinazione.

Di questo tipo sono i flussi streaming di Real Video e Real Audio, Windows Media Player, QuickTime.

La distribuzione dei dati di streaming può essere Multicast, Unicast e HTTP.

Esistono tre tipi di Streaming (i più importanti):

1. Streaming in differita;
2. Streaming Live con stream basato su Playlist;
3. Streaming Live con stream ricevuto da fonte diretta (diretta di un evento live);

Nel dettaglio:

Capitolo 9.2.A

1 - Streaming in differita

Lo streaming in differita è il più semplice da implementare poichè è necessaria la sola pubblicazione dei filmati/file sonori per poi richiamarli o dalla pagina Web o direttamente da WindowsMediaPlayer.

Come:

Al momento dell'attivazione, riceverete una Username, una password ed un'IP, dopo di che sarà sufficiente pubblicare i files desiderati all'interno di uno spazio della pagina da cui far partire lo stream utilizzando un Client FTP.

N.B.: L'FTP, acronimo di File Transfer Protocol (protocollo di trasferimento file), è un servizio che fornisce gli elementi fondamentali per la condivisione di file tra host.

Gli obiettivi dell'FTP sono:

1) promuovere la condivisione di file (programmi o dati);

2) incoraggiare l'uso indiretto o implicito (tramite programma) di computer remoti;

3) salvaguardare l'utente al variare dei sistemi di stoccaggio file, tra un host e l'altro;

4) trasferire dati in maniera affidabile ed efficiente.

Capitolo 9.2.B

2 - Streaming Live con stream basato su Playlist

Lo streaming LIVE con Playlist è un ottimo compromesso tra LIVE puro e la differita, dove si può pubblicare una Playlist (l'elenco dei titoli che verranno poi pubblicati in rete) e, una volta avviati sul server (usando un'apposita Utility nel pannello di controllo) verranno visualizzati come si trattasse di una trasmissione televisiva ma differita, nel senso che i contenuti saranno pre-registrati.

Ovviamente chi entra ad evento già iniziato non parte dall'inizio, ma dal punto dove il server è arrivato a visualizzare (in pratica come una normale TV dove l'emittente invece di dare l'evento LIVE lo da in differita).

Come:

Al momento dell'attivazione, riceverete una Username, una password ed un'IP, dopo di questo è sufficiente pubblicare i propri files (compresa la Playlist) all'interno di questo spazio semplicemente utilizzando un Client FTP.

Capitolo 9.2.C

3 - Streaming Live con stream ricevuto da fonte diretta (diretta di un evento live)

Lo Streaming Live con stream ricevuto da fonte diretta (diretta di un evento live) è la forma più conosciuta e diffusa di streaming. Basti pensare alle partite di calcio in diretta sulla rete, eventi sportivi vari e (ultimamente) anche la visione di film. Il tutto avviene tra client e server passando da un canale dotato di Encoder.

Per effettuare questa sessione LIVE è necessario che il Server sia a conoscenza dell'IP di trasmissione del client con l'eventuale Encoder, che deve essere comunicato usando l'apposita Utility del pannello di controllo di una qualsiasi interfaccia presente nei lettori di filmati.

Inoltre è necessario che la connessione sia diretta sulla porta (socket) 1755 del client, infatti si consiglia di avere la connessione dedicata all'evento e non utilizzare Routers, Firewall o Proxy server che in qualche modo possono creare problemi alla connessione tra server e client.

Se non fosse possibile dedicare la connessione all'evento, si possono fare dei tentativi di connessione seguendo le FAQ Microsoft all'indirizzo:

http://www.microsoft.com/windows/windowsmedia/9series/encoder/faq.aspx

ma essendo comunque certi di avere la porta socket 1755 (sia UDP che TCP) libera e dedicata usando un NAT (Network Address Translation ovvero letteralmente Traduzione degli indirizzi di rete) statico in port forwarding sull'IP del client all'interno della rete privata.

I contenuti trasmessi dal client saranno immediatamente visibili/udibili sul Web, senza che l'utente possa salvarsi in locale lo stream oppure interagire con esso (riavvolgimento e/o avanzamento).

Ovviamente chi entra ad evento già iniziato non parte dall'inizio, ma dal punto dove il server e l'encoder sul client è arrivato a visualizzare, in pratica come una normale TV.

Il ritardo tra Client e Server è all'incirca di 5/7 secondi in base alla banda richiesta, dalla qualità della trasmissione e dal tipo di trasmissione (se audio o audio/video).

Al momento dell'attivazione, riceverete una Username, una password ed un'IP che serviranno per instaurare il canale tra Server ed il client con l'encoder.

Per concludere il discorso sullo streaming:

E' orami noto a tutti che le informazioni audio e video richiedono per la trasmissione una notevole quantità di banda, più precisamente, la dimensione della quantità di dati cresce con l'aumentare della qualità richiesta.

Con le normali modalità di funzionamento del Word Wide Web, questo si traduce nel tempo lungo necessario a scaricare un file audio e/o video sul disco del proprio PC, prima di iniziare l'ascolto e/o la visione. Lo spazio disco richiesto può inoltre non essere affatto trascurabile: un clip audiovisivo digitale di 10 minuti con qualità discreta impiegherebbe decine di MB di memoria.

Le tecniche di streaming permettono di ridurre questo tempo ad un piccolo ritardo iniziale (causato dal tempo di connessione, il riempimento parziale dei buffer ecc.), senza richiedere

alcuno spazio su disco locale: il file richiesto viene infatti visualizzato al momento, senza un preventivo download.

La modalità streaming consente inoltre la funzione di contenuti audiovisivi in tempo reale coma ad esempio un canale radiofonico o televisivo. In questo caso non esiste un vero e proprio file, ma piuttosto, un flusso continuo (detto appunto stream) di bit che vengono prodotti codificando in tempo reale la sorgente analogica (l'evento live in questione).

Capitolo 10
Formati video e Codec

Come detto in precedenza, la mole di dati che una sequenza video comporta è molto elevata e quindi sarebbe improponibile trasmettere un filmato senza averlo prima compresso (quindi codificato) con un algoritmo di compressione.

Il compito dei famosissimi CODEC consiste nel trasformare, in fase di acquisizione, le informazioni ricevute in un piccolo file compatto, e in fase di riproduzione, nell'interpretare il codice del file per poterlo visualizzare integro.

Quindi la scelta del formato video, e del codec, è di fondamentale importanza per la buona riuscita di un progetto di streaming.

Tra i formati di compressione più utilizzati, troviamo:

- Microsoft Video for Wimdows (AVI o MVW)
- Apple Quick Time (MOV)
- MPEG o MPEG2
- Real Media (RA o RM)
- Microsoft NetShow (ASF)

Ed ognuno di questi formati supporta vari tipi di codec.

A questo punto manca solo l'analisi delle simulazioni create con NS e lo studio dei relativi dati di output.

Capitolo 11
ESEMPI DI SCRIPT

Esempio 1

In questo esempio, vedremo le caratteristiche principali del comportamento del protocollo TCP in caso di congestioni presenti sulla rete.

In particolare, si potrà osservare come il traffico generato dal protocollo UDP abbia la precedenza nell'utilizzare tutte le risorse disponibili dalla rete anche in condizioni critiche, senza badare alla perdita di dati o a eventuali congestioni di rete.

Su questo principio si basa il concetto di "flusso di dati" o streaming, ciò che importa non è la qualità del dato, ma la quantità.

SCRIPT

Premessa:

Il protocollo UDP genera una sorgente di traffico (nodo1) (immaginiamo un traffico di streaming, dato che l'RTP si comporta esattamente come UDP) che converge su di un nodo, chiamato nodo 2.

Il protocollo TCP genera una sorgente di traffico (nodo 0) (in questo caso il protocollo usato è l'FTP usato soprattutto per gli Upload e per i Download, l'ho usato per rendere più credibile la simulazione) che converge sullo stesso nodo 2 su cui converge UDP.

Dal nodo 2 al nodo 3 il link è stato ristretto in modo da impedire la spedizione simultanea dei due traffici e quindi di creare una congestione sulla rete.

#creo una simulazione

set ns [new Simulator]

#creo il tracciato NAM e il tracciato della simulazione

set nf [open prest1.nam w]

$ns namtrace-all $nf

```
set tf [open prest1.txt w]
$ns trace-all $tf

#settaggio delle variabili e del trace dei monitoraggio
set q12 [open mon12.txt w]
set q02 [open mon02.txt w]
set q23 [open mon23.txt w]

#Chiamata della procedura FINISH
proc finish {} {
    global ns tf q12 q02 q23 nf
    $ns flush-trace
    #chiusura del tracciato NAM
    close $nf
    #chiusura del tracciato della simulazione
    close $tf
    #chiusura dei tracciati di monitoraggio
    close $q12
    close $q02
    close $q23
    #esecuzione automatica del file.NAM leggendo il tracciato della simulazione
    exec nam prest1.nam &
    exit 0
}
#creazione di 4 nodi
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```

```
set n3 [$ns node]

#dichiarazione dei 2 colori per i flussi TCP e UDP all'interno del NAM
$ns color 1 Blue
$ns color 2 Red

#Creazione dei link
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 5ms DropTail
$ns duplex-link $n2 $n3 1Mb 20ms DropTail

#Monitoraggio delle code tra il link 1 e 2
$ns duplex-link-op $n1 $n2 queuePos 0.5
set mon1_2 [$ns monitor-queue $n1 $n2 [$ns get-ns-traceall]]

#Monitoraggio delle code tra il link 0 e 2
$ns duplex-link-op $n0 $n2 queuePos 0.5
set mon0_2 [$ns monitor-queue $n0 $n2 [$ns get-ns-traceall]]

#Monitoraggio delle code tra il link 2 e 3
$ns duplex-link-op $n2 $n3 queuePos 0.5
set mon2_3 [$ns monitor-queue $n2 $n3 [$ns get-ns-traceall]]

#punto di contollo della coda tra i 2 nodi (n2-n3)(solo per il NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

#creazione delle procedure per la registrazione del passaggio dei pacchetti su i link

```
#monitoraggio traffico UDP da n1 a n2
proc record1 {} {
        global q12 mon1_2
        set ns [Simulator instance]
        set time 0.1
        set now [$ns now]
        set pacchetti [$mon1_2 set parrivals_]
        puts $q12 "$pacchetti"
        $ns at [expr $now + $time] "record1"
}
#monitoraggio traffico TCP da n0 a n2
proc record2 {} {
        global q02 mon0_2
        set ns [Simulator instance]
        set time 0.1
        set now [$ns now]
        set pacchetti  [$mon0_2 set parrivals_]
        puts $q02 "$pacchetti"
        $ns at [expr $now + $time] "record2"
}
proc record3 {} {
        global q23 mon2_3
        set ns [Simulator instance]
        set time 0.1
        set now [$ns now]
        set pacchetti [$mon2_3 set parrivals_]
        puts $q23 "$pacchetti"
```

```
        $ns at [expr $now + $time] "record3"
}

#disposizione dei nodi
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right

#Creazione della sorgente TCP da attaccare al nodo n0
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp

#Creazione del TCP agent ma in modalità Sink con solo invio di pacchetti ACK
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Creazione di una applicazione di tipo FTP (File Transfer Protocol) sulla connessione TCP
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Creazione di una sorgente UDP sul nodo n1
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
$udp set fid_ 2
```

#creazione di un agente null

set null [new Agent/Null]

$ns attach-agent $n3 $null

#connessione delle applicazioni

$ns connect $udp $null

#creo del traffico di tipo CBR(costant bit rate) dalla sorgente UDP

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set type_ CBR

$cbr set packet_size_ 1000

$cbr set rate_ 1mb

$cbr set random_ false

#calendario degli eventi inizio la trasmissione dei pacchetti CBR, attivazione del FTP agent(down/up load) e inizio procedure di monitoraggio

$ns at 0.1 "record1"

$ns at 0.1 "record2"

$ns at 0.1 "record3"

$ns at 0.2 "$cbr start"

$ns at 0.3 "$ftp start"

$ns at 4.5 "$cbr stop"

$ns at 4.5 "$ftp stop"

#Chiamata della procedura finish dopo 4 secondi dall'inizio della simulazione

$ns at 5.0 "finish"

#visualizzazione delle dimensione dei pacchetti e intervallo di trasmissione

#(ritardo di propagazione)(facoltativo)

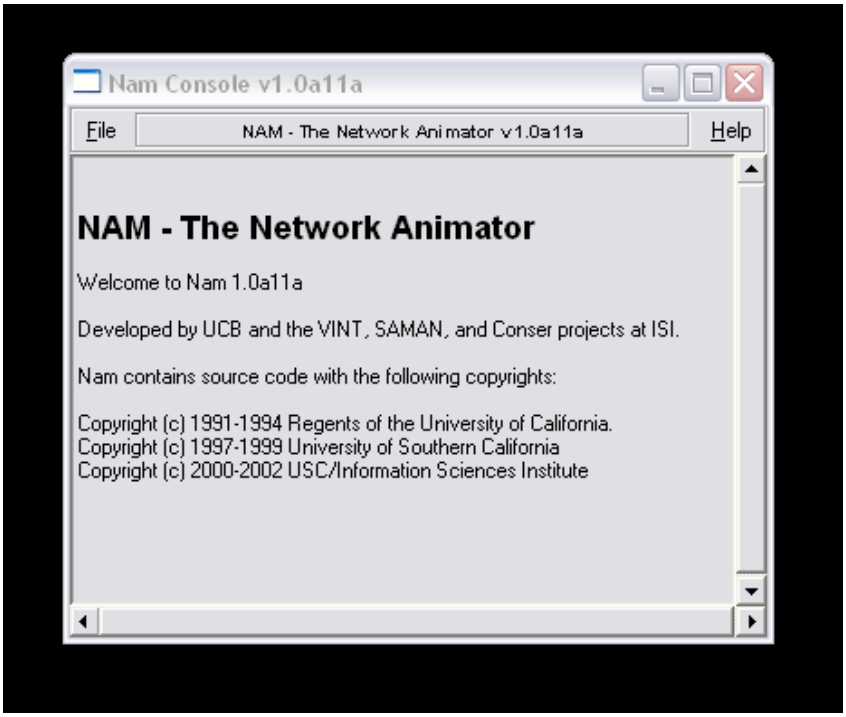
#puts "CBR packet size = [$cbr set packet_size_]"

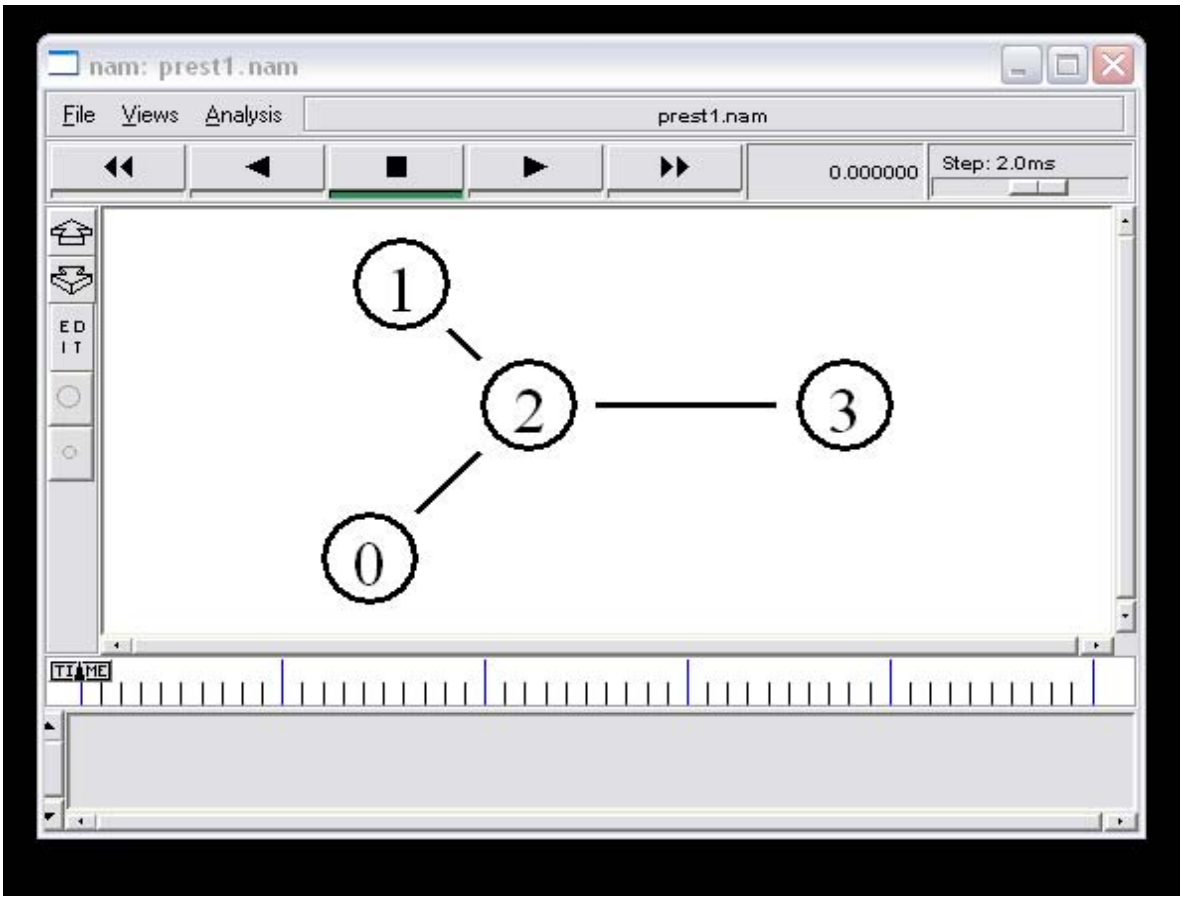
#puts "CBR interval = [$cbr set interval_]"

#Inizio simulazione

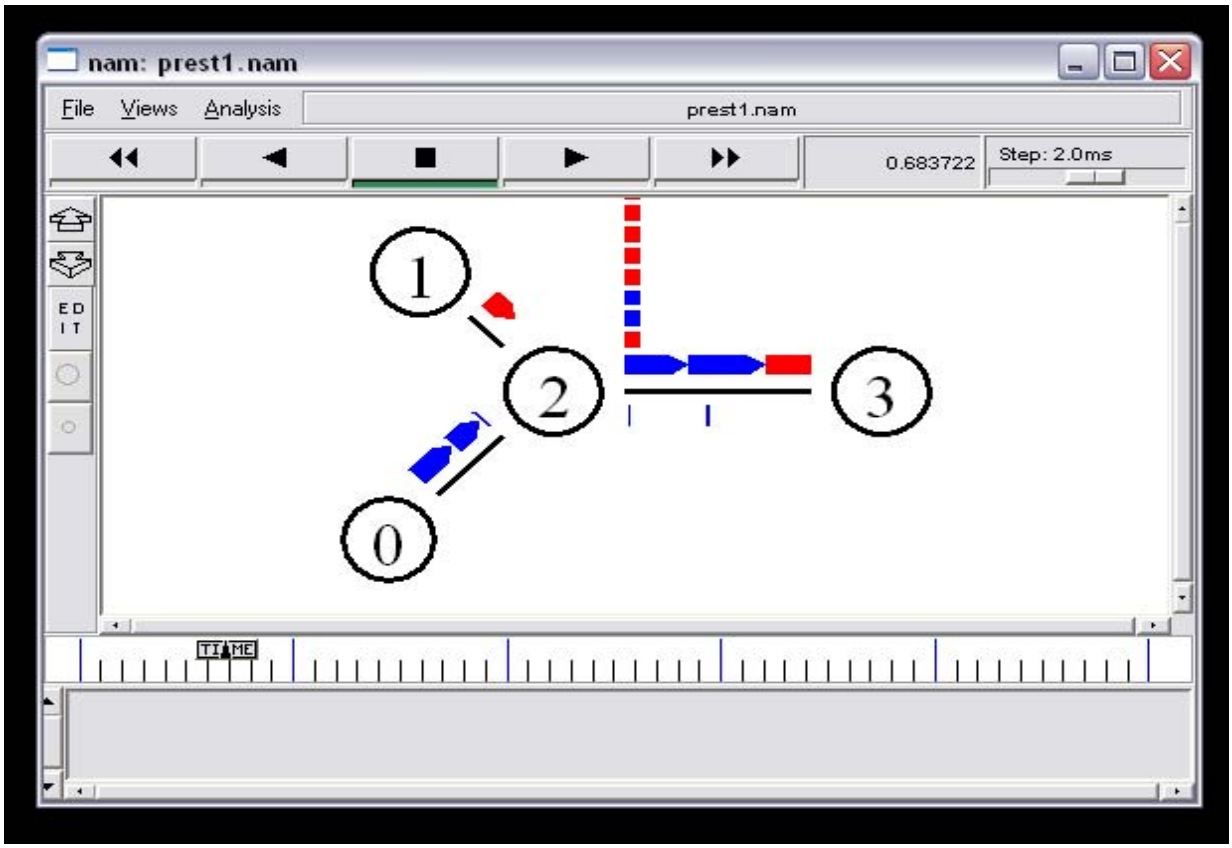
$ns run

Quello che vi si presenterà davanti dopo la compilazione sarà questo:





Ciccando con il tasto sinistro del mouse sul pulsante “START” dell’interfaccia, si darà inizio alla simulazione vera e propria:



In rosso il traffico UDP, in blu quello TCP.

Dopo la compilazione dello script, verranno creati anche 3 files di testo appartenenti allo stato di monitoraggio delle code e 1 file chiamato prest1.txt che riporta il flush trace della simulazione.

I file di monitoraggio servono per estrarre informazioni riguardanti lo stato delle code e quindi la quantità di pacchetti passanti dai nodi monitorati.

mon23.txt file che controlla il traffico sul link tra il nodo 2 e il nodo3

0	330	606
0	342	608
12	355	608
27	368	608
44	381	608
64	393	610
81	406	
105	418	
122	431	
142	443	
163	456	
175	468	
195	481	
212	493	
229	506	
242	518	
255	531	
267	543	
280	556	
292	568	
305	581	
317	593	

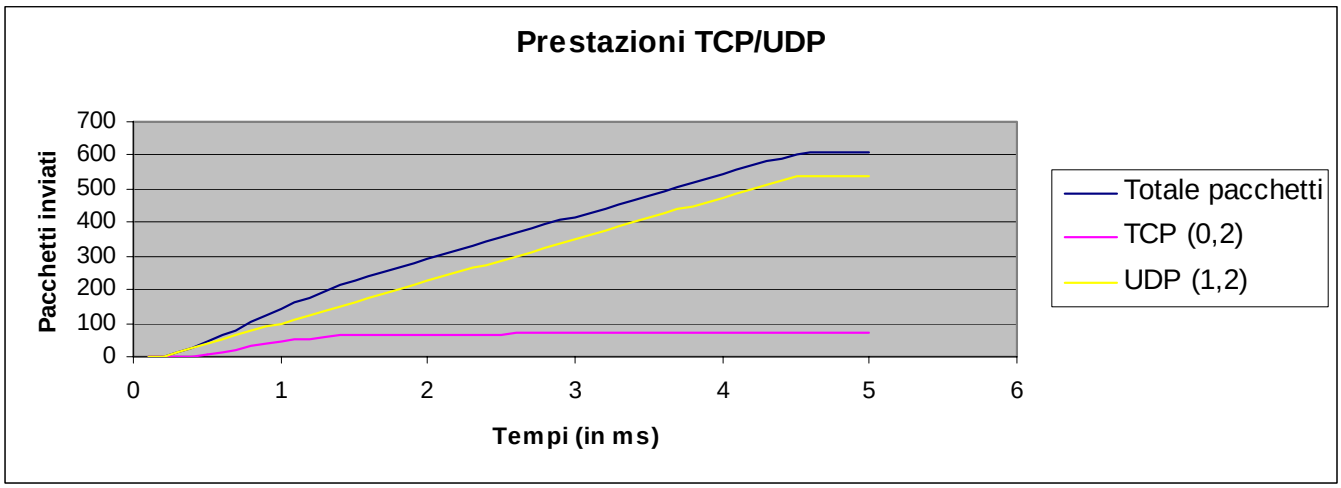
Il monitor mi riporta il totale dei file arrivati al nodo 3 (ovviamente sul file creato da NS i dati saranno disposti su di un'unica colonna).

Monitorando i 2 flussi delle sorgenti e il flusso del destinatario, si può avere un'idea di quanti pacchetti manda TCP e quanti UDP sul totale dei pacchetti ricevuti dal destinatario.

Mon02 riporta i pacchetti totali di TCP.

Mon12 riporta i pacchetti totali di UDP.

Unendo i 3 file .txt e organizzandoli su di grafico, si avrà la seguente situazione:



e come è facile da vedere, TCP invia molti meno pacchetti di UDP a causa del controllo di congestione della rete.

Per questo motivo, si è scelto di sviluppare lo Streaming partendo dal protocollo con maggior forza nel trasportare grandi quantità di dati, pur sacrificando molti aspetti riguardanti l'affidabilità del trasporto.

FILE USATI:

prest1.tcl = file contenente lo script;

prest1.nam = file generato dalla compilazione dello script e usato per la simulazione con il NAM;

prest1.txt = file contenente il flusso dei pacchetti, composto da 14 campi spiegati nel dettaglio dal manuale di NS;

mon12
mon02
mon23

} .txt = generati dalla compilazione e usati per le statistiche e i grafici;

Grafico prestazionale TCPUDP.xls = file di excel usato per raggruppare tutti i dati e creare il grafico;

Esempio 2

Un esempio di protocollo RTP con 2 gruppi multicast, il primo formato dai nodi 0 – 2, l'altro formato dai nodi 3 – 4. Il nodo 1 funge da router tra le due sessioni di trasmissione.

#PROGRAMMA CHE SIMULA DUE SESSIONI DI RTP IN MULTICAST;

#attivazione del multicast routing

set ns [new Simulator -multicast on]

#creazione dei 4 nodi

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

#assegnazione del colore al protocollo di controllo RTCP **(vedi dettagli nella tesi)**

#SR = sender reports: statistiche e informazioni sulla sorgente;

#RR = reception reports: statistiche ed informazioni sul destinatario;

$ns color 30 purple

$ns color 32 green

#creazione del trace flush

set tf [open rtpM.txt w]

$ns trace-all $tf

#creazione del nam flush

set nf [open rtpM.nam w]

$ns namtrace-all $nf

#creazione dei link tra i nodi

$ns duplex-link $n0 $n1 1.5Mb 10ms DropTail

$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail

$ns duplex-link $n1 $n3 1.5Mb 10ms DropTail

$ns duplex-link $n1 $n4 1.5Mb 10ms DropTail

#disposizione forzata dei nodi

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

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

$ns duplex-link-op $n1 $n3 orient right-down

$ns duplex-link-op $n0 $n1 queuePos 0.5

#configurazione del protocollo di multicast (in questo caso DM=dynamic multicast)

set mproto DM

#inserimento in ogni gruppo della capacità di trasmettere in multicast

set mrthandle [$ns mrtproto $mproto {}]

#allocaddr serve per suddividere i 2 gruppi che sono in multicast

set group1 [Node allocaddr]

set group2 [Node allocaddr]

#creazione delle sessioni di multicast con protocollo RTP

set s0 [new Session/RTP]

set s2 [new Session/RTP]

set s3 [new Session/RTP]

set s4 [new Session/RTP]

#dichiarazione del bandwidth di ogni sessione

$s0 session_bw 400kb/s

$s2 session_bw 400kb/s

$s3 session_bw 400kb/s

$s4 session_bw 400kb/s

#attacco i nodi alle sessioni di RTP

$s0 attach-node $n0

$s2 attach-node $n2

$s3 attach-node $n3

$s4 attach-node $n4

#calendario degli eventi

#dichiarazione inizio sessione gruppo1

$ns at 0.4 "$s0 join-group $group1"

$ns at 1.5 "$s0 start"

$ns at 1.6 "$s0 transmit 400kb/s"

$ns at 1.4 "$s2 join-group $group1"

$ns at 2.5 "$s2 start"

$ns at 2.6 "$s2 transmit 400kb/s"

#dichiarazione inizio sessione gruppo2

$ns at 0.4 "$s3 join-group $group2"

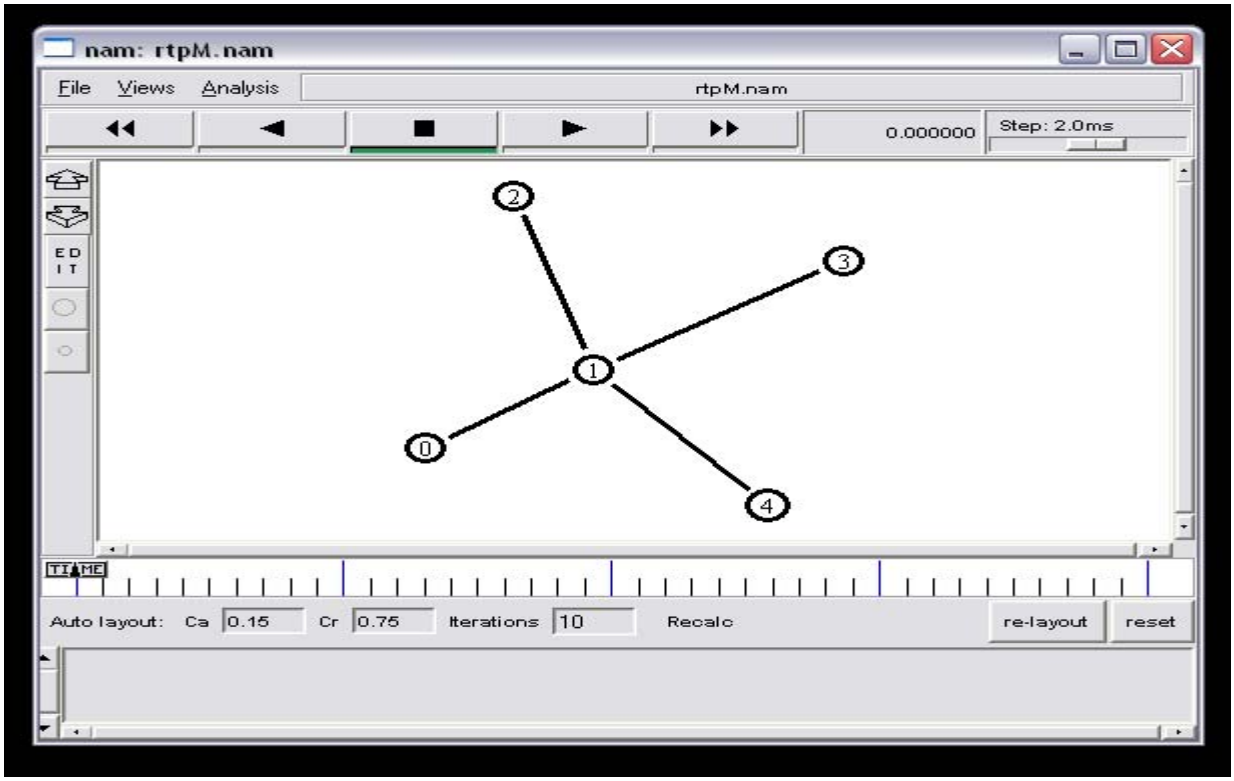
$ns at 1.5 "$s3 start"

$ns at 1.6 "$s3 transmit 400kb/s"

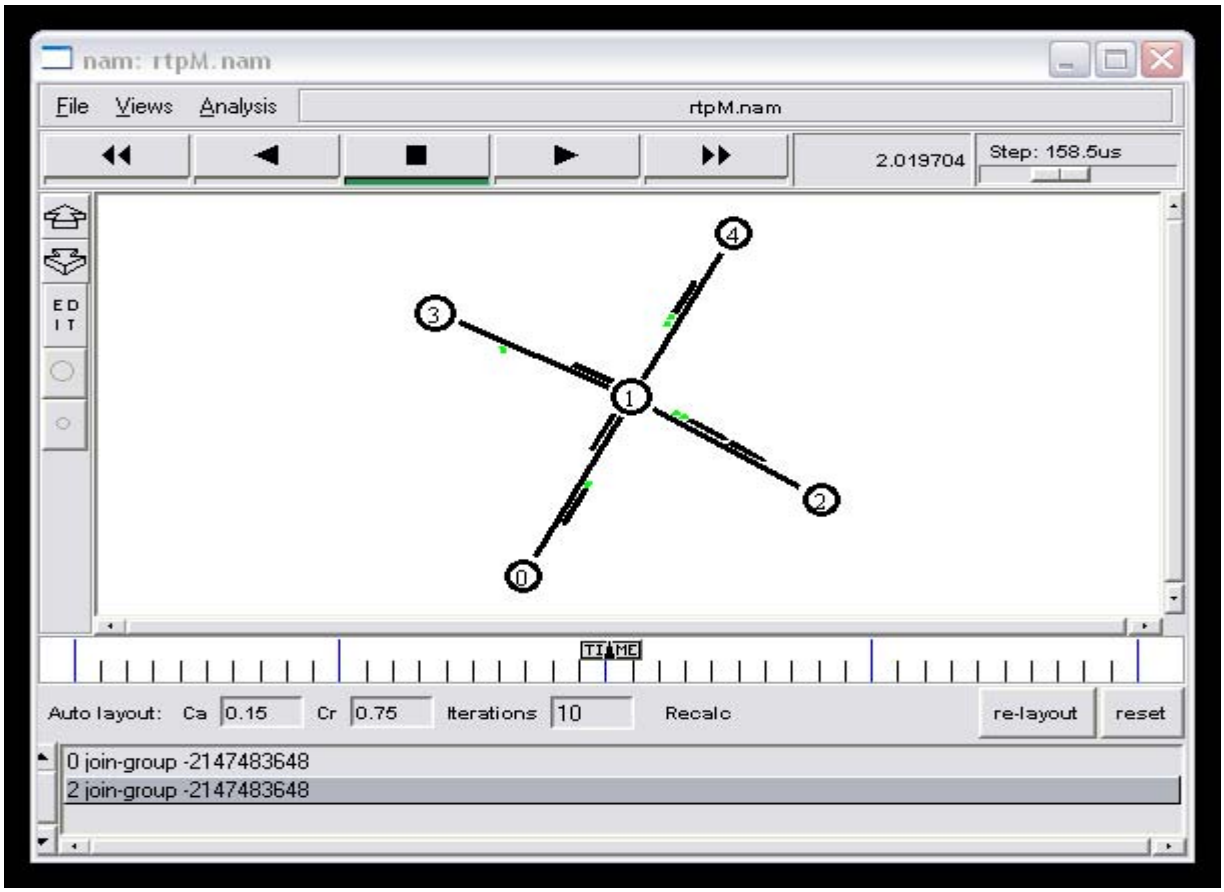
$ns at 1.4 "$s4 join-group $group2"

```
$ns at 2.5 "$s4 start"
$ns at 2.6 "$s4 transmit 400kb/s"

#chiamata della procedura finish
$ns at 4.0 "finish"

proc finish {} {
        global ns tf nf
        $ns flush-trace
        #chiusura di tutti i tracciati
        close $tf
        close $nf

           #esecuzione automatica del NAM e fine della compilazione
        exec nam rtpM.nam &
        exit 0
}
$ns run
```

Prima schermata:



inizio simulazione:



FILE USATI:

rtpMulti.tcl = file contenente lo script;

rtpM.nam = file generato dalla compilazione dello script e usato per la simulazione con il NAM;

rtpM.txt = file contenente il flusso dei pacchetti, composto da 14 campi spiegati nel dettaglio dal manuale di NS;

Bibliografia

Materiale essenziale, manuale ufficiale e FAQ : http://www.isi.edu/nsnam/

Installazione su Linux Red Hat 7.1: http://www.paolocastagna.org/ns/

Tutoria per le prime simulazioni: http://www.isi.edu/nsnam/ns/tutorial/

Compilazione di NS su Windows: http://www.isi.edu/nsnam/ns/ns-win32-build.html

Sito non ufficiale di NS (Italiano):
http://www.di.unito.it/~matteo/DIDATTICA/aa05/RdE/ns_res.html

Enciclopedia libera: http://it.wikipedia.org/wiki/Pagina_principale

Corso di reti di calcolatori dell'università di Pavia:
www.unipv.it/retical/home.html

Protocolli di comunicazione e tecnologie per la trasmissione di dati: http://www.ce.unipr.it/

Corso di reti di calcolatori dell'università di Genova:
http://www.disi.unige.it/person/BelleG/Reti99/

Protocollo RTP e Streaming: http://telemat.die.unifi.it/book/VideoConferencing/4rtp/rtp.htm

Elenco degli acronimi usati nella tesi

DARPA: Defense Advance Research Project Agency (http://www.darpa.mil/);

USC/ISI: University of South California / Information Sciences Institute (http://www.usc.edu/);

Xerox PARC: Xerox (agenzia americana) Palo Alto Research Center (creatore di GUI = Graphic User Interface) (http://www.parc.xerox.com/);

LBNL: Lawrence Berkeley National Laboratory (http://www.lbl.gov/LBNL.html);

VINT: Virtual InterNational Testbed (http://www.isi.edu/nsnam/vint/);

ACIRI: AT&T Cenetr for Internet Research at ICSI (http://www.att.com/welcome.html);

UCB Daedalus: University of California, Berkeley (http://www.berkeley.edu/);

SUN Microsystem: Java & Solaris (http://www.sun.com/);

ISO/OSI: Internation Organization for Standardization / Open System Interconnection (http://www.iso.org/);

TFRC: TCP Friendly Rate Control = protocollo UDP in grado di condividere equamente le risorse con gli altri traffici presenti in rete;

RED: Random Early detection = algoritmo usato per la gestione delle code;

CBQ: Class Based Queueing = divide la banda in porzioni eque per ogni traffico;

SYN/FIN: SYNchronise pachet in TCP (indica il primo pacchetto della connessione) / FINaly pachet in TCP (indica l'ultimo pacchetto della connessione);

FTP: File Transfer Protocol = protocollo usato per Upload e Download;

NAT: Network Address Translation = traduttore di indirizzi IP in nomi.