Calcolo del throughtput stop and wait. C’è sul compito 100%.

Appunti 01/03/2022

Che cos’è Internet?

Esistono due modi di rispondere a questa domanda. La prima è

descrivere gli “ingranaggi” di Internet, ossia i componenti di base hardware e software

che la compongono. Un altro metodo consiste nel descrivere Internet in termini di infrastruttura di rete.

# Gli “ingranaggi” di Internet

Internet è una rete di calcolatori che interconnette miliardi di dispositivi di calcolo in tutto il mondo. Tutti questi dispositivi sono detti **host** (ospiti) o **sistemi periferici** (end system).

Questi ultimi sono connessi tra loro tramite una **rete di collegamenti** (communication link) e **commutatori di pacchetti** (packet switch).

# Descrizione dei servizi

Internet come abbiamo detto prima può essere visto come un’infrastruttura che fornisce servizi alle applicazioni. Tali applicazioni, dette **applicazioni distribuite**, sono:

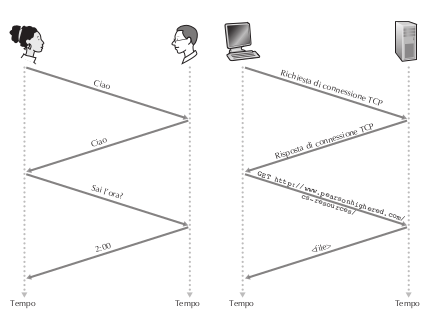
posta elettronica, navigazione sul Web, streaming musica e video, etc..

Chiamate così perché coinvolgono più sistemi periferici che si scambiano dati.

I sistemi periferici collegati a Internet forniscono una **interfaccia socket** (socket interface) che è un insieme di regole che il programma mittente deve seguire in modo che i dati siano recapitati al programma destinatario.

# Che cos’è un protocollo?

Probabilmente è più facile comprendere il funzionamento di un protocollo di rete considerando dapprima delle analogie:



**Protocolli di rete**

Un protocollo di rete è simile a un “protocollo umano”, a eccezione del fatto che le entità che si scambiano messaggi e intraprendono azioni sono componenti hardware e software di qualche dispositivo (computer, tablet, router, etc..). Qualsiasi attività in Internet che coinvolge due o più entità remote in comunicazione viene governata da un protocollo.

Un Esempio: i protocolli nei router determinano un percorso per il pacchetto dalla sorgente alla destinazione.

**Definizione migliore:**

Un protocollo definisce il formato e l’ordine dei messaggi scambiati tra due o più

entità in comunicazione, così come le azioni intraprese in fase di trasmissione e/o

di ricezione di un messaggio o di un altro evento.

# Ai confini della rete

I sistemi periferici vengono detti **host** in quanto ospitano programmi applicativi quali browser e web server o software di lettura. Talvolta gli host vengono ulteriormente suddivisi in due categorie:

* **Client**: sono host che richiedono dei servizi (PC, smartphone, etc..);
* **Server**: si occupano di erogare dei servizi e sono macchine più potenti che memorizzano e distribuiscono pagine web o flussi di video.

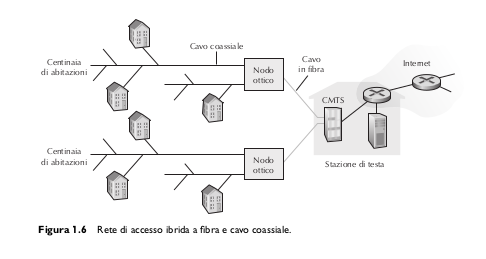
# Le reti di accesso

Le **reti di accesso** (access network), è la rete che connette fisicamente un sistema al suo **edge router** (router di bordo), che è il primo router sul percorso dal sistema d’origine a un qualsiasi altro sistema di destinazione collocato al di fuori della stessa rete di accesso.

**Accesso residenziale: DSL, via cavo, FTTH, dial-up e satellite**

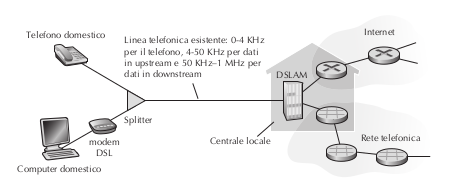
Oggigiorno i due accessi residenziali a larga banda più diffusi sono il **digital subscriber line (DSL)** e quello via cavo.

Mentre la DSL usa le infrastrutture già esistenti della compagnia telefonica locale, l’**accesso a Internet** via cavo utilizza le infrastrutture esistenti della televisione via cavo. Esempio sulla figura sotto: le fibre ottiche connettono la terminazione del cavo a giunzioni a livello di quartiere, dalle quali viene usato il tradizionale cavo coassiale per la distribuzione televisiva per raggiungere le singole case o appartamenti.



L’accesso a Internet via cavo richiede modem speciali, chiamati **cable modem.** Il cable modem è un dispositivo esterno che si connette al PC di casa attraverso una porta Ethernet. I cable modem dividono la rete HFC (Hybrid fiber-coaxial è una rete che combina la fibra ottica e il cavo coassiale) in due canali, uno in downstream (download) e uno in upstream (upload), l’accesso è asimmetrico: naturalmente al canale in downstream vengono allocati tassi di trasmissione più elevati di quello upstream.

Un **accesso residenziale a Internet di tipo DSL** viene fornito dalla stessa compagnia che fornisce anche il servizio di telefonia fissa. Il modem DSL dell’utente usa la linea telefonica esistente per scambiare dati con un digital subscriber line access multiplex (DSLAM) che si trova nella centrale locale. Il modem DSL residenziale converte i dati digitali in toni ad alta frequenza per poterli trasmettere alla centrale locale sul cavo telefonico; tutti i segnali analogici in arrivo dalle abitazioni vengono riconvertiti in formato digitale nel DSLAM.



**Accesso aziendale(e residenziale): Ethernet e WiFi**

Nelle aziende e nelle università, e sempre di più nelle abitazioni, per collegare i sistemi periferici al router di bordo si utilizza una **rete locale** (LAN). Ethernet utilizza un doppino di rame intrecciato per collegare numerosi sistemi periferici tra loro e connetterli a uno switch Ethernet, che a sua volta viene connesso ad Internet.

L’accesso tramite Ethernet ha generalmente velocità di 100 Mbps, che possono arrivare fino a 1 o 10 Gbps.

Mentre invece l’accesso da WiFi provvede una velocità di 11, 54, 450 Mbps.

Sempre più utenti accedono a Internet via wireless da computer portatili, smartphone, table, etc.... In una LAN wireless gli utenti trasmettono e ricevono pacchetti entro un raggio di poche decine di metri da e verso un access point wireless connesso da una rete aziendale, che include una rete Ethernet cablata, a sua volta connessa ad Internet.

**Accesso wireless su scala geografica: 3G e LTE**

I dispositivi iPhone e Android vengono sempre più frequentemente usati per inviare messaggi/foto, etc... mentre si è fuori; tali dispositivi utilizzano la stessa infrastruttura wireless usata dalla telefonia cellulare per inviare/ricevere pacchetti tramite una stazione base. A differenza del WiFi, l’utente può trovarsi a poche decine di chilometri dalla stazione base, invece che a poche decine di metri.

**Mezzi transitivi**

Quando un bit viaggia da un sistema periferico ad un’altro, attraversando una serie di collegamenti e router, viene ritrasmesso più volte, quando passa dalla sorgente alla destinazione, passa per una serie di coppie trasmettitore-ricevitore, propagandosi dall’uno all’altro sotto forma di onda elettromagnetica o impulso ottico attraverso **mezzo fisico**.

Tra gli esempi di mezzi fisici abbiamo: il doppino intrecciato, il cavo coassiale, la fibra ottica multimodale, etc..

I mezzi fisici ricadono in due categorie: i **mezzi vincolati** (guided media) e quelle **non vincolati** (unguided media), la differenza tra i due è che nei primi, le onde vengono contenute in un mezzo fisico; mentre nel caso dei secondi, le onde si propagano nell’atmosfera o nello spazio esterno.

**Doppino di rame intrecciato**

Il doppino intrecciato è costituito da due fili di rame distinti, disposti a spirale regolare è avvolto da uno schermo protettivo a formare un cavo. Una coppia di fili costituisce un singolo collegamento di comunicazione. Il **doppino intrecciato non schermato** (UTP, unshielded twisted pair) viene comunemente utilizzato per le reti all’interno di un edificio, cioè le LAN (trasmissione tra 10 Mbps e 10 Gbps).

**Cavo coassiale**

Anche il cavo coassiale è costituito da due conduttori di rame, ma questi sono concentrici anziché paralleli. Il suo impiego è piuttosto comune nei sistemi televisivi via cavo, questi sistemi sono stati di recente abbinati a modem via cavo per fornire agli utenti residenziali accesso a Internet a velocità di decine di Mbps. Inoltre, il cavo coassiale può essere utilizzato come **mezzo condiviso** vincolato (più sistemi periferici possono essere connessi direttamente via cavo e tutti ricevono quanto inviato da altri sistemi periferici).

**Fibra Ottica**

La fibra ottica è un mezzo sottile e flessibili che conduce impulsi di luce, ciascuno dei quali rappresenta un bit. Una singola fibra ottica può supportare enormi velocità trasmissive, fino a decine o centinaia di gigabit al secondo. Tale mezzo è immune all’interferenza elettromagnetica. La maggior parte delle reti telefoniche a lungo raggio degli Stati Uniti e di altri Paesi impiega esclusivamente fibre ottiche.

**Canali radio terrestri**

I canali radio trasportano segnali all’interno dello spettro elettromagnetico. Non richiede l’installazione fisica dei cavi, è in grado di attraversare le pareti, fornisce connettività agli utenti mobile e trasporta segnale da lunghe distanze.

L’ambiente determina la perdita di segnale e può essere causata dalla distanza (path loss), dall’attraversamento di ostacoli (shadow falling), dalla riflessione sulle superfici (multipath fading).

Di norma i canali radio terrestri si dividono in tre gruppi: quelli che operano a piccole distanze (1-2 metri) [LAN wireless usate da cuffie, tastiere etc...], quelli che operano i aree locali (centinaio di metri) ed infine quelli che operano in aree più vaste (decine di chilometri) [accesso cellulare ad Internet].

**Canali radio satellitari**

Un satellite per le comunicazioni collega due o più trasmettitori terrestri a microonde, noti come stazioni a terra (ground station). Il satellite riceve le trasmissioni su una banda di frequenza, rigenera il segnale utilizzando un ripetitore e trasmette segnali su un’altra frequenza. Nelle comunicazioni si usano due tipi di satellite: quelli geostazionari (GEO, geostationary earth orbit) e quelli a bassa quota (LEO, low-earth orbiting).

La grande distanza tra la stazione a terra e il satellite (e ritorno) introduce un ritardo di 280 millisecondi nella propagazione del segnale. Ciò non di meno, i collegamenti via satellite, in grado di operare alla velocità di centinaia di Mbps, vengono spesso utilizzati dove non è presente accesso a Internet via DSL o cavo.

# Il nucleo della rete

Il nucleo della rete viene vista come una maglia di commutatori di pacchetti e collegamenti che interconnettono i sistemi periferici di Internet.

Per spostare i dati in una rete di collegamenti e commutatori esistono due approcci:

* **commutazione di circuito**;
* **commutazione di pacchetto**.

# Commutazione di pacchetto

Le applicazioni scambiano **messaggi** che possono contenere qualsiasi cosa (da un messaggio testuali a un immagine JPEG). La sorgente suddivide i pacchetti lunghi in parti più piccole note come **pacchetti**, questi pacchetti viaggiano attraverso i **commutatori di pacchetti**.

I pacchetti vengono trasmessi su ciascun collegamento a una velocità pari alla velocità totale della trasmissione del collegamento stesso. Quindi, su un sistema periferico o commutatore inviano un pacchetto di L bit su un canale con velocità di R bps, **il tempo di trasmissione** risulta pari a L/R secondi.

**Trasmissione store-and-forward**

La maggior parte dei commutatori di pacchetto utilizza questo metodo, ciò significa che il commutatore deve ricevere l’intero pacchetto prima di poterne cominciare a trasmettere sul collegamento in uscita il primo bit.

.  
Per approfondire la trasmissione store-and-forward calcoliamo ora il tempo che intercorre da quando la sorgente inizia a inviare il pacchetto a quando il destinatario lo riceve completamente.

Calcoliamo ora l’intervallo di tempo intercorso da quando la sorgente inizia a inviare il primo pacchetto a quando il destinatario li ha ricevuti tutti e tre. Come prima, al tempo L/R il router inizia a inoltrare il primo pacchetto. Ma sempre al tempo L/R la sorgente inizia a inviare il secondo pacchetto, perché ha appena completato l’invio del primo pacchetto. Quindi al tempo 2L/R il destinatario ha ricevuto il primo pacchetto e il router ha ricevuto il secondo. Allo stesso modo, al tempo 3L/R, il destinatario ha ricevuto i primi due pacchetti e il router ha ricevuto il terzo pacchetto. Infine al tempo 4L/R il destinatario ha ricevuto tutti e tre i pacchetti.

Si consideri ora il caso generale della trasmissione di un pacchetto dalla sorgente alla destinazione su un percorso consistente di N collegamenti ognuno con velocità di trasmissione R (quindi vi sono N – 1 router tra la sorgente e il destinatario).

Applicando lo stesso ragionamento si trova che il ritardo da un capo all’altro (end-to-end) è:

delay → d (end-to-end)=N \* (L / R)

**Ritardi di accodamento e perdita di pacchetti**  
Ogni commutatore ha più collegamenti e per ciascuno di questi, il commutatore mantiene un **buffer di output** (o **coda di output**) per conservare pacchetti che sta per inviare su quel collegamento.

Cosa servono questi buffer?

Un pacchetto in arrivo che debba essere inviato attraverso un collegamento occupato dalla trasmissione di un altro, deve attendere nella coda di output. Quindi inoltre al ritardo derivante dal store-and-forward i pacchetti avranno anche **ritardi di accodamento** nei buffer (tali ritardi variano dal traffico che c’è in rete).

Dato che la memoria del buffer è finita può capitare che un pacchetto si ritrova il buffer pieno di altri e quando questo accade si ha una **perdita di pacchetto** (packet loss): verrà eliminato o uno in coda o quello arrivato.

# Commutazione di circuito

Nelle reti a commutazione di circuito le risorse richieste lungo un percorso per consentire la comunicazione tra sistemi periferici sono riservate per l’intera durata della sessione di comunicazione.  
Le reti telefoniche sono un esempio di reti a commutazione di circuito. Si consideri cosa avviene quando una persona vuole inviare un messaggio ad un altra. Prima che venga digitato invio la rete crea un collegamento tra mittente e destinatario per tutta la durata della comunicazione e quando ciò avviene viene detto circuito (viene garantita la velocità di trasmissione costante).  
  
Appunti 03/03/2022

# CIRCUIT SWITCHING: FDM and TDM

**Multiplexing nelle reti a commutazione di circuito**

Un circuito all’interno di un collegamento è implementato tramite **multiplexing a divisione di frequenza** (FDM) o **multiplexing a divisione di tempo**(TDM).

FDM sta per Frequency Division Multiplexing: il collegamento dedica una banda di frequenza a ciascuna connessione, normalmente questa banda di frequenza ha un **ampiezza di banda**. (per tutta la durata del tempo della connessione)

TDM sta per Time Division Multiplexing: il tempo viene diviso in intervalli di durata fissa che a loro volta sono ripartiti in un numero fisso di slots; ad ogni collegamento viene dedicata una proprio banda con un proprio massimo ratio di banda durante il tempo dei slot.

# PACKET SWITCHING versus CIRCUIT SWITCHING

**Confronto tra commutazione di pacchetto e commutazione di circuito**

Il Packet Switching è sempre vincente rispetto il circuit switching?

I denigratori della commutazione di pacchetto dicono che il metodo non è adatto ai servizi in tempo reale a causa de suoi ritardi end-to-end variabili (dovuti alla variabilità e imprevedibilità dei ritardi di accodamento).

I sostenitori rispondono che quest’ultima non soltanto offre una migliore condivisione della larghezza della banda rispetto alla commutazione di circuito, ma è anche più efficiente e meno costosa.

Possibilità di eccessiva congestione: ci può essere un ritardo di un pacchetto a causa di un buffer overflow.

Più avanti vedremo come poter passare da packet switching a circuit switching.

La commutazione di circuito prealloca l’uso del collegamento trasmissivo indipendentemente della richiesta, con collegamenti garantiti, che provoca dispendio di tempo. La commutazione di pacchetto d’altro canto alloca l’uso di collegamento su richiesta.

# STRUTTURA DI INTERNET: una “rete di reti”

* Gli hosts si connettono ad internet tramite Internet Service Providers (ISPs);

È necessario sapere che l’ISP non deve necessariamente essere di una compagnia di telecomunicazione o di televisione via cavo, ma potrebbe essere un’università o un’azienda.

* Per far sì che due hosts si colleghino tra di loro devono avere entrambi accesso ad un ISP;

Se abbiamo un milione di ISPs come si connettono tutti assieme?

Obbiettivo finale è quello di interconnettere gli ISP di accesso con tutti gli altri. Una struttura a maglia sarebbe troppo costosa per gli ISP, in quanto richiederebbe che ad ognuno di essi di avere un collegamento separato per ciascuna delle centinaia di migliaia degli altri ISP sparsi in tutto il mondo.

Poiché l’ISP di accesso paga l’ISP globale di transito, l’ISP di accesso è detto **cliente** (customer) e l’ISP globale di transito viene detto anche **fornitore** (provider).

Nella realtà, sebbene alcuni ISP abbiamo veramente copertura globale e siano invero connessi a molti ISP di accesso alla base, nessun ISP è presente in ogni città del mondo. Al contrario, in ogni regione può esservi un **ISP regionale** al quale tutti gli ISP di accesso della regione si connettono. Ogni ISP regionale si connette all’ISP di primo livello(tier-1 ISP). Esistono solo una dozzina di **ISP di primo livello** tra i quali troviamo Level 3 Communications, AT&T, Sprint e NNT.

ISP globali sono divisi per gruppi in base alla nazione in cui si è e per farsì che gli ISP globali si connettano da gruppo a gruppo esistono gli IXP (è una infrastruttura di rete che consente l’interconnessione tra più di due Autonomous System indipendenti, allo scopo di facilitare lo scambio di traffico Internet).

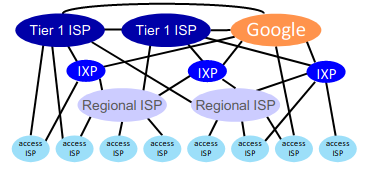
Sempre riguardo agli IXP:

La connessione agli IXP consente a ogni operatore di utilizzare un unico flusso geografico per interconnettersi a una molteplicità di reti di altri operatori (Autonomous System – AS), evitando di dover realizzare tante connessioni quanti sono gli AS con i quali desidera scambiare traffico.

A sua volta esistono gli ISP Regionali = EMEA Europe Middle-East Africa, che si collegano ai gruppi di ISP globali.

Ed infine, i Content Provider Network (p.e. , Google, Microsoft, Akamai) potrebbero eseguire i loro propri network, per portare servizi al utente finale.

Alla fine viene creata una gerarchia, dal Tier 1 che sono gli ISP più potenti fino agli access ISP che sono quelli di più basso livello.



Come possiamo vedere nella gerarchia ad alto livello abbiamo sia i Tier 1 ISP che le reti di fornitori di contenuti, la differenza tra i due è questa:

* “tier-1” ISPs commerciali (e.g., Level 3, Sprint), hanno copertura nazionale e internazionale;
* content provider network (e.g., Google, Facebook): sono reti private che connettono i centri dati ad Internet, spesso bypassando i tier-1, gli ISPs regionali.

# Performance: perdita, ritardo, capacità di trasmissione

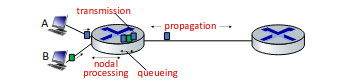
# Come si verificano perdite o ritardi di un pacchetto?

I pacchetti che vanno da un router A a uno B, si mettono in coda dentro il router A se la trasmissione viene fatta su due pc che usano lo stesso router in questo caso c’è un ritardo di trasmissione dei pacchetti.

I pacchetti si mettono in coda nei buffers dei router, aspettando per il turno della trasmissione, la lunghezza della coda cresce quando la velocità di arrivo al collegamento supera la capacità del collegamento di output.

La perdita di un pacchetto si verifica quando la memoria per mantenere i pacchetti in coda si riempe.

# Ritardo di un pacchetto: quattro fonti



d = delay

d(nodal) = d(processing) + d(queue) + d(trans) + d(prop)

d(processing): ritardo di elaborazione

Controlla i bit con dei errori, determina il collegamento di output, tipicamente ci mette meno dei microsecondi;

d(queue): ritardo di accodamento

È il tempo in cui aspetta al collegamento di output per la trasmissione, qui dipende dalla congestione del router;

d(trans): ritardo di trasmissione = L/R

È dato dalla lunghezza del pacchetto (L) e la velocità di trasmissione di un canale (R);

d(prop): ritardo di propagazione = d/s

È dato dalla lunghezza fisica del collegamento (d) e la velocità di propagazione (s) (~2x10^8 m/sec).

# Ritardo di accodamento e perdita di pacchetti (revisited)

a : media del arrivo del pacchetto;

L : lunghezza del pacchetto (bits);

R : larghezza di banda del collegamento (velocità di trasmissione del bit).

L’”intensità del traffico” è dato da: ( L \* a ) / R. Se il risultato è uguale a 0 c’è in media poco ritardo di accodamento; se è uguale a 1 c’è in media un largo ritardo di accodamento e se invece è > 1 c’è una media di ritardo “quasi infinita”.

# I “reali” ritardi di Internet e dei routers

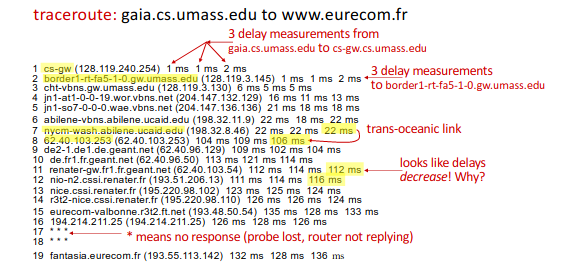
Come vengono visti in realtà in Internet il ritardo e la perdita?

Il programma traceroute ti permette di misurare il ritardo proveniente dal router di partenza (fonte) fino al router destinatario (mandando una sorta di pacchetto), mostrandoti da router a router (intermedi) quanto ci impiega ad arrivare.

Per tutti le i:

* manda 3 pacchetti che raggiungeranno il router i nel percorso in direzione della destinazione (con un campo “tempo di vita” della i);
* il router i restituisce i pacchetti al mittente;
* il mittente misura l’intervallo di tempo tra la trasmissione e la risposta.

Esempio:



# Perdita di pacchetto

Dato che il buffer che mantiene la coda di pacchetti ha uno spazio finito, se per esempio il buffer è pieno, il pacchetto viene perso, ma appena c’è dello spazio libero lo rimanda.

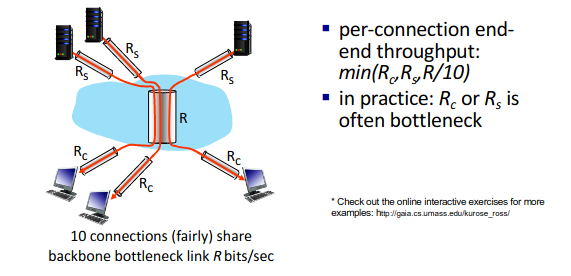
# Capacità di trasmissione

Il Throughput è la velocità(rate) in cui i bits sono mandati da mittente al ricevente. Di solito il Throughput che comanda più di tutti è quello con rate più piccolo (di solito viene chiamato collegamento del collo di bottiglia = bottleneck link). Di tipologie di Throughput ce ne sono due:

* **throughput istantaneo**: è la valutazione che avviene in ogni istante di tempo è la velocità (in bps) in cui il ricevente riceve un file;
* **throughput medio**: è la valutazione oltre il più lungo periodo di tempo.

Rc = rate client

Rs = rate server



# Security

# Network Security

Internet originariamente non era progettata con la sicurezza in mente, ma adesso noi pensiamo a :

* Quanto le persone possano essere cattive nel poter attaccare le reti;
* Come ci possiamo difendere contro degli attacchi;
* Come creare architetture che ti rendano immuni agli attacchi.

Appunti 08/03/2022

# Bad guys: packet interception = intercettazione dei pacchetti

Il packet “sniffing” è una tecnica d’attacco che ti permette di ascoltare cosa passa in rete o no (prende il pacchetto che sta passando in rete e lo analizza). (per farlo esiste un programma : Wireshark).

# Bad guys: fake identity

IP spoofing è la capacità di pretendere di essere qualcun’altro. Iniezione su un pacchetto con falso indirizzo della fonte.

# Bad guys: denial of service = negazione del servizio

Denial of Service (DoS): quelli che ti attaccano fanno in modo che le risorse (esempio: server) non siano più disponibili ad un traffico legittimo sovraccaricando la risorsa con un traffico fasullo.

Il funzionamento di questo attacco è questo, selezioni un target e manda tanti pacchetti al target alla stessa ora rompendo il sito.

Linee di difesa:

* L’autenticazione: ti provvede di dire chi sei tu e se sei veramente tu, per esempio nello smartphone viene dato un hardware identify via SIM card.
* Confidenzialità: via criptata (quando un messaggio è criptato possiamo dire che sotto c’è un algoritmo che ti permette di non fare vedere apertamente cosa dice il tuo messaggio).
* Controlli di integrità: esempio: la firma digitale verifica che non sia un file che possa danneggiare il proprio PC.

VPN = Virtual Private Network.

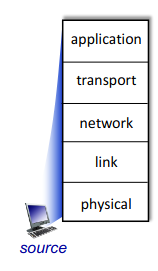
* Accessi ristretti: i VPN sono protetti da password criptate. Con la VPN crei un tubo che va dal host ad un server, che poi vieni visto globalmente come un host con una rete di un’altra nazione.
* Firewalls: ti permettono di filtrare tutte le reti e ti consente di collegarti a reti + sicure, identificano attacchi e ti difendono.

## Livelli di protocollo e modelli di riferimento

Le reti sono complesse e hanno molti “pezzi” (es. Hosts, routers, collegamenti di vari media, etc...), perciò per poter organizzare tutta la struttura di Internet, si suddivide tutto a più livelli.

La visione a più livelli viene detto anche end-to-end,

Layered Internet protocol stacks => Stratificazione dei protocolli



* Applicazioni: supporto alla rete delle applicazioni: HTTP, IMAP, DNS, SMTP;
* Trasporto: si occupa del trasferimento dei dati da processo a processo: TCP, UDP;
* Network: trova un percorso da un nodo ad un’altro/ o anche detto instradamento (da fonte a destinazione) per i datagrammi: IP, routing protocols;

**Definizione datagramma**: pacchetto dati di dimensioni limitate contenente gli indirizzi di provenienza e di destinazione del pacchetto e l’indicazione delle risorse e servizi utilizzati.

* Link: trasferimento di dati nelle vicinanze: Ethernet, 802.11 (WiFi), PPP;
* Fisico: i bits “dentro il cavo”.

Ogni livello ha bisogno di una certa informazione e su quello ci lavorano sopra.

In base al Hardware in cui si è (es: PC, router) ha uno o più livelli in base alle loro necessità.

## Servizi, Livello e Incapsulamento

L’applicazione scambia messaggi per implementare alcuni servizi dell’applicazione usando servizi del livello di trasporto.

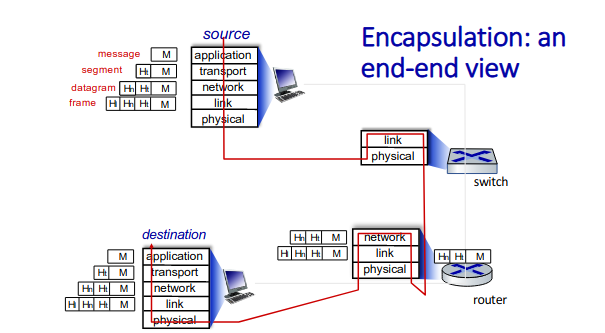
Il protocollo del livello di trasporto trasferisce M (= messaggio) da un processo ad un altro, usando servizi del livello della rete.

Il protocollo del livello della rete trasferisce il segmento del livello di trasporto [Ht | M] (Ht = Header Transport) da un host ad un altro, usando i servizi del livello del collegamento.

* Il protocollo del livello della rete incapsula il segmento del livello del trasporto con l’header Hn (network layer-layer) per creare un livello della rete datagramma.

Il protocollo del livello dei collegamento trasferisce il datagramma [ Hn | [ Ht | M ] ] da un host ad un altro nelle vicinanze, usando i servizi del livello della rete.

Quindi si può ben capire che dopo che un livello più alto ha finito di fare le proprie operazioni su un “dato” e dopo essere stato incapsulato verrà passato ad un livello più basso, che successivamente grazie ai suoi servizi verrà incapsulato e passato ad altri livelli e cosi via finché non si arriva al livello fisico.



Appunti 09/03/2022

# Protocolli a livello applicazione

Alcune apps di rete:

* Social network
* Web
* E-mail
* Giochi di rete multi-user
* Video in streaming (YouTube, Netflix);

# Creazione di una rete di un app

C’è bisogno di scrivere un programma che, venga eseguito in differenti sistemi, che possano comunicare con la rete.

Scrivere un programma che:

* venga eseguito in (differenti) end system → dispositivi che si connette ad Internet;
* comunica con la rete;
* per esempio: un software web server che comunica con il software di un browser.

Non c’è bisogno di scrivere software per i dispositivi network-core (è l’infrastruttura che connette insieme le differenti reti che collegano il tuo sistema del computer).

* i dispositivi network-core non eseguono le applicazione del utente;
* le applicazioni nel end system permettono un rapido sviluppo delle app o propagazione.

# Client-server paradigma

Un server è un computer che ha alcune caratteristiche importanti: deve essere sempre accesso, deve avere un indirizzo IP fisso (noto da tutti i client che usano quel indirizzo), il DNS ti permette di vedere l’elenco di tutti gli IP Server esistenti e infine come ultima caratteristica deve trovarsi in dei data centers (con dentro tante macchine).

Il Client è colui che chiede il servizio dal Server, al Client viene dato un indirizzo IP dinamico rispetto al Server, i Client difficilmente si comunicano direttamente tra di loro.

Ai Client vengono dati dei protocolli che ti permettono di connetterti ai Server: HTTP, IMAP (e-mail), FTP.

# Architettura Peer-to-peer

Nel architettura peer-peer non c’è sempre un server attivo, i peers posso comunicare tra loro, possono richiedere servizi e provvedono servizi ad altri peer. Tra le caratteristiche più importanti dell’architettura: scala in base alla rete, più peers ci sono meglio è (nuovi peer portano una nuova capacità di servizio, così come nuove domande di servizi).

I peers sono connessi in modo intermittente a cambiano l’IP.

Esempio: P2P file sharing.

# Comunicazione tra processi

I processi sono programmi che vengono eseguiti dentro un host, all’interno dello stesso host, due processi comunicano usando inter-process communication. (definito dal SO)

I processi in differenti hosts comunicano scambiandosi i messaggi.

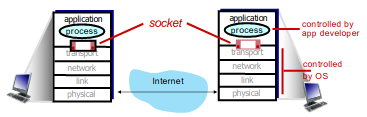
Clients, Servers:

Processo Client: è il processo che inizializza la comunicazione.

Processo Server: è il processo che aspetta di essere contattato.

# Sockets

Il processo manda/riceve i messaggi da/a un socket. Un socket è una porta dove io metto un messaggio e poi arriva ad un altro (processo ricevente).

La porta è un identificativo del processo.

Il socket è analogo ad una porta:

* il processo di invio spinge il messaggio fuori dalla porta;
* il processo di invio si basa sull’infrastruttura di trasporto dall’altra parte della porta per consegnare il messaggio al processo ricevente;
* due sockets sono coinvolti: uno in ogni lato.

# L’indirizzamento dei processi

Per poter ricevere i messaggi, il processo deve avere un identificatore.

Il dispositivo host ha un unico Indirizzo IP a 32-bit

L’Indirizzo IP del host in cui il processo viene eseguito è sufficiente per identificare il processo?

No, dato che molti processi posso essere eseguiti nello stesso host.

Perciò come identificatore si usa sia l’indirizzo IP, che il numero della porta associato con il processo del host.

Esempio:

to send HTTP message to

gaia.cs.umass.edu web server:

• IP address: 128.119.245.12

• port number: 80

# Un protocollo di livello applicativo definisce:

* I tipi di messaggi di scambio possono essere delle richieste o risposte;
* la sintassi del messaggio: indica che campi ci sono in un messaggio e come sono delineati;
* la semantica del messaggio: è il significato del informazione dei campi e le regole su quando e come i processi mandano e rispondono ai messaggi.

I protocolli aperti sono definiti in RFCs e di cui tutti hanno accesso, esempio: HTTP, SMTP; mentre, i protocolli chiusi sono quelli che hanno un proprietario ed è possibile comunicare solo se i protocolli sono comuni tra di loro, esempio: Skype, Zoom.

Che tipo di servizio di trasporto ha bisogno un app? Dipende

Requisiti:

**L’integrità dei dati**;

alcune apps (per esempio: il trasferimento dei file, web transactions) richiedono il 100% dell’affidabilità di trasferimento di dati, ma altre app hanno tolleranza di alcune perdita (anche se alcuni pacchetti vengono persi per loro va bene).

**Tempistica**:

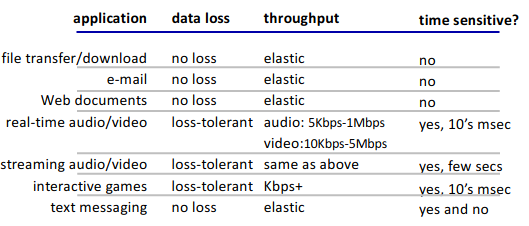
alcune apps possono essere tolleranti nella perdita dei pacchetti ma non sul ritardo, quindi richiedono poco ritardo.

**Capacità di Trasmissione**:

alcune apps richiedono un minimo ammontare di throughput per essere “operativi”; mentre, altre fanno uso di qualsiasi throughput loro attengano.

**Sicurezza**:

encryption, integrità dei dati, etc….

4K = 50kbs

# Protocolli su servizi di trasporto di Internet

Le tipologie di servizi (modelli di trasporto di Internet) sono due:

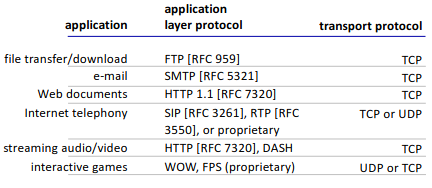
Servizio TCP:

* Affidabilità di trasporto tra un processo che riceve e uno che spedisce;
* Il flow control permette di rallentare una rete più forte rispetto a una più debole;
* Il controllo sulla congestione è un meccanismo in cui stimo che la rete sta diventando congestionata e prendo delle azioni in cui riduco la velocità di un pacchetto per farlo arrivare alla destinazione;
* Connessione orientata: prima di mandare i pacchetti ci mettiamo d’accordo che io devo mandare i pacchetti e io li ricevo, questo viene definito setup tra il processo client e server.
* Il TCP non garantisce il servizio di: tempo, sicurezza e capacità. Lui ti garantisce che il pacchetto ti arriva

Servizio UDP (Unreliable = non garantisce affidabilità):

* Non c’è affidabilità nel trasferimento di dati;
* Non ti da nulla in confronto al TCP.

Perciò domanda che ci viene in mente è: “perché dover utilizzare il servizio UDP?”



Appunti 10/03/2022

# Securing TCP

**Vanilla TCP & UDP sockets** :

non avviene il criptaggio dei messaggi p.e. le password vengono mandate da socket a socket in modo chiaro (cleartext).

**Trasporto di Livello di Sicurezza - [ Transport Layer Security (TLS) ]** :

provvede il criptaggio tra le connessioni TCP, garantiscono l’integrità dei dati e i messaggi che vengono mandati da socket a socket sono criptati.

# Web e HTTP

*Definizione di pagina Web* :

una pagina web è un’insieme di oggetti che sono conservati nel disco di uno o più computer (Web Servers) che mantengo il sito Web.

Gli oggetti in questione possono essere file HTML, immagini, applicazioni Java, etc..

Una pagina Web consiste in un file base HTML che include molti oggetti di riferimento, ognuno indirizzato da un URL.

Esempio:

[www.someschool.edu/someDept/pic.gif](http://www.someschool.edu/someDept/pic.gif)

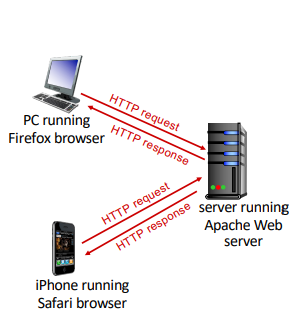
La prima parte di una pagina web è il nome del sito e la seconda parte è il percorso del file che ti serve.

# HTTP overview

HTTP sta per HyperText Transfer Protocol è il livello applicativo del protocollo Web.

Modello client/server :

* Client : naviga quella richiesta, riceve la risposta dal Server, (usando il protocollo HTTP) e “mostra” gli oggetti del Web.
* Server : Web server manda (usando il protocollo HTTP) gli oggetti in risposta alle richieste.



L’HTTP usa il servizio TCP:

* Il cliente inizializza una connessione TCP (crea un socket) al server, con la porta 80;
* il server accetta la connessione richiesta dal client;
* i messaggi HTTP vengono scambiati tra chi cerca in rete (HTTP client) e i Web server (HTTP server).

**HTTP is “stateless”**

La caratteristica principale del protocollo HTTP è di essere **stateless**, ovvero privo di stato.

Ciò significa che la connessione tra *client* e *server* viene chiusa ogni volta che viene esaudita una richiesta.

# Le connessioni HTTP sono di due tipi:

**Connessione HTTP non continua (not-persistent):**

la connessione TCP viene aperta, almeno un oggetto viene mandato con la connessione e poi viene chiusa la connessione.

Scaricando multipli oggetti richiede multiple connessioni.

**Connessioni HTTP continua:**

la connessione TCP viene aperta dal server, una moltitudine di oggetti può essere mandato in una singola connessione TCP tra il cliente e quel server ed infine la connessione viene chiusa.

# Connessione HTTP non continua: Esempio

Utente digita l’URL : www.someSchool.edu/someDepartement/home.index

(contenente un testo, si riferisce a 10 immagini jpeg)

1. l’HTTP del client inizializza la connessione TCP all’HTTP del server al [www.someSchool.edu](http://www.someSchool.edu/) alla porta 80;

2. l’HTTP del server al host w[ww.someSchool.edu](http://www.someSchool.edu/) aspetta per una connessione TCP alla porta 80 “accetta” la connessione, notificando al client;

3. l’HTTP del client manda il messaggio di richiesta HTTP (contenente l’URL) nel socket della connessione TCP;

4. l’HTTP del server riceve il messaggio di richiesta, forma il messaggio di richiesta contenente l’oggetto, e manda il messaggio nel suo socket;

5. l’HTTP del server chiude la connessione TCP;

6. l’HTTP del client riceve il messaggio di risposta contenente il file html, gli viene mostrato l’html. Viene fatto un parsing (analisi) del file html, trova 10 riferimenti oggetto jpeg

7. Gli step 1-6 vengono ripetuti per ciascuno dei 10 oggetti jpeg.

La connessione non persistenti ha problemi di overhead, ciò va ad impattare le performance del caricamento del sito

Il RTT (Round Trip Time) è tempo necessario di un pacchetto per andare da una sorgente a una certa destinazione.

# Connessione HTTP non continua: tempo di risposta

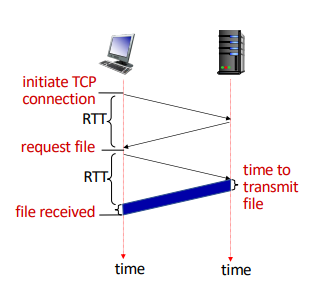
RTT (Round Trip Time = Tempo di Andata e Ritorno):

è il tempo per un piccolo pacchetto per viaggiare dal client fino al server e ritornare indietro.

Tempo di risposta del HTTP (per ogni oggetto):

* un RTT per inizializzare la connessione TCP;
* un RTT per la richiesta HTTP e i primi pochi bytes dell’HTTP di risposta per restituire l’oggetto richiesto;
* il tempo di trasmissione dell’oggetto/file.

**Il tempo di risposta di una connessione HTTP non continua è dato da: 2RTT + il tempo di trasmissione del file.**

****

# Connessione HTTP continua (HTTP 1.1)

**Problemi della connessione HTTP non continua:**

* richiede 2 RTTs per oggetto;
* il Sistema Operativo va in overhead per ogni connessione TCP;
* le ricerche spesso aprono multiple connessioni TCP parallele per andare a prendere oggetti di riferimento in parallelo.

**Connessione HTTP continua (HTTP1.1):**

* il server lascia la connessione aperta dopo aver mandato la risposta;
* le sotto sequenze dei messaggi HTTP tra lo stesso client/server vengono mandati nella connessione aperta;
* il client manda le richieste non appena incontra un oggetto di riferimento;
* solamente un RTT per tutti gli oggetti di riferimento (tagliando in mezzo il tempo di risposta)

# Messaggi di richiesta HTTP

I due tipi di messaggi che ha HTTP: richiesta e risposta, la richiesta dei messaggi viene codificato in ASCII (human-readable format).

Linea di richiesta (GET, POST, HEAD commands)

All’interno ci possono essere i caratteri speciali, es. Carriage return o new line (\r, \n).

# Messaggio di richiesta HTTP: formato generale

# Altri tipi di messaggi di richiesta HTTP:

POST method:

* nelle pagine web spesso include qualcosa da mettere in input;
* l’input dell’utente mandato dal client al server in un corpo di un’entità del messaggio di richiesta HTTP POST.

GET method (per mandare dei dati al server):

* include il dato dell’utente nel campo dell’URL del messaggio di risposta HTTP GET ( seguito da un ‘?’):

Esempio: www.somesite.com/animalsearch?monkey&banana

HEAD method:

* le richieste headers (solo) che dovrebbero essere restituite se viene specificato l'URL che è stato richiesto con un metodo HTTP GET.

PUT method:

* carica un nuovo file (oggetto) al server;
* rimpiazza completamente il file che esiste ad un specifico URL con contenuto dentro il corpo dell'entità del POST HTTP messaggio di risposta.

# Messaggio di risposta HTTP

Esempio:

**HTTP/1.1 200 OK**

Linea di stato: protocollo, versione e numero per dirti se è tutto ok (200) o no (404);

Linea di Header: Data, server, ultima modifica della pagina, ETag signature per identificare la transazione, i bytes.

# Codici di stato di risposta HTTP

* Il codice di stato appare nella prima linea del messaggio di risposta dal server al client.
* Ecco alcuni esempi di codice:

200 OK

* la richiesta ha avuto successo, ha richiesto l’oggetto dopo questo messaggio

301 Mosso Permanentemente

* la richiesta dell’oggetto si è mossa, una nuova locazione è stata specificata dopo questo messaggio

400 Richiesta non formata correttamente

* il messaggio di richiesta non è stato capito dal server

404 Non Trovato

* il documento richiesto non è stato trovato in questo server

505 Versione dell’HTTP non supportata

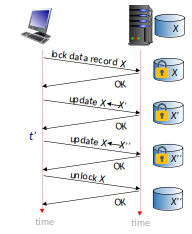
Appunti 16/03/2022

# Mantenere lo stato del user/server: cookies

Richiamo: all’interazione di risposta/GET HTTP che è stateless.

* Non c’è la nozione di multi-step di scambio di messaggi HTTP per completare una transazione del Web;
* non c’è bisogno per il client/server di tracciare lo “stato” dei multi-step di scambio;
* tutte le richieste HTTP sono indipendenti una dalle altre;
* non c’è bisogno per il client/server di “recuperare” da una parzialmente completa ma mai completamente completa transazione.

Un protocollo a stati: il client fa due cambi di X, o non del tutto.



In generale è molto importante per la rete ciò che per esempio stai cercando o cosa ti interessa, per poi venderti ciò a cui sei interessato.

Il meccanismo che permette alla rete di bloccare le informazione che stai cercando è attraverso i cookies, permettono di mantenere una sorta di stato.

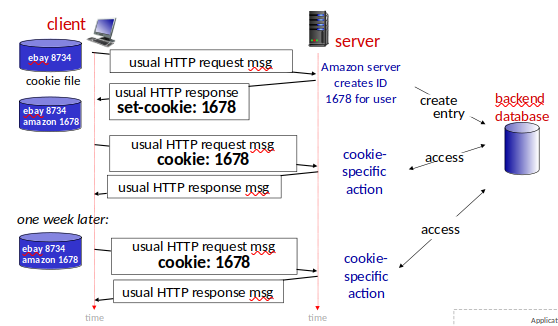
In HTTP questo non avviene (il mantenimento di uno stato) quindi appena finisce la storia dell’altra, non si ricorda più quello che hai fatto prima.

Principalmente ci sono 4 componenti:

1. la linea del Cookie header del messaggio di risposta HTTP
2. Lo stesso header viene messo nel prossimo HTTP
3. Cookie mantengono il file del host del utente, gestito dal browser del utente
4. I database del back-end sul sito Web

Esempio:

* Susan usa un browser nel laptop, per la prima volta visita un sito specifico di e-commerce;
* quando iniziano le richieste HTTP arrivano al sito, il sito crea:
  + un ID unico (chiamato “cookie”);
  + viene messo in entrata nel database back-end per l’ID.
* le sotto sequenze delle richieste HTTP da Susan a questo sito conterranno il valore dell’ID del cookie, permettendo al sito di “identificare” Susan.



I cookies sono utili per:

* l’autorizzazione, quindi per esempio di memorizza i dati di accesso di un account;
* Carrelli dei shopping;
* Raccomandazioni;
* Lo stato della sessione del utente (p.e. ti ricorda che messaggi hai letto e cosa no nelle e-mail).

**Come mantiene lo stato?**

Lo può mantenere in due modi:

* Da un protocollo endpoint: mantiene lo stato al mandante/ricevente oltre a multiple transazioni;
* In messaggi: i cookies stanno nel carry state dei messaggi HTTP.

Dall’altra parte

Cookies e privacy:

I Cookies permettono ai siti di imparare molto su di te.

Le terze parti in continuazione permettono alle identità comuni di essere tracciati in differenti siti web.

# Web caches

**Obiettivo**: soddisfare le richieste del cliente senza coinvolgere il server originale.

Se l’oggetto che vogliamo si trova già nel Web cache ti dà quello che ha già in disposizione (copia), se invece non era già disponibile, ti dà quello che ha il server originale (copia).

* L’utente configura il browser per puntare ad un (locale) Web cache;
* il browser manda tutte le richieste HTTP alla cache:
  + se l’oggetto è in cache: la cache restituisce quell’oggetto al client
  + altrimenti mette in cache la richiesta dell’oggetto dal server originale, memorizza l’oggetto ricevuto, poi restituisce l’oggetto al client.

Se guardassimo sempre cose diverse la cache non servirebbe a nulla, ma dato che noi molto spesso guardiamo le stesse cose è indispensabile ogni tanto non disturbare direttamente la pagina web per la richiesta.

I Web cache attuano sia da parte dei client che da parte dei server:

* I servers per la richiesta originale del client
* Dai client al server originale

I server dicono alla cache che tipo di oggetto è permesso di essere messo in cache nella risposta dell’header.

# Perché fare del caching nel Web?

* Per ridurre il tempo di risposta per la richiesta del cliente, il cache più vicino al cliente;
* Riduce il traffico in un accesso (organizzativo) del collegamento;
* Internet è più denso (più fluido) con le caches.

Se installiamo un web cache, per far sì che sia stato worth bisogna vedere se le richieste più simili tra loro ci sono già nella cache.

L’introduzione della cache ha aggiunto all’HTTP una nuova necessità

# Conditional GET

Obiettivo: non mandare l’oggetto se la cache ha aggiornato (up-to-date) la versione memorizzata.

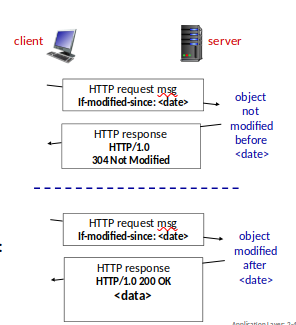
* Il cliente specifica la data della copia memorizzata nella richiesta HTTP.

Esempio: **If-modified-since: <date>**

* il server risponde che non contiene l’oggetto se la copia memorizzata è aggiornata.

Esempio: **HTTP/1.0 304 Not Modified**

Quindi di norma se la data non è cambiata ti dà ciò che c’è già nella cache attuale.



# **HTTP/2**

Obiettivo chiave: decrementare il ritardo di multipli oggetti nella richiesta HTTP.

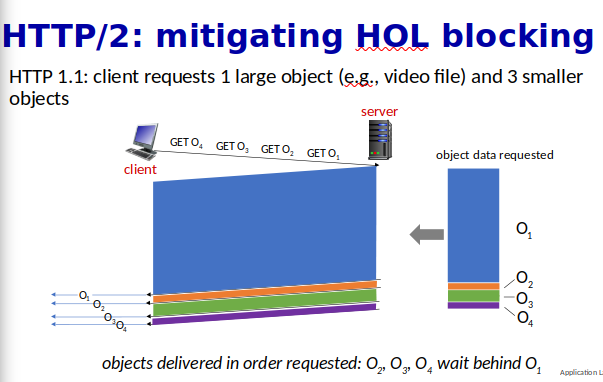
HTTP1.1 ha introdotto multipli GETs oltre alla connessione TCP

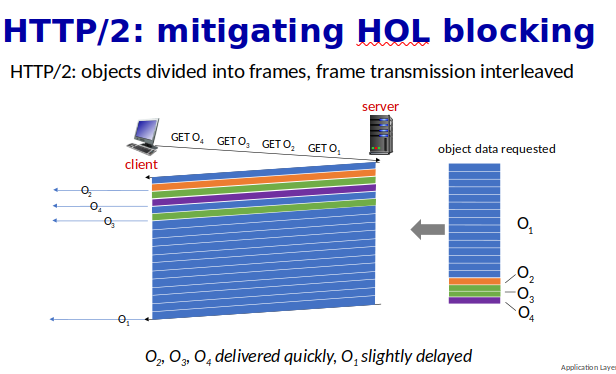
* I server risponde in ordine (FCFS: first-come-first-served scheduling -> il primo che viene è il primo ad essere “servito”);
* Con FCFS, i piccoli oggetti potrebbero aspettare per la trasmissione di oggetti più grandi.

HTTP/2: ha incrementato la flessibilità al server mandando oggetti al client:

* Metodi, codici di stato, e molti campi header non cambiano dal HTTP 1.1
* La trasmissione viene ordinata in base alla richiesta dei oggetti ed è basata sulla priorità dell’oggetto specificata dal client;
* “Spinge” gli oggetti non richiesti dal client;
* Divide gli oggetti in frames, ordina i frames per ridurre (mitigate) l’HOL (Head-of-line) blocking -> è un fenomeno sulla limitazione delle performance che accade quando una linea di pacchetti è tenuto da primo pacchetto. Quindi se per esempio due flussi arrivano contemporaneamente solo il primo che si è imposto prima può essere trasferito, mentre l’altro resta bloccato.

**Esempi GRAFICI BELLI:**



Piccola nota di Gianluca (facoltativo, ma se lo capisci, capisci come avviene la riduzione dell’HOL blocking):

Nell’HTTP 2 se per esempio un’immagine grande che deve essere caricata in una pagina,

frammenta l’immagine e poi manda l’immagine grande, pezzetto per pezzetto mixato con i pezzetti di immagine più piccoli, ciò rende all’occhio umano il caricamento un po’ più fluido

Appunti 17/03/2022

# **Da HTTP/2 a HTTP/3**

Un problema dell’HTTP 2 è che c’è un’unica connessione tra server e client, ciò comporta a una cosa, se viene perso un pacchetto per ritrasmetterlo ci mette di più con un delay, per risolvere questo problema avevano avuto l’intuizione di aprire più connessioni TCP (ricordiamo che questo tipo di connessione non è sicura).

(Protocollo usato principalmente da Google Chrome)

A questo punto alla fine hanno deciso di creare un nuovo protocollo: HTTP/3, che aggiunge più sicurezza, più controlli sulla congestione e l’utilizzo della connessione UDP.

# E-mail

Di protocolli per l’e-mail ce ne sono molti. La e-mail ha tre di componenti principali:

* User agent: sono le applicazioni che usiamo noi per mandare la posta (out-look, apple-mail);
* Server di posta: ha due elementi, user-mail box: una la buchetta di posta dell’utente (contenitore e-mail); messaggi in coda in uscita: uno spazio che mette in coda tutti i messaggi che sono arrivati;

N.B. : Esiste un certo contratto (GDBR) che dice, che ogni user mailbox deve restare nello stesso territorio in cui abiti (es. Europeo).

* Il semplice protocollo di trasferimento delle mail: SMTP.

Il protocollo SMTP è tra i server di posta per mandare i messaggi email

* client: manda al mail server;
* “server”: riceve dal mail server.

# SMTP RFC (5321)

Anche l’SMTP usa la connessione TCP per il trasferimento dei messaggi, la porta dell’SMTP è la 25, la connessione avviene con un round trip time che la traccia.

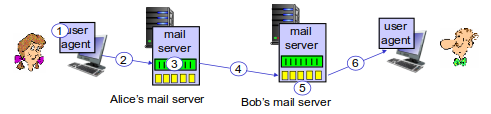
L’email ha tre fasi di trasferimento:

* la prima è l’handshake autentica verso il server per sapere chi sei tu; (SMTP handshaking)
* viene mandato il messaggio; (SMTP transfer of messages)
* viene chiusa la connessione. (SMTP closure)

l’SMTP usa come comandi l’ASCII text e risponde con i codici di stato e le frasi (come HTTP).

# Scenario: Giock manda l’e-mail a Gillas

1. Giock usa UA per comporre il messaggio e-mal “a” [gillas@someschool.edu](mailto:gillas@someschool.edu);
2. L’UA di Giock manda il messaggio al suo server delle mail usando SMTP; il messaggio viene piazzato nella coda dei messaggi;
3. dalla parte del client del SMTP nel server delle mail apre una connessione TCP con il server delle mail di Gillas;
4. l’SMPT del client manda il messaggio di Giock nella connessione TCP;
5. il server delle mail di Gillas piazza il messaggio nella mailbox di Gillas;
6. Gillas invoca il suo user agent per leggere il messaggio.



# Esempio SMTP interazione

Quando uno si collega ti dà il server e ti conferma

l’altro saluta HELO

Il mittente viene identificato con MAIL FROM

Ti dà l’ok

Conferma chi sei RCTP TO

Recipient ok se il destinatario esiste

QUIT -> chiude la connessione TCP

# SMTP: osservazioni

Le grandi differenze tra HTTP e SMTP è che nel primo si pulla e nel secondo si pusha, entrambe usano interazioni di command/response, ASCII e codici di stato.

HTTP: ogni oggetti viene incapsulato nel proprio messaggio

SMTP: molti oggetti possono essere inviati in un singolo messaggio.

SMTP richiede di essere 7-bit ASCII

# Formato dei messaggi delle Mail

SMTP: è il protocollo per lo scambio dei messaggi, definito in RFC 5321

RFC 2822 definisce la sintassi per i messaggi delle mail e la semantica del protocollo (proprio come fa HTML per definire la sintassi per documenti web).

Header line e.g.:

* To:
* From:
* Subject:

# Recupero di una email: protocolli di accesso della mail

* Il protocollo di accesso della mail: recupera dal server
* IMAP: è un protocollo che mantiene tutto lo stato del utente che usa la mail, IMAP se la salva sul server di modo che tutte le modifiche che hai fatto restino salvate.
* HTTP ti provvedere l’interfaccia per l’SMTP (per mandare), per IMAP che recupera i messaggi delle email (non essendo stateless si ricorda tutto ciò che hai fatto), mentre POP essendo stateless non si ricorda.

# ***DNS: Domain Name System***

Il DNS consente di associare identificativi. Per le persone, abbiamo il CF, il passaporto etc..., per Internet come codice univoco c’è l’indirizzo IP. Per ricordare l’indirizzo IP invece che usare dei numeri si usa un “nome” che possa essere capito dai umani, es.: cs.umass.edu.

Come è possibile fare un mapping tra il nome e l’indirizzo IP e viceversa?

DNS è stata costruita in maniera gerarchica (per una gestione migliore) in base ai molti nomi dei servers.

Protocollo a livello applicativo: gli hosts, i DNS dei server comunicano per risolvere i nomi (traslazione tra indirizzo/nome)

* nota: è il nucleo del funzionamento di Internet, ed è implementato come un protocollo a livello applicativo.

Il DNS si trova nel protocollo delle applicazioni, se il DNS non funziona più possiamo dire che non funziona più nulla, in linea di massima è più importante rispetto ad un Web server, le informazioni sono conservate nelle foglie, se voglio aggiungere il nome di un indirizzo IP basta chiedere a quello che ha configurato tutto localmente.

# DNS: servizi, strutture

I servizi DNS:

* Traslazione dal nome del host al indirizzo IP;
* Pseudonimo host;
* Pseudonimo sui server delle mail;
* Caricamento distribuito, riprodotto nei Web servers: permette di collegare più indirizzi IP ad un nome univoco.

Perché il DNS non è centralizzato?

* singolo punto di fallimento;
* volume del traffico;
* distanza del database centralizzato;
* manutenzione.

Il DNS non lo hanno fatto centralizzato perché basterebbe poco per buttarlo giù in un colpo solo, ma anche per le biliardi o triliardi di queries che vengono mandati al DNS.

# Pensando riguardo il DNS

Un database distribuito in modo gigantesco:

* miliardi di registri, ogni semplice

maneggia molte bilioni di queries al giorno:

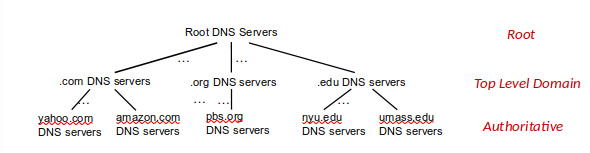
* ci sono molte più letture che scritture
* per le performance: almeno ogni transizione di Internet interagisce con il DNS – contano msec!

Organizzativo, fisicamente decentralizzato:

* milioni di differenti organizzazioni sono responsabili dei loro registri.

“a prova di proiettile”: affidabile, sicuro.

# DNS: un database gerarchico e distribuito



Un DNS è stato strutturato come un albero, ad ogni nodo (dominio) ci sono dei sotto-domini.

Esempio:

Il cliente vuole l’indirizzo IP per [www.amazon.com](http://www.amazon.com/):

* Il cliente chiede al server root di cercare il .com DNS server
* Per poi chiedere al .com DNS server di ottenere amazon.com DNS server
* Per poi chiedere amazon.com DNS server di ottenere l’indirizzo IP per www.amazon.com.

# DNS: root name servers (nomi dei server radice)

* Ufficialmente, è il contatto di ultima istanza per nome dei servers che non posso risolvere il nome
* incredibilmente importante per il funzionamento di Internet
* DNSSEC – protocollo di sicurezza che provvede la sicurezza (autenticazione, integrità di messaggi)
* ICANN (Internet Corporation for Assigned Names and Numbers) gestisce il dominio dei root DNS. In totale sono 13 logici nomi root “servers” in tutto il mondo.

# Top-Level Domain, and authoritative servers

I Servers Top-Level Domain (TLD):

* I domini Top-Level sono responsabili per siti che finiscono per .com, .org, .net, etc..., e di tutti i domini top-level di tutti i paese .it, .fr, .jp, etc…;
* Le soluzioni della rete: il registro autorevole per .com, .net TLD;
* Educazione: .edu TLD

I Servers DNS autorevoli:

Subito sotto i Top-Level ci sono i DNS autorevoli, che è proprietario dell'organizzazione del DNS del server, associano l’hostname agli IP mappando gli hosts nominati dall’organizzazione.

Infine ci sono i server locali, se sa dove si trova ti manda a quel sito, se non lo sa risale la gerarchia per potertelo trovare.

Appunti 22/03/2022

# DNS: local name servers

* Quando l’host fa una query di un DNS, è mandato al suo local DNS server
  + I DNS locali restituiscono una risposta, ad una richiesta:
    - dal suo cache locale della recente traslazione del nome di indirizzo ( è possibilmente non aggiornato!)
    - va avanti con la richiesta nella gerarchia dei DNS per la risoluzione
  + ogni ISP (Internet Service Provider) ha un local DNS name server; per cercare il vostro:
    - MacOS: % scutil –dns
    - Windows: >ipconfig /all
  + i DNS locali non appartengono in modo stretto alla gerarchia.

I DNS locali sono presenti sia nelle organizzazioni che a casa nostra, il tuo PC per capire che DNS locale prendere ha un protocollo speciale che gli dice qual’è l’IP di Default.

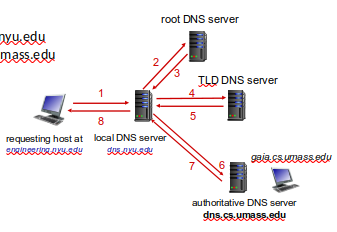
Come risolvere le query del DNS?

Per il momento esistono due tipi di query, una iterativa e l’altra ricorsiva.

La query iterativa è pensata per limitare il carico sulla parti alte della gerarchia

# *Risoluzione del nome del DNS: query iterativa*

Esempio: un host al engineering.nyu.edu vuole risolvere l’indirizzo IP di gaia.cs.umass.edu



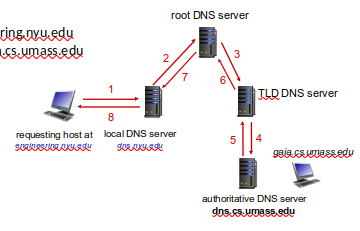
Query iterativa:

* Il server contattato risponde con il nome del server da contattare.
* “io non conosco questo nome, ma chiedi a questo server”

Prima il server locale chiederà al root poi ti viene mandata una risposta dicendoti di cercare nel Top Level, poi a sua volta il Top Level ti risponde mandandoti direttamente al DNS authoritative.

# *Risoluzione del nome del DNS: query ricorsiva*

Riferito all’esempio fatto prima:



Query ricorsiva:

* mette il carico della risoluzione del nome nel nome del server contattato
* il carico è pesante ai livelli alti della gerarchia?

l’host chiede al DNS locale, DNS locale al root DNS server, il root DNS server chiede al Top Level Domain, che a sua volta chiede al authoritative, a quel punto la risposta rimbalza da Authoritative DNS → TLD → root DNS → local DNS a host DNS

# Caching DNS Information

Una volta che il nome del server è stato memorizzato, viene mappato in una cache e se viene richiesto, immediatamente ti ritorna ciò che è stato mappato nella cache in risposta alla query.

Il seguente caching non resta memorizzato all’infinito, dopo un certo tempo viene rilasciato.

I servers dei Top-Level Domain vengono cached nei servers dei nomi locali.

Le entrate potrebbero essere obsolete:

Se il nome del host cambia l’IP (significa che ciò che c’è dentro la cache potrebbe non essere aggiornato), potrebbe non essere conosciuto nell’ampiezza di Internet fino a quando tutti gli TTL scadono.

**Notazione del TTL (Time To Live) Giockiana:**

Il time to live (TTL) è un meccanismo che determina il tempo di vita di un dato computer di una rete.

Nel caso di un record DNS, indica per quanto tempo può essere ritenuto valido. Il tempo è espresso in secondi oppure può essere un timestamp.

# DNS records

DNS: è un database distribuito che immagazzina le resource records(RR)

formato RR: (name, value, type, ttl)

type=A

* *name* è l’hostname
* *value* è l’indirizzo IP

type=CNAME

* *name* è il nome pseudonimo per alcuni nomi “canonici” (quello vero)
* [www.ibm.com](http://www.ibm.com/) è in realtà servereast.backup.ibm.com
* *value* è il nome canonico

type=NS

* *name* è il dominio (per esempio: foo.com)
* *value* è l’hostname del nome del server autorevole per questo dominio

type=MX

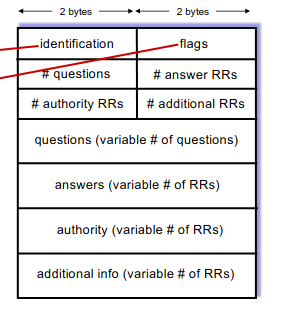
* *value* è il nome del server mail SMTP associato con il *nome*

# Protocollo dei messaggi del DNS

I messaggi di domanda e risposta del DNS, hanno lo stesso formato:

L’header del messaggio:

* l’identificazione: 16 bit # per la query, risposta alla query usa lo stesso # di bit;
* Flags: query o risposta, ricorsione desiderata o disponibile, la risposta è autorizzata.



# Ottenere le tue informazioni dal DNS

Esempio: nuova startup “Network Utopia”

* registrare il nome networkuptopia.com al registratore di DNS (per esempio: Network Solutions)
  + provvedi i nomi, gli indirizzi IP del nome del server autorevole (primario e secondario)
  + il registratore inserisce come type l’NS, A Resource Records nel .com Top-Level Domain server:

(networkutopia.com, dns1.networkutopia.com, NS)

(dns1.networkutopia.com, 212.212.212.1, A)

* crea un server autorevole localmente con l’indirizzo IP 212.212.212.1
  + tipo A registro per www.networkuptopia.com
  + tipo MX registro per networkuptopia.com

# DNS sicurezza

Gli attacchi alla root molto difficilmente hanno successo, stessa cosa per i TLD, gli attacchi spoofing intercettano.

Gli attacchi DoS

* bombardare i server root con il traffico
  + non riuscito fino ad oggi;
  + viene filtrato il traffico
  + i servers dei DNS locali memorizzano gli IPs dei server Top-Level Domain, permettendo di bypassare il server root
* bombardare i server TLD
  + potenzialmente molto più pericoloso

Gli attacchi Spoofing

* intercetta le queries DNS, restituendo risposte fasulle
  + avvelenando la cache del DNS;
  + RFC 4033: DNSSEC servizi di autenticazione.

# Architettura peer-to-peer

Caratteristiche di quest’architettura:

Non sempre sono on i server, arbitrariamente (di propria iniziativa) i sistemi possono comunicare direttamente.

Le richieste dei peers di servizio da altri peers, provvede servizi di ritorno ad altri peers.

I peers sono connessi in modo intermittente e cambiano gli indirizzi IP.

Una proprietà che ha quest’architettura è l’auto-scalabilità (self scalability): nuovi peers porta una nuova capacità di servizio, e nuove domande di servizio.

Esempi: condivisione di file P2P (BitTorrent), streaming (KanKan), VoIP (Skype).

# Distribuzione di un file: client-server vs P2P

Quanto tempo serve per distribuire un file (di dimensione F) da un server a N peers?

Il peer nella capacità di caricare o scaricare è limitato dalla risorsa.

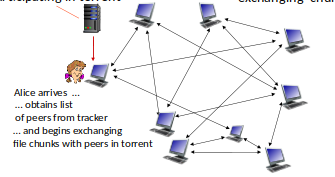
**MANCANO LE SLIDE IN CUI PARLA DEL CALCOLO DI TRASMISSIONE DI UN FILE: FATTO TRA CLIENT-SERVER VS P2P**

# Distribuzione di un file P2P: BitTorrent

* Il file viene suddiviso in 256Kb di pezzetti
* i peers nel “torrente” = torrent mandano/ricevono i pezzetti del file

tracker: traccia i peers che partecipano al torrente

torrent: gruppo di peers che scambiano i pezzetti del file



BitTorrent: suddivide la parte che c’è in unpload da ciò che c’è in download.

Il Tracker contiene la lista dei peer attivi in quel Torrent.

Un tipo arriva ottiene la lista dei peers dai tracker e inizia a scambiare chucks con i peers in torrent.

* Il peer che entra nel torrente:
  + non ha pezzetti, ma li accumulerà col tempo da altri peers
  + registra (si appunta) con il tracker per ottenere la lista di peers, connessi al sottoinsieme di peers (“vicini”)
* mentre sta scaricando, il peer carica dei pezzetti ad altri peers
* il peer potrebbe cambiare i peers con cui stavano scambiando i pezzetti
* churn: i peers potrebbero venire e andare via
* una volta che il peer ha l’intero file, può decidere di andarsene o rimanere nel torrente

Appunti 23/03/2022

# Streaming video e CDNs (Content Delivery Network = “Rete per la consegna di contenuti): contesto

Tramite lo streaming dei video è molto alto il traffico su Internet.

La prima challenge: è quella del traffico.

La seconda: è di poter raggiungere 1 Miliardo (Billion) di utenti.

La terza: è la eterogeneità, dato che differenti utenti hanno differenti capacità di rete.

Perciò la soluzione è la costruzione di una sovrastruttura di rete distribuita, che determina una serie di algoritmi per una distribuzione a una grande scala.

Un esempio sono: Netflix, Akamai, KanKan (Cina), YouTube

# *Video multimediale*

Viene definito in questa maniera, dato che inoltre a trasmettere il video viene trasmesso anche l’audio.

Il video è una sequenza di immagini, mostrata all’utente ad una velocità costante.

L’immagine digitale: è un vettore di pixel a sua volta è composto da byte di diverso colore, descrizione immagine, per mandarle devono essere codificate con meccanismi di encoding per la ridondanza (anche se si perdono i pacchetti, si possono riottenere).

Coding spaziale:

Invece di mandare N valori dello stesso colore, ne vengono mandati solo due valori: il colore del valore e il numero di valori ripetuti.

Coding temporale:

Per avere un costo minore per i frame, vengono trasmesse solo le differenze, non tutto il contenuto (da frame a frame).

Le velocità di codifica che esistono sono:

* CBR (Constant Bit Rate = Velocità dei Bit Costante): la velocità di codifica del video è fissata;
* VBR (Variable Bit Rate = Velocità dei Bit Variabile): la velocità di codifica cambia come l’ammontare del cambiamento di codifica spaziale o temporale.

# Video Streaming immagazzinato

Ci sono alcuni algoritmi/standard per la compressione del video, la quantità di output dipende dalla qualità del video. Più alti sono i valori in ingresso, più alti sono quelli in output.

Transcoding video: in base a dove viene riprodotto il video, ci sono differenti versioni per ogni tipo di dipositivo

I video sono dentro un server che sono pronti all’uso e il client quando vuole se li scarica/li e se li guarda.

* La principale challenge è che la banda di rete cambierà nel tempo, con il cambiamento del livello della congestione (ma anche per altri fattori: nella casa, l’accesso della rete, il nucleo della rete, il server del video).
* La perdita di un pacchetto, ritarda a causa della congestione che ritarderà il playout, o risulta in una qualità “povera”.

# Video Streaming immagazzinato: sfide

Per risolvere questo problema bisogna avere dei buffer un po’ per tutta la rete.

Il video viene registrato in 30fps e poi facciamo vedere il video al client a 30fps, se non ci fossero problemi di rete tutto funziona perfettamente senza l’uso dei buffer (cosa irrealizzabile nella realtà).

I clienti esprimono tanta rottura de scatole se si ferma il video, ma non sentono la differenza nella qualità del video.

I server devono avere sempre pacchetti che vanno 30 fps.

Il problema vero nel mondo reale è che non ci sia una rete senza ritardo di pacchetto (jitter problematic).

**Appunto Giockiano sul jitter:**

il jitter è quando più pacchetti hanno valori di latenza molto diversi tra loro. Questo fenomeno è indesiderato perché può interrompere il buon funzionamento della connessione e causare la perdita dei pacchetti.

Un’altra challenge è l’interazione dell’utente: pausa, mandare indietro o avanti il video.

Per risolvere tutti questi problemi vengono creati più buffer usati dall’utente (buffering and playout delay).

# Streaming multimediale: DASH (Dynamic Adaptive Streaming over HTTP)

Visto che all’utente gli importa dei 30 fps, è stato creato un protocollo (DASH – Dynamic Adaptive Streaming over HTTP [usa TCP]) che riuscisse a gestire tutte le problematiche che abbiamo visto prima:

Server:

* Divide il video in più chunck (pezzetti ognuno di esse ha vita a se);
* Il pezzetto viene codificato in differenti velocità;
* codifiche di velocità diverse vengono messe in un file differente;
* I files vengono posizionati e replicati in vari nodi CDN;
* Manifest file: file HTML che descrive in modo dettagliato tutto ciò che riguarda i chunck, dove si trovano nella rete, etc...

Client:

* Osserva la banda disponibile (guardo il mio limite in cui posso scaricare);
* Consultando il manifest, richiede un chunck alla volta;
* Sceglie il massimo encoding rate sostenibile data la corrente banda;
* Può scegliere diverse codifiche in differenti punti nel tempo e da differenti server.

L’intelligenza del client determina:

* Quando richiedere il chunck evitando di avere il buffer vuoto o avere troppi dati in memoria;
* Che velocità di codifica: in base alla stima della capacità disponibile sulla rete decide la qualità del chunck;
* Dove richiedere il chunck: ha la scelta di molteplici sorgenti del video, decide in che server prendere il video (più vicino o no)

Streaming video = enconding + DASH + playout buffering.

# Content distributions on networks (CDNs = “Rete per la consegna di contenuti)

Challenge: come streammare il contenuto a centinaia di utenti simultaneamente?

La prima soluzione è un server unico che fa tutto, questa soluzione non è scalabile.

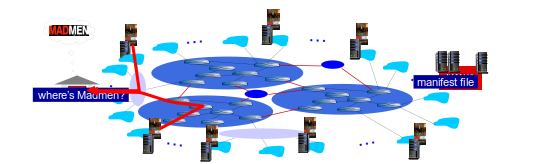
La seconda è quella di distribuire i video (copie) in più server.

Leader di mercato Akamai.

La CDN è una sorte di rete che sta sopra Internet, Internet gli da il tubo e la CDN gli dà la strategia per usare quel tubo.

La CDN viene usata per lo streaming video.

* CDN: immagazzina copie (per esempio: MADMEN) ai nodi CDN
* l’iscritto richiede il contenuto, colui che provvede il servizio restituisce il manifest
  + usando il manifest, il cliente recupera il contenuto ad una velocità sostenibile (ma più alta possibile);
  + potrebbe scegliere una velocità diversa o direttamente fare una copia se il percorso della rete è intasato.



Appunti 24/03/2022

# Programmazione di un socket

Obiettivo: imparare come costruire un’applicazione client/server che comunica usando i sockets

Socket: porta tra il processo applicativo e il protocollo di trasporto end-end.

**Commento di Giock: Mi sà che sta parte è stata skippata solo perché la programmazione di un socket avviene già/viene spiegata durante le ore di laboratorio.**

# Servizi e protocolli di trasporto

* Provvede una comunicazione logica tra i processi applicativi che sono in esecuzione in diversi hosts.

Un protocollo di trasporto non ha idea di che semantica ci sia dietro.

Azioni del protocollo di trasporto:

* Mittente: spezza i messaggi dell’applicazione in segmenti per poi passarli al livello di rete
* Destinatario: riassembla i segmenti in messaggi, passando al livello applicativo.

Ci sono due tipi di protocollo di trasporto disponibili nelle applicazioni di Internet: TCP, UDP.

Abbiamo in host con diversi processi, e ovviamenti tutti i messaggi inviati devono arrivare ai processi giusti (cosa già vista).

# Trasporto vs. livello dei servizi e protocolli della rete

* Livello della rete: comunicazione logica tra gli hosts.
* Livello di trasporto: comunicazione logica tra i processi.

# Azioni nel livello di trasporto

Mittente:

* Il messaggio viene passato nel livello applicativo;
* Determina i valori dei campi header del segmento;
* Crea il segmento;
* Passa il segmento in un IP.

Ricevente:

* Riceve il segmento dal IP;
* Controlla i valori dei headers;
* Estrae il messaggio dal livello applicativo;
* Unisce il messaggio dal applicazione via socket.

# I due principali protocolli di trasporto di Internet

* TCP: Transmission Control Protocol
* UDP: User Datagram Protocol

# Multiplexing/Demultiplexing

Multiplexing al mittente:

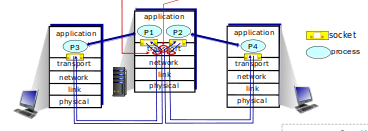
maneggia il dato da multipli socket, aggiunge l’header di trasporto. (questo header verrà poi utilizzato nel demultiplexing)

Demultiplexing al destinatario:

usa le informazioni dell’header per consegnare i segmenti ricevuti per correggere il socket.

Primo problema da risolvere: multiplexing e demultiplexing

Abbiamo applicazioni multiple; esempio ci sono due client che vogliono la stessa pagina e alla fine bisogna mandare il sito giusto al client giusto

Per risolvere questo problema ci sono dei socket che sono collegati da client a server.

Gli host mandano i pacchetti IP, dentro i pacchetti ci sono dei header che ti dicono in che processo andare alla fine.

L’host usa gli indirizzi IP e il numero delle porta per direzionare il segmento al appropriato socket.

Nell’header ci sono 2 campi 16 bit uno: campo host source a un’altro sulla destinazione.

# Come funziona il demultiplexing

* L’host riceve gli IP dei datagramma
  + ogni datagramma ha l’indirizzo IP della fonte, della destinazione;
  + ogni datagramma porta con se il segmento del livello di trasporto;
  + ogni segmento ha il numero di porta della fonte e destinazione.
* Gli hosts usano gli indirizzi IP & il numero delle porte per dirigere il segmento al appropriato socket.

# Demultiplexing senza connessione (UDP)

Richiamo:

* quando crei un socket, devi specificare il numero di porta locale del host:

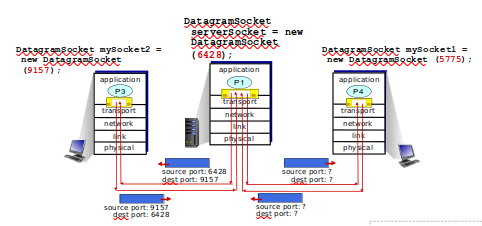
Esempio:

DatagramSocket mySocket = new DatagramSocket(12534);

* quando crei un datagramma per mandarlo dentro il socket UDP, devi specificare:
  + l’indirizzo IP di destinazione;
  + il numero di porta di destinazione.
* Quando l’host ricevente riceve il segmento UDP:
  + controlla il numero di porta di destinazione nel segmento;
  + direziona il segmento UDP al socket con quel numero di porta

Altrimenti:

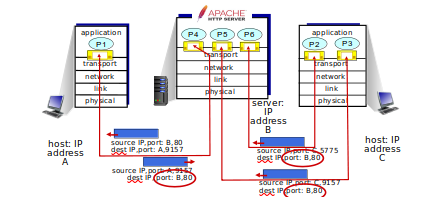
I datagrammi UDP/IP con stessa destinazione, numero di porta, ma con differente fonte di indirizzi IP e/o differente fonte del numero di porta saranno direzionati allo stesso socket all’host ricevente.

Quando creiamo un socket quello che facciamo è che utilizzeremo il socket 1 per mandarlo ad un’altro socket.

# Connessione-orientata al demultiplexing (TCP)

Nel caso del TCP dobbiamo individuare la connessione (IP e Porta), IP destinazione e IP sorgente.

* Usiamo 4 tupla:
* IP sorgente;
* IP destinazione;
* Numero porta sorgente;
* Numero porta destinazione.
* demux: il ricevente usa tutti i quattro valori (4-tupla) per direzionare il segmento al appropriato socket;
* il server potrebbe supportare molti socket TCP simultanei:
  + ogni socket è identificato dal sua propria 4-tupla;
  + ogni socket è associato con una diversa connessione del client.



# Riassunto

* Multiplexing, demultiplexing: sono basati sul segmento, sui valori dei campi headers del datagramma:
* UDP: demultiplexing usa (solo) il numero di porta di destinazione;
* TCP: demultiplexing usa la tupla da 4: indirizzi IP fonte e destinazione, e il numero delle porte;
* Multiplexing/demultiplexing succedono in tutti i livelli.

# UDP: User Datagram Protocol

Nel protocollo UDP, prende un pacchetto e se non ci sono problemi arriva alla destinazione, se c’è invece una perdita il pacchetto non arriva a destinazione.

UDP è connectionless, significa che non ci sono accordi (no handshaking) tra ricevente e mandante, ed ogni segmento UDP viene maneggiato in modo indipendente dai altri.

Perché c’è l’UDP?

* Non c’è da stabilire una connessione (che può aggiungere un ritardo del RTT.);
* semplice: non c’è lo stato della connessione ne del ricevente ne del destinatario;
* piccola dimensione dell’header;
* non c’è il controllo della congestione
  + l’UDP può “esplodere” alla velocità desiserata;
  + può funzionare alla faccia della congestione.

Il TCP è molto limitato nelle performance sul HTTP, perciò alla fine si è scoperto che è meglio usare altri protocolli che usano il protocollo UDP: es. le app streaming multimediali, DNS, HTTP/3.

L’UDP viene usato da:

* le app streaming multimediali;
* DNS
* SNMP
* HTTP/3

Usando questi protocolli ti aggiungerà l’affidabilità e il controllo della congestione a livelli applicativo.

# UDP: Azioni del livello di trasporto

Azioni UDP del mittente:

* è passato un messaggio nel livello applicativo;
* determina i valori dei campi header del segmento UDP;
* crea il segmento UDP;
* passa il segmento all’indirizzo IP.

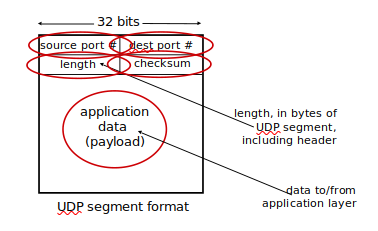
Azioni UDP del destinatario:

* riceve il segmento dall’indirizzo IP;
* controlla il valore header del segmento UDP con il checksum;
* estrae il messaggio dal livello applicativo;
* demultiplexa il messaggio al applicazione via socket.

Simple Network Management Protocol (SNMP).

# Header del segmento UDP

Il segmento header UDP è composto da: porta sorgente(16 bit), porta destinazione(16 bit), la lunghezza, il checksum (per controllare se il pacchetto che è arrivato è giusto), application data.



# UDP checksum

L’UDP checksum ha come unico obbiettivo quello di trovare gli errori e non quello di correggerli.

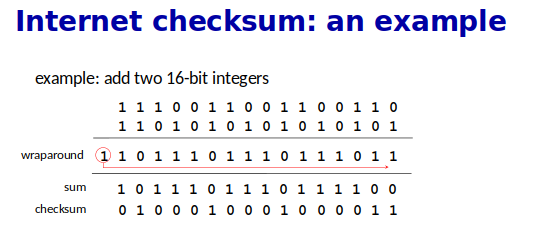
Il checksum di Internet è molto debole, dato che può succedere che anche se c’è un’errore sui bit, quei bit si swappano in maniera tale che sono corretti per il checksum e quando questo accade, per esempio ad una persona arriva un file e possiamo dire che se lo trova corrotto.

Mittente:

* tratta il contenuto del segmento UDP come sequenza di 16-bit interi;
* checksum: aggiunta del contenuto del segmento
* il valore del checksum viene messo dentro il campo del checksum UDP.

Ricevente:

* calcola il checksum del segmento ricevuto;
* controlla se il checksum calcolato equivale al valore del campo del checksum:
  + Non uguale – errore trovato;
  + Uguale – nessun errore trovato.



Prima di tutto vengono sommati i 16-bit del segmento ricevuto rispetto al valore che è stato messo nel campo del checksum, il risultato di questa somma sarà il wraparound, se cè un bit in eccesso viene messo sotto il ris. del wraparound e viene fatta la somma; che determinerà il risultato della sum.

Ed infine il checksum deriva dal flipping dei bit della sum (quindi avviene un complemento ad uno).

# Principi di affidabilità nel trasporto dei dati

Stiamo astraendo l’affidabilità al canale per darla al servizio. Nel mondo reale parlare di un canale affidabile al 100% è quasi impossibile.

Se di norma il canale è affidabile a sua volta anche le applicazioni lo sono, però nella realtà il canale è inaffidabile, quindi dobbiamo rendere il servizio affidabile, per farlo si può costruire sia da lato sender che da receiver dei algoritmi che si occupano di capire se il pacchetto che è arrivato è giusto, di capire se uno dei pacchetti è stato perso e poi tenere in atto azioni necessari per garantire l’affidabilità.

Il sender e il receiver non conoscono lo stato di uno e dell’altro, esempio il messaggio è arrivato?

A meno che gli venga comunicato via messaggio.

Costruiremo un protocollo che possa risolvere tutti i problemi che dà Internet nel momento in cui viene mandato qualcosa in rete.

**Interfacce**

Dalla parte del trasmittente c’è l’implementazione del rdt (reliable data transfer protocol -> protocollo sull’affidabilità del trasporto dei dati) e anche da parte del ricevente c’è questo protocollo.

Prima da parte del trasmittente viene chiamata la funzione rdt\_send() il dato viene passato per consegnare al ricevente fino al livello più in alto; udt\_send() viene chiamato dal rdt per trasferire il pacchetto nel canale inaffidabile al ricevente; viceversa accade dalla parte del ricevente ma si hanno come funzioni rdt\_rcv() che riceve il pacchetto e deliver\_data() che è chiamato dal rdt per consegnare il dato al livello più alto. Il canale inaffidabile molto spesso è bidirezionale.

Ogni processo conosce il proprio senza saper minimamente lo stato di un’altro processo.

La reliable data transfer protocol(rdt) chiamata delle applicazioni, vengono fatte tutte le operazioni dovute e poi viene mandato tutto nella rete tramite unreliable data transfer protocol().

Il cambiamento dello stato deriva da un evento, nel caso delle reti c’è lo stato di un protocollo, arriva questo messaggio lo leggo e poi viene cambiato lo stato del protocollo; stato di descrizione usato sia dai protocolli che dai algoritmi.

Appunti 13/04/22

# Affidabilità nel trasporto dei dati: iniziamo

Andremo a:

* In modo incrementale svilupperemo la parte del trasmittente e dal ricevente del protocollo (rdt);
* Considereremo solo uni-direzionalmente per il trasferimento dei dati;
  + ma le info sul controllo fluiranno da entrambe le direzioni!
* Useremo delle macchine a stati finiti per specificare il ricevente e il trasmittente.

Stato: quando si è in uno stato il prossimo sarà determinato da un evento.

L’evento causa la transizione di stato, poi ci saranno delle azioni prese durante la transizione di stato.

# Rdt1.0: trasferimento affidabile oltre ad un canale affidabile

* il canale sottostante è perfettamente affidabile: quindi senza errori di bit e senza perdita di pacchetti;
* Separiamo FSMs (Finite State Machine = Macchina a Stati Finiti) per il mittente e il ricevente: il mittente manda i dati nel canale sottostante e il ricevente legge i dati mandati dal canale sottostante.



# Rdt2.0: canale con errori di bit

* Il canale in questione potrebbe flippare dei bit del pacchetto: il checksum serve ad individuare i bit di errori
* La domanda è: come recuperiamo i messaggi con gli errori?
* Acknowledgementes (ACKs): il ricevente dice in modo esplicito al mittente che il pacchetto ricevuto è OK
* Negative Acknowledgementes (NAKs): il ricevente dice al mittente che il pacchetto ha degli errori;
* Il mittente ritrasmette il pacchetto quando riceve un NAKs

stop and wait

il mittente manda un pacchetto, poi aspetta di ricevere un risposta

# Rdt2.0: FSM specifiche tecniche

Facoltativo ? Maybe yes

Il trasmittente aspetta per una chiamata da sopra, quando arriva crea il pacchetto in cui avviene un checksum, manda il pacchetto e si mette in attesa di un ACK o di un NAK, se avviene un NAK il trasmittente ritrasmette il pacchetto a cui è avvenuto un’errore durante la ricezione, mentre se avviene un ACK è tutto apposto e si ritorna ad aspettare una chiamata da sopra.

Il ricevente aspetta il pacchetto dal data transfer, appena arriva controlla se ha un’errore o meno, se da un’errore chiede al trasmittente di ritrasmetterlo, se invece è tutto apposto manda un ACK al trasmittente e ritorna allo stato di prima, quindi aspetta che il trasmittente mandi un’altro pacchetto.

# Rdt2.0 ha un difetto fatale!

Attenzione! Non tutto ciò che vedi è bello, può succedere che cmq si spacca tutto se il pacchetto di ACK o NAK siano corrotti e lì è un problema.

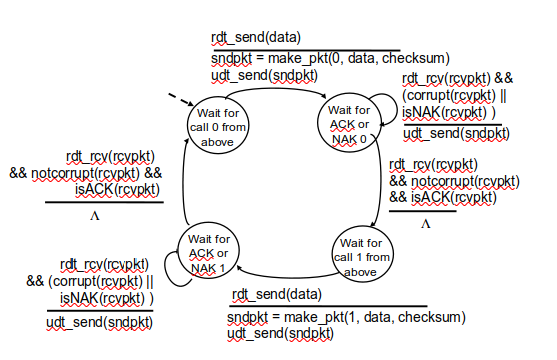
Perché il mandante non sa cosa è successo al ricevente e non può solo ritrasmettere: ci possono essere delle duplicazioni.

# Rdt2.1: Risoluzione ACK/NAKs corrotti

**Maneggiare i duplicati:**

Il mittente ritrasmette il pacchetto corrente se ACK/NAK è corrotto, per capire se è stato rimandato o meno il mandante aggiunge una sequenza di numeri ad ogni pacchetto e perciò il ricevente scarta i pacchetti duplicati (lo capisce dal numero).

# Rdt2.1: mittente, maneggiando i ACK/NAKs alterati



Come funziona:

Primo stato: Il mittente aspetta per la chiamata 0 da sopra

durante il cambiamento di stato:

manda il dato, crea il pacchetto con (0, il dato, il checksum), manda il pacchetto nel canale.

Secondo stato: aspetta per l’ACK o NAK 0; se riceve una risposta in cui dice che il pacchetto è corrotto o un NAK, ritrasmette il pacchetto, altrimenti avviene un...

cambiamento di stato (c’è solo se il pacchetto non è arrivato corrotto e non riceve un NAK):

no operation durante il cambiamento di stato.

Terzo stato: aspetta per la chiamata 1 da sopra

cambiamento di stato:

manda il dato, crea il pacchetto con (1, il dato, il checksum), manda il pacchetto nel canale.

Quarto stato: aspetta per l’ACK o NAK 1; se riceve una risposta in cui dice che il pacchetto è corrotto o un NAK, ritrasmette il pacchetto, altrimenti avviene un...

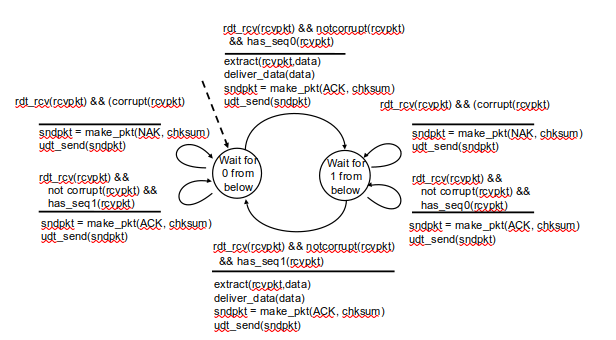
cambiamento di stato (c’è solo se il pacchetto non è arrivato corrotto e non riceve un NAK):

no operation durante il cambiamento di stato.

Si ritorna alla primo stato.

Ciò che cambia rispetto a prima è che prima di mandare il pacchetto, il trasmittente inoltre a fare un checksum aggiunge un numero al pacchetto. Ed in entrambi i casi se il NAK/ACK non sono corrotti si prosegue al prossimo pacchetto, se invece è corrotto si aspetta la ritrasmissione.

# Rdt2.1: ricevente, maneggiando i ACK/NAKs alterati



Come funziona:

Primo stato: il ricevente aspetta per lo 0 da sotto; se riceve il pacchetto ed è corrotto, manda un pacchetto di NAK oppure riceve il pacchetto, non è corrotto ma ha ricevuto un pacchetto con il numero 1, manda un pacchetto per chiedere una ritrasmettere.

Durante il cambiamento di stato:

significa che il pacchetto è stato ricevuto, non corrotto e con la sequenza del pacchetto è uguale a 0; estrae il pacchetto ricevuto, lo consegna nel livello applicativo, manda un pacchetto di ACK nel canale.

Secondo stato: il ricevente aspetta per l’1 da sotto e cosi via.., viene fatto il procedimento fatto prima.

# Rdt2.1: discussione

Mittente:

* numero della sequenza aggiunta al pacchetto;
* due numeri di sequenza (0,1) è sufficiente.

Perché?

* Deve controllare se è stato ricevuto un ACK/NAK corrotto;
* due volte come molti stati;
  + lo stato deve “ricordare” se “è previsto” un pacchetto dovrebbe avere un sequenza di numeri 0 o 1.

Ricevente:

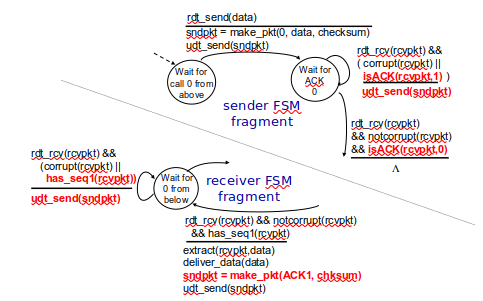
* deve controllare se è stato ricevuto un pacchetto duplicato;
  + lo stato indica se è 0 o 1 è aspettato il numero della sequenza;
* nota: il ricevente non può sapere se il suo ultimo ACK/NAK ricevuto è OK al mittente.

# Rdt2.2: a NAK-free protocol

* Usa la stessa funzionalità del rdt2.1, usando solo ACKs
* Invece del NAK, il ricevente manda ACK per l’ultimo pacchetto ricevuto l’OK: il ricevente deve esplicitamente includere la sequenza del pacchetto essendo ACK
* duplica l’ACK al risultato del mittente nella stessa azione come NAK:

ritrasmittendo il pacchetto corrente

Questo approccio lo usa molto spesso il TCP.



# Rdt3.0: canali con errori e perdita

Il checksum, la sequenza di numeri, gli ACKs, la ritrasmissione aiutano, ma non abbastanza.

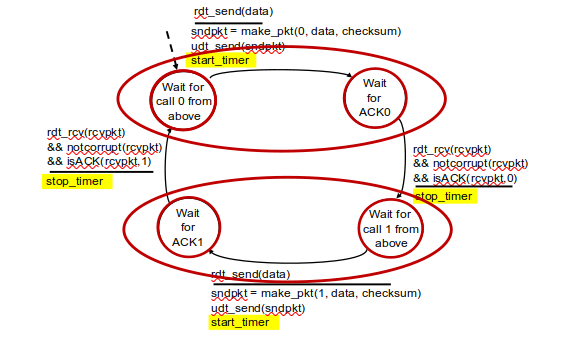
Approccio: il ricevente aspetta un ragionevole ammontare di tempo per l’ACK

* Ritrasmettere se non viene ricevuto nessun ACK in questo tempo;
* Se il pacchetto o l’ACK è solo in ritardo: la ritrasmissione sarà duplicata, il ricevente deve specificare la sequenza del pacchetto per essere conosciuto.

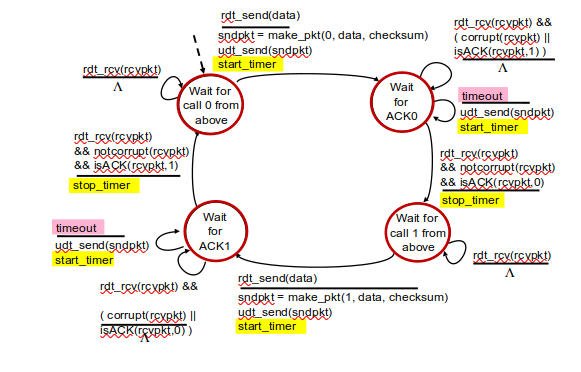
Usa un timer che fa il countdown dopo un ragionevole ammontare di tempo.

Appunti 20/04/2022

# Rdt3.0 lato mittente



Nel momento che siamo in attesa di un ACK0, se il timer dà un timeout si dà scontato che il pacchetto o l’ACK non è arrivato e quindi si procede a ritrasmettere il pacchetto; stessa cosa accade nell’ACK1.



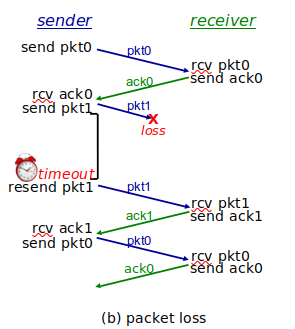
# Rdt3.0: in azione

Mandato pacchetto 0, ricevuto pacchetto 0, appena arriva il ricevente spedisce l’ACK 0, il trasmittente lo riceve e così via per i prossimi pacchetti.

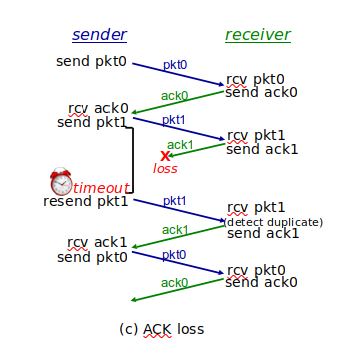
Nei esempi in figura la pendenza delle freccette determina il tempo trascorso tra macchina trasmittente e ricevente.

N.B.: il timeout può avvenire sia per una perdita di un pacchetto che da un’ACK.

Se avviene una perdita sulla trasmissione di un pacchetto, viene ritrasmesso quest’ultimo.

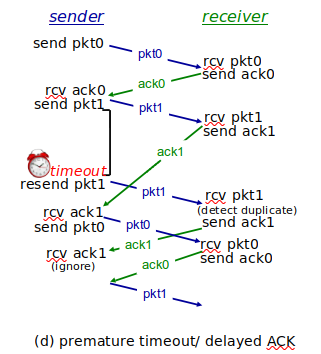


Se invece la perdita accade su un’ACK, il trasmittente ritrasmette lo stesso pacchetto, il ricevente capisce che gli è arrivato un pacchetto duplicato e rimanda l’ACK.



Come ultima casistica può avvenire che avvenga un prematuro timeout e avvenga un ritardo dell’ACK. Nel momento in cui viene ricevuto il pacchetto al ricevente, quest’ultimo invia l’ACK n [ l’ACK n non viene perso ma ritarda la sua trasmissione ], a quel punto nel momento in cui l’ACK n ritarda, il trasmittente rimanda il pacchetto, il ricevente capisce che è una copia nello stesso istante viene ricevuto dal trasmittente l’ACK n ritardatario, ed infine quando il ricevente rimanderà l’ACK n triggerato della resend del pacchetto di prima, quando lo riceverà il trasmittente lo ignorerà perché starà (ormai) già aspettando l’ACK n+1.

**Nota Giockiana: forse si capisce meglio cosa accade guardando l’esempio disegnato.**

****

# Performance del rdt3.0 (stop-and-wait)

Utilizzo del trasmittente: è la frazione di tempo il trasmittente è occupato nel mandare il pacchetto.

Esempio: 1 Gbps collegamento, 15 ms di ritardo di propagazione, pacchetto di 8000 bit.

* Tempo di trasmissione del pacchetto nel canale:

D (trans) = L / R = 8000 bits / 10^9 bits/sex = 8 microsec

# Rdt3.0: l’operazione di fermare e aspettare

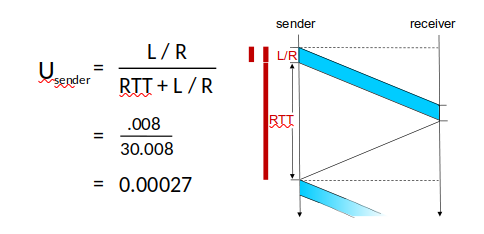
In base ai calcoli fatti prima un pacchetto per arrivare al ricevente ci vogliono 8ms, poi prima che arrivi l’ACK ci vogliono altri 8ms, RTT = 8ms + 8ms = 16ms.

Alla fine si è calcolato che l’utilizzo del trasmittente è di 0.00027% pochissimo, quindi il protocollo limita le performance nel canale, perciò c’è bisogno di migliorarlo.

Come viene calcolato il tempo fatto sull’operazione stop-and-wait;

Il primo pacchetto di bit viene trasmesso al tempo = 0, quando sono arrivati tutti i bit del pacchetto il ricevente manda l’ACK e poi il mittente riceverà il seguente ACK; il tempo che va dall’invio del messaggio (dal mittente) fino al arrivo dell’ACK (al mittente) determina il RTT.

Quindi il prossimo pacchetto verrà mandato al tempo = RTT + L / R.



# Rdt3.0: operazione su protocolli pipelined

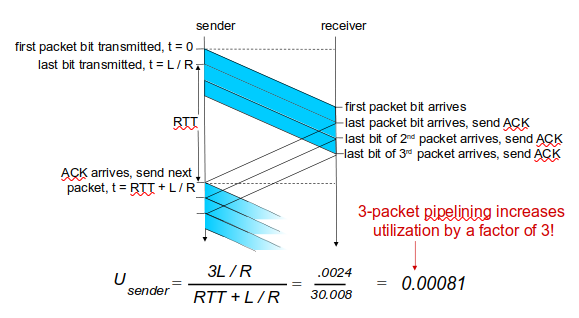
Usando il pipelining: il trasmittente permette multipli, pacchetti ancora da essere conosciuti:

* il range della sequenza di numeri deve essere incrementato;
* buffering al trasmittente e/o al ricevente.

Usando il pipelining le performance del rdt3.0 migliorano.

# Pipelining: incrementa l’utilizzo

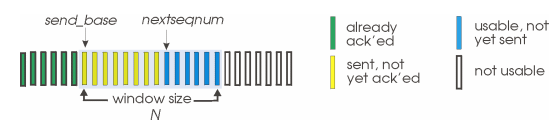
Con il pipelining il mittente manda più pacchetti alla volta anche se non ha ottenuto l’ACK da parte dei precedenti pacchetti; con questa tecnica si vanno a migliorare le performance.



# Go-Back-N: trasmittente

* Da parte del trasmittente: viene scelta una finestra di n pacchetti, di modo che si capisca quali sono i pacchetti già conosciuti, quali sono stati mandati ma non conosciuti, quali usati ma non ancora mandati e quali quelli ancora non usati.

Questa finestra di N pacchetti sono consecutivi trasmessi ma non conosciuti ( “unACKed” ); la finestra mostra i pacchetti non ancora conosciuti e quelli usabili ma non ancora mandati.



* ACK accumulativi: ACK(n): conosce tutti i pacchetti fino a, includendo la sequenza di numeri fino a n;
* nel ricevimento di un ACK(n): muove la finestra in avanti iniziando a n+1.
* timeout(n): ritrasmette il pacchetto n e tutte le più alte sequenze dei pacchetti nella finestra.

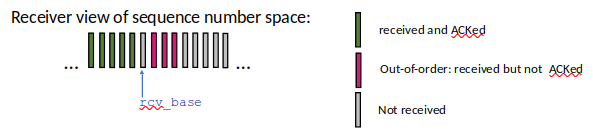
# Go-Back-N: ricevente

Solo l’ACK: manda sempre l’ACK per ricevere correttamente il pacchetto lontano, con il più alto in ordine della sequenza:

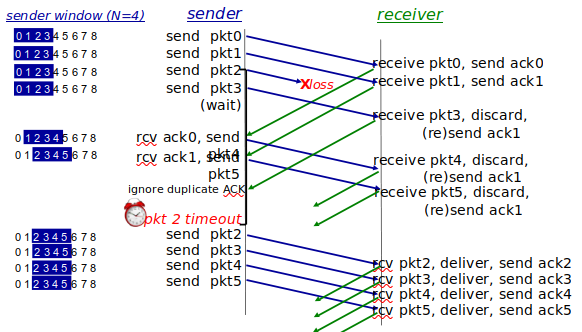
* potrebbe generare ACKs duplicati
* a solo bisogno di ricordare il rcv\_base()

nel ricevimento di pacchetto fuori ordine:

* può scartare (non memorizza) o bufferare: una decisione implementativa;
* re-ACK (ritrasmette l’ACK) il pacchetto con il più alto (numero) in ordine di sequenza di numero.



# Go-Back-N in azione



In sostanza, se per esempio un M pacchetto viene perso e non viene ricevuto l’M ACK, i pacchetti M+1, M+2, M+3, etc.. anche se arrivano e mandano l’ACK del numero del loro pacchetto questi ultimi vengono ignorati dal mittente e vengono ritrasmessi tutti.

# Ripetizione selettiva

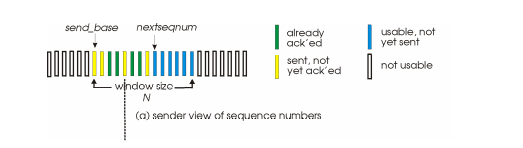
Il ricevente individualmente conosce tutti i pacchetti correttamente ricevuti:

* memorizza i pacchetti , quando ne ha bisogno, per un eventuale consegna in ordine dal livello più alto.

Il trasmittente avrà in modo individuale i times-out e le ritrasmissioni per i pacchetti non conosciuti.

E sempre quest’ultimo ha una finestra, con N sequenze consecutive e limiti di numeri di sequenza da mandare, pacchetti non conosciuti.

# Selective repeat: sender, receiver windows



# Ripetizione selettiva: trasmittente e ricevente

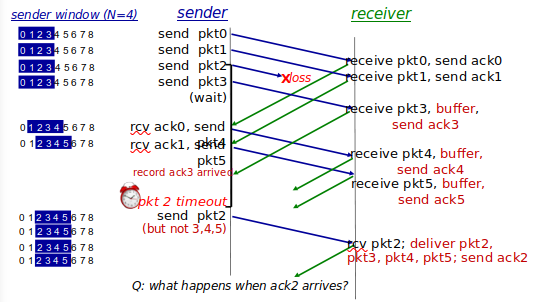
Trasmittente:

* dato (informazione) da sopra:
* se la prossima sequenza di numero disponibile è nella finestra, manda il pacchetto.
* timeout(n):
* rimandare il pacchetto n, restarta il timer;
* ACK(n) tra [sendbase, sendbase+N]:
* marca il pacchetto n come ricevuto;
* se n è più piccolo di un pacchetto sconosciuto, avanza la finestra alla prossima sequenza di numeri non conosciuti.

Ricevente:

* il pacchetto n tra [rcvbase, rcvbase+N-1]:
* manda l’ACK(n);
* se è fuori ordine: memorizza (buffer);
* in ordine: consegna, avanza la finestra al prossimo pacchetto non ancora ricevuto
* pacchetto n tra [rcvbase-N, rcvbase-1]
* ACK(n);
* altrimenti:
* ignora.

# Selezione Ripetitiva in azione



Quindi cosa succede in breve: se una pacchetto n viene perso; tutti i pacchetti successivi ad n (es. n+1, n+2, n+3, etc..) non verranno scartati dal ricevente ma verranno bufferizzati, mandando anche l’ACK. Nel frattempo il pacchetto n verrà ritrasmesso e quando finalmente il pachetto n sarà ricevuto dal ricevente avverrà sia la consegna del pacchetto n che di quelli successivi + verrà mandato l’ACK del pacchetto n.

Appunti 21/04/2022

# Capitolo 3: connessione orientata al trasporto: TCP

TCP viene usato generalmente per lo streaming. È uno dei protocolli più complessi, esempio: gestisce la congestione nella rete.

# TCP: overview

RFCs: 793, 1122, 2018, 5681, 7323

Principalmente queste versioni sono cambiate col tempo per il fatto della congestioni

Comunicazione one to one, da punto a punto, prevede solo una connessione;

Porta i pacchetti da un punto A ad un punto B in modo affidabile ed in ordine, il TCP vede uno stream di byte, li divide in messaggi (segmenti divisi);

Protocollo che permette di comunicare in modo bidirezionale;

Come abbiamo detto divide il stream di byte in segmenti tutti con la stessa dimensione, con una dimensione massima (maximum segment size = MSS).

Usa il cumulative ACKs, lo stesso che viene usato nella famiglia GO BACK end, ha una finestra ha dimensione dinamica lo ha perché in condizioni normali non sappiamo a priori la capacità del canale a disposizione;

è un protocollo orientato alla connessione: prima di inviare i dati, viene creata la connessione tra i due (o canale di comunicazione), un po’ come succede alla chiamate (commutazione a circuito).

Faccenda di flow-controll, esempio: uno ha una sufficiente banda, ma il cellulare non ha la velocità computazionale di star dietro, perciò TCP abilita delle tecniche che per macchine più deboli manda un’informazione (alla pagina) di flusso di protocollo di smettere fino a che glielo dice.

# TCP: struttura del segmento

La dimensione è di 32 bits

* numero di porta della fonte e della destinazione (16 bit ciascuno);
* sequence number: numero di byte che stiamo mandando in formati stream di byte;
* ACK(n): sequenza di numeri del prossimo byte che si aspetta;
* 16 bits contenenti dei flags:
  + lunghezza del TCP header;
  + bit C, E = notifiche sulla congestione;
  + bit A = ACK;
  + bit R, S, F = RST, SYN, FIN controllare/maneggiare la connessione.
* 16 bits finestra da ricevere: flusso di controllo (flow control) = numero di bytes che il ricevente dovrà accettare;
* Internet checksum; (16 bits)
* opzioni (lunghezza varibile, bit messi a disposizione 32)
* dato dell’applicazione, dato mandato dall’applicazione nel TCP socket.

dati passati dall’applicazione (pay load) questi dati in condizioni generali sono abbastanza buoni ad arrivare alla massima dimensione del segmento; meno header ho rispetto ai dati più è efficiente;

c’è il checksum;

ci sono eventuali opzioni, se il pacchetto deve essere

3 bit per la connessione: R S F;

receive window: dice quanti dati il ricevente può gestire nel prossimo pacchetto, se è = 0 il sito si ferma, può controllare quanto spazio hai;

altri 2 bit che indicano lo stato di congestione nella rete: C E.

Campo da 8 bit per l’header;

un bit che serve per i pacchetti urgenti da mandare: U (poco utilizzata).

Sequence number TCP

Sequence number: numero di byte nel byte stream.

Abbiamo una certa finestra:

pacchetti verdi ACK mandati, gialli mandati ma non ancora conosciuti, blu usabili ma non mandati e grigi non usati.

Gli ACKs sono il numero in sequenza che deve essere mandato dopo che è stato mandato il pacchetto e sono comulativi.

Per quanto riguarda gli ACK fuori ordine non sanno (è raro che lo siano) cosa farci.

# TCP sequenza di numeri, ACKs

Utente schiaccia un tasto, viaggia dal host A(mittente) al B(ricevente).

L’utente digita la ‘C’ → manda al ricevente una sequenza = 42, un ACK = 79, con il dato ‘C’;

L’host B riceve la ‘C’ quindi “riecheggia” indietro la ‘C’ → manda al mittente una sequenza = 79, un ACK = 43, con il dato ‘C’;

L’host A riceve l’ACK di “riecheggiamento” ‘C’ → manda al ricevente una sequenza = 43, un ACK = 80.

Semplice telnet scenario

# TCP round trip time, timeout

Come definire il valore del timeout del TCP?

Più lungo del round trip time, ma varia! Se è troppo corto: ci sarà un un timeout prematuro, e ci sarebbero ritrasmissioni non necessarie; se è troppo lungo: abbiamo una reazione lenta per una perdita di un segmento.

Come stimare il round trip time?

Può essere fatto tramite un campione di un RTT: viene misurato il tempo dalla trasmissione di un segmento fino alla ricezione dell’ACK. (viene ignorato il tempo di ritrasmissione se avviene)

Oppure, prendo gli ultimi k stime di timeout, li infilo in una media e uso quello come approssimazione del prossimo timeout.

**Calcolo:**

RTTStimato = (1 – alpha) \* RTTStimato + EsempioRTT.

* Media mobile esponenziale moderata;
* influenza del esempio passato che decrementa esponenzialmente veloce;
* valore tipico: alpha = 0.125.

**L’intervallo del timeout**: è il RTT stimato + “il margine di sicurezza”.

TimeoutInterval = EstimatedRTT + 4 \* DevRTT

Il timeout (in un certo intervallo di tempo)lo settiamo, la stima del RTT + 4 volte la deviazione di questa stima.

Per calcolare la deviazione standard viene usata la media del peso esponenziale del campione medio che deriva dalla stima del RTT.

DevRTT = (1 – beta) \* DevRTT + beta \* val.ass.(SampleRTT – EstimatedRTT)

beta = 0.25

Timeout usato di solito per la ritrasmissione

# TCP trasmittente

Evento: dato ricevuto dall’applicazione

* crea un segmento con una sequenza di numeri;
* la sequenza di numeri è un numero di stream di byte del primo dato in byte nel segmento;
* parte un timer che ci dice quando c’è un timeout (TimeOutInterval), timer settato sul segmento più vecchio.

Evento: timeout

* ritrasmette il segmento che ha causato il timeout e poi resetta il timer.

Evento: ACK ricevuto

* se l’ACK sa i segmenti che precedentemente non sono conosciuti: aggiorna cosa si sa che deve essere ACKed; inizia il timer se ci sono ancora dei segmenti non conosciuti.

# TCP ricevente

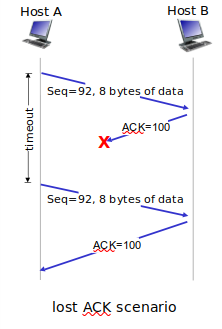
ordine pacchetto che si aspetta quelli conosciuti restano li dove sono.

Il TCP ha un algortimo (nagol algorithm), che quando arriva un pacchetto aspetta 500ms per il prossimo segmento.

# TCP: scenari di ritrasmissione

perdita di un ACK:

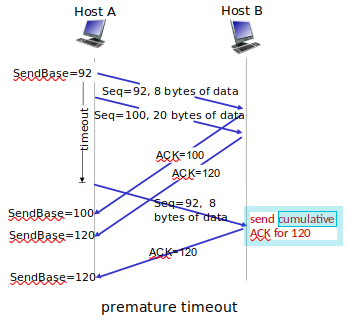
se sta venendo mandato un’ACK (viene perso) e scatta il timeout, verrà ritrasmesso il pacchetto a cui il mittente stava per ricevere quell’ACK.



Timeout prematuro

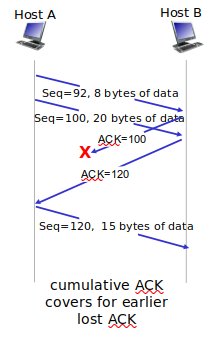
Può accadere che dopo che è arrivato un segmento, il ricevente invia l’ACK prima che arriva al mittente ci impiega un po’ di tempo, nello stesso momento avviene un timeout prematuro di quell’ACK, il mittente ritrasmette lo stesso pacchetto, nel mentre riceve anche l’ACK “ritardatario” e poi il ricevente manda un ACK cumulativo e quest’ultimo verrà ignorato.

Host A manda prima il segmento 92 e anche il 100, il pacchetto 92 prima che venga conosciuto ci mette troppo scatta il timeout troppo presto e viene rimandato il pacchetto.



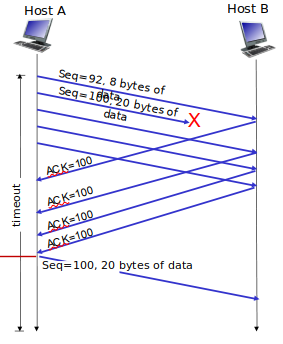
ACK cumulativo che compre per la perdita dell’ACK avvenuta prima

**Commento di Giock: “Non l’ho ancora capito del tutto”**



# TCP ritrasmissione veloce:

mandato primo pacchetto 92, 100 + altri pacchetti, viene ack il 100 non il 120 perché non cè, ad ogni pacchetto che arriva dice che sta aspettando il 100, se arriva 3 volte lo stesso ACK viene fatta una assunzione “perso” (invece che aspettare che avvenga un timeout), ritrasmette il pacchetto per evitare un timeout, male che vada ho sprecato un po’ di banda.



Nel mondo moderno non funziona così bene, anche perché noi di solito siamo in via Wireless.

# Primo servizio flow control: (TCP)

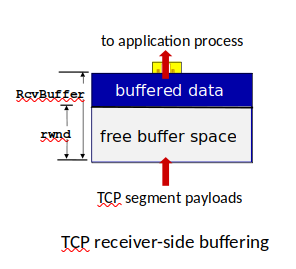
buffer sono buoni per non fare ritrasmettere pacchetti che ci hanno messo molto ad arrivare.

Cosa succede se il livello della rete consegna i dati i modo più veloce rispetto al livello applicativo che rimuove il dato dal socket del buffer?

Nella struttura del segmento si trova nella finestra di ricezione (receive window) => flusso di controllo: numero di bytes che il ricevente è disposto ad accettare

**Flow control:** il ricevente controlla il trasmittente, quindi il trasmittente non potrà mandare in overflow il buffer del ricevente trasmettendo troppo, o troppo velocemente.

**RWND = Receiver WiNDow (Flow Control)**

* Il TCP ricevente “avvisa” che c’è dello spazio libero nel campo **rwnd** del header TCP;
* **RcvBuffer** dimensione settata nelle opzioni del socket (tipicamente è di 4096 bytes)
* al mittente limita l’ammontare di dati (“in-flight”) sconosciuti da essere ricevuto nel **rwnd**;
* garantisce che il buffer del ricevente non vada in overflow.

# TCP: controllo della connessione

Prima di poter scambiare il dato, il mittente e il ricevente fanno un “handshake”:

* fanno un accordo per stabilire la connessione (ognuno sa la volontà dell’altro per stabilire la connessione);
* fanno un accordo sui parametri di connessione.

Esiste una macchina a stati finita che definisce la connessione.

Avviene la così detta stretta di mano: sarebbe uno scambio di pacchetti tra una macchina A e B.

Un problema che può accadere è questo: il cliente chiede la connessione, il server la stabilisce, il cliente “muore”, viene richiesta una riconnessione al server viene stabilita ma alla fine abbiamo una connessione per metà aperta; per il server c’è n’è una mentre per il cliente non c’é proprio; quindi alla fine la 2-way handshake non è che sia tanto buono da usare.

Per ovviare a questo hanno pensato di creare un 3 way handshake, il cliente manda un pacchetto al server (voglio parlare), il server manda un pacchetto al cliente dice che vuole un flag send e un ACK, il cliente manda l’ACK.

# Chiusura di una connessione TCP

* Il cliente, server ognuno chiude la propria parte della connessione
  + manda un segmento TCP con FIN bit = 1
* risponde per aver ricevuto FIN con ACK
  + nella ricezione del FIN, ACK può essere combinato con il proprio FIN
* simultaneamente lo scambio di FIN può essere maneggiato.

# Principi sul controllo della congestione

# TCP: controllo della congestione

**La congestione:**

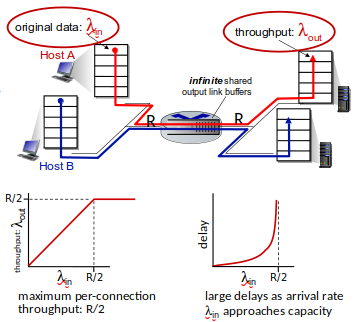
in modo informale accade quando vengono mandate troppe risorse in modo troppo veloce per farsi che la rete lo “maneggi”;

Cosa accade durante la congestione:

* lunghi ritardi (derivante dalle code dei buffer dei routers);
* perdita dei pacchetti (overflow dei buffer dei routers).

Scenario semplice:

ci sono due hosts, sono collegati ad un router con linea di capacità R, abbiamo due flussi di dati concorrenti entrambi gli hosts mandano dei pacchetti con stessa frequenza (intensità di traffico), e non c’è bisogno di ritrasmettere.



Dato che la grandezza della trasmissione è R, al massimo entrambi gli host possono usare una capacità di R/2, se ne usano di più si rischia che il ritardo o da parte di un’host o dell’altro aumenti.

Secondo scenario:

abbiamo un router, con buffers finiti; il trasmittente ritrasmette le perdite, quando avviene un timepout.

Idealizzazione : conoscenza perfetta, il trasmittente manda solo quando il buffer del router ha dello spazio libero/disponibile.

Idealizzazione : con un po’ di conoscenza

* i pacchetti possono essere persi a causa del riempimento del buffer;
* il trasmittente sa quando il pacchetto è stato droppato: ritrasmette solo se sa che il pacchetto è stato perso.

Scenario realistico: duplicati non necessari

* i pacchetti possono essere persi, sempre a causa del riempimento dei buffers – viene richiesta la ritrasmissione;
* ma i tempi del trasmittente può andare in time out prematuramente, mandando 2 copie, di cui entrambi sono consegnate.

L’effetto finale è che il throughtput cala quando mandando un pacchetto a R/2, alcuni pacchetti vengono ritrasmessi, inclusi quelli a cui c’era bisogno perché il pacchetto è stato perso e anche per quelli a cui non c’era bisogno solo perché era scattato il time-out.

Un’altro “costo” della congestione:

quando un pacchetto viene droppato, in ogni trasmissione in upstream e il buffering usato per quel pacchetto è stato sprecato.

# Cause/Costi della congestione: intuizioni

* La capacità di trasmissione non può mai andare oltre la capacità.
* Il ritardo aumenta in base a come approccia la capacità.
* La perdita/ritrasmissione diminuiscono la capacità di trasmissione effettiva.
* La duplicazione non necessaria nel futuro diminuisce il throughput
* La capacità di trasmissione del caricamento/buffering sprecato per i pacchetti persi in scaricamento.

# Approccio sul controllo della congestione

Controllo della congestione end-end:

non viene detto in modo esplicito il feedback dalla rete; la congestione viene dedotta da una perdita osservata, ritardo; approccio preso dal TCP.

Controllo della congestione rete-assistita:

* i routers provvedono un feedback diretto agli hosts trasmittenti o riceventi con i flussi passando attraverso un router con una congestione;
* potrebbe indicare il livello di congestione o esplicitamente settare il rate trasmittente.

# **TCP controllo della congestione: AIMD**

**AIMD – Additive Increase Multiplicative Decrease**

Il TCP usa come controllo della congestione: AIMD

approccio: i trasmittent possono incrementare il ratio di trasmissione fino a quando avviene una perdita di un pacchetto (congestione), poi decrementano il ratio di trasmissione dopo gli eventi di perdita.

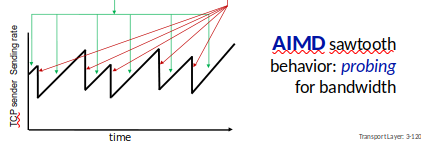
Additive Increase = Incremento Aggiuntivo:

incrementa il ratio di 1 la dimensione massima di un segmento ogni RTT fino a quando viene trovata una perdita.

Multiplicative Decrease = Diminuzione Moltiplicata:

dimezza (taglia a metà) il ratio di trasmissione per ogni evento di perdita.

Quando nel grafico il ratio raggiunge una certa altura prima di essere tagliato, sta facendo un probing della banda, sta vedendo fin dove può arrivare.



# TCP AIMD: more

Multiplicative Decrease dettagli: il ratio di trasmissione è:

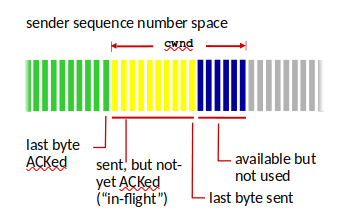
* taglia a metà quando la perdita è derivata dai tre ACK duplicati (**TCP Reno**);
* taglia di 1 mss (maximim segment size) quando la perdita avviene da un timeout (**TCP Tahoe**).

Perché AIMD?

AIMD – un algoritmo distribuito e asincrono – è stato mostrato per:

* ottimizzare il ratio del flusso di congestione per rete estesa!
* Ha desiderabili proprietà stabili.

# TCP controllo della congestione: dettagli



TCP comportamento durante la trasmissione:

in modo grezzo: manda dei bytes della **Congestion WiNDow (cwnd)**, aspetta l’RTT per gli ACKs, poi manda altri bytes.

TCP rate = cwnd/RTT bytes/sec

* TCP limite di trasmissione del mittente: LastByteSent – LastByteAcked <= cwnd;
* la cwnd è dinamicamente aggiustata per rispondere alla congestione della rete osservata.

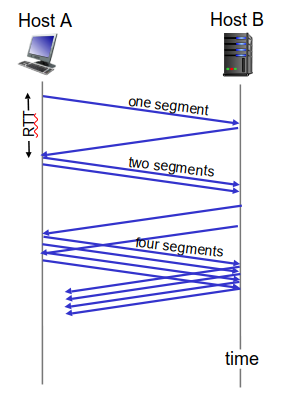
# **TCP slow start (inizio lento)**

Quando la connessione inizia, incrementa il ratio in modo esponenziale fino a che avviene un evento di perdita:

* inizialmente cwnd = 1 MSS;
* raddoppia cnwd ogni RTT;
* tutto ciò viene fatto incrementando la cwnd per ogni ACK ricevuto.

In generale: il ratio è inizialmente è lento, ma accelera in modo esponenziale.

Esempio GRAFICO:



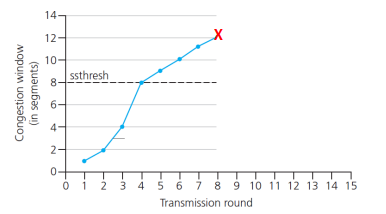
# TCP: da un inizio lento al evitare la congestione (congestion avoidance)

Q: Quando dovrebbe l’incremento esponenziale cambiare in lineare?

A: Quando cwnd ottiene al ½ del suo valore prima del timeout.

Implementazione:

* variabile **ssthresh**;
* Il “**slow start threhold**” è ciò che divide la fase di slow start dalla fase AIMD.
* durante un evento di perdita, **ssthresh** è settato a ½ della cwnd giusto un secondo prima del evento di perdita.



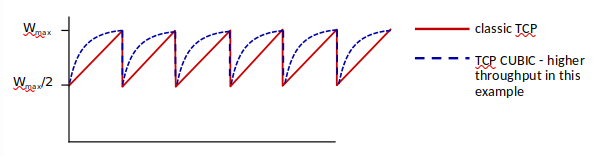
Appunti 28/04/2022

# **TCP CUBIC**

C’è qualche miglior modo rispetto al AIMD per controllare quanta ampiezza della banda è usabile?

Intuizione:

* W(max) : il ratio di trasmissione alla quale è stata rilevata la perdita di congestione;
* lo stato della congestione del limite probabilmente non è cambiato molto;
* dopo aver tagliato il rate / la finestra a metà durante una perdita, inizialmente sale fino a W(max) più velocemente, ma quando si sta avvicinando a W(max) diventa più lento. (La salita avviene in modo parabolico)



K: punto nel tempo quando la dimensione della finestra del TCP raggiungerà W(max).

Incrementa W come una funzione del cubo della distanza tra il tempo corrente e K:

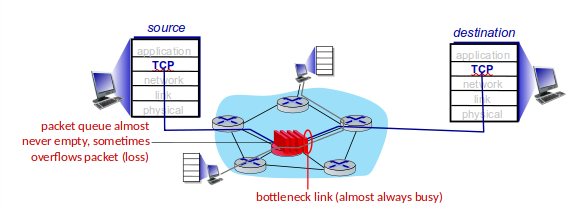
* incrementa maggiormente quando ci si allontana da K;
* incrementa poco quando è più vicino a K.

TCP CUBIC è di default in Linux, è una tra i più popolari TCP per Web servers popolari.

# TCP e il congestionato “collegamento collo di bottiglia”

TCP (classico, CUBIC) incrementa la trasmissione di rate del TCP fino a quando avviene una perdita di pacchetto al output di alcuni routers: nel collegamento “col collo di bottiglia – intasato”.

Capire la congestione: è utile per concentrarsi al collegamento intasato.



Intuizione 1.0: incrementando il ratio di invio TCP non incrementerà la capacità di trasmissione end-end con il collo di bottiglia intasato.

Intuizione 1.1: incrementando il ratio di invio TCP incrementerà la misura del RTT.

Obiettivo: “mantenere il tubo end-end giustamente pieno, ma non del tutto”.

# Ritardo basato sul controllo di congestione del TCP

Mantenendo il tubo da mittente a ricevente “abbastanza pieno, ma non del tutto pieno”: mantiene il collegamento del collo di bottiglia occupato nella trasmissione, ma evita alti ritardi/buffering.

Throughtput misurato = numero di bytes mandati nell’ultimo intervallo del RTT / RTT misurato

* Controllo della congestione senza forzatura di perdita;
* massimizzando il throughput (“mantenendo il tubo che si è appena riempito”) mentre si mantiene basso il ritardo (“… ma non riempito del tutto”);
* un numero di TCPs impiegato prende un approccio basato sul ritardo.

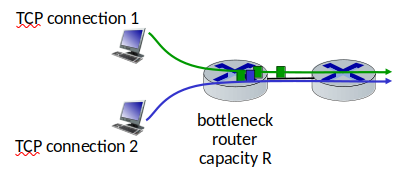
# Esplicitare la notificazione della congestione (ECN)

Gli impieghi del TCP spesso implementano una rete assistita o che assiste sul controllo della congestione:

* 2 bits nel IP header marcati dal router del network per indicare la congestione
* indicazione della congestione che importa al destinatario
* la destinazione setta dei ECN bit nel segmento del ACK per notificare al trasmittente della congestione
* coinvolge entrambi sia l’IP (IP header marcando l’ECN bit) che il TCP (TCP header marcando C, E bit).

# TCP fairness (equità)

Obiettivo della giustizia: se K TCP sezioni condividono lo stesso collegamento del collo di bottiglia della larghezza della banda R, ognuno dovrebbe avere il rate medio di R/K.



Il TCP è fair?

Esempio: ci sono 2 sezioni TCP che stanno competendo:

* l’incremento additive da uno pendenza di 1, come incremento del throughtput;
* il multiplicative decremento decrementa il throughtput in modo proporzionale.

# Giustizia: devono tutte le apps della rete essere “giuste” ?

Giustizia e UDP

Diverse apps spesso non usano TCP:

* non vogliono rate strozzato dal controllo della congestione.

Invece usano UDP:

* mandare audio/video ad un ratio costante, hanno una tolleranza bassa sulla perdita dei pacchetti.

Non c’é “Polizia Internet” controllo nel uso del controllo della congestione.

Giustizia, connessioni TCP parallela

* L’applicazione può aprire multiple connessioni parallele tra 2 hosts.
* Web Browsers fanno questo, esempio: collegamento del rate R con 9 connessioni esistenti.
  + Nuova app chiede per 1 TCP, ottiene un rate R / 10.
  + Nuova app chiede per 11 TCPs, ottiene R / 2.

# Evoluzione sulle funzionalità del livello di trasporto

* TCP, UDP: sono i principali protocolli di trasporto per 40 anni;
* differenti “favoriti” del impiego del TCP, per specifici scenari:
* muovendo le funzioni del il livello di trasporto al livello applicazione, in cima al UDP: HTTP/3: QUIC è un protocollo di Google.

# QUIC: Quick UDP connessioni di Internet

È un protocollo a livello applicativo, in cima al UDP:

* incrementa le performance del HTTP;
* impiegato da molti servers di Google e apps.

Addotta l’approccio che abbiamo studiato in questo capitolo per stabilire la connessione, controllo dei errori, controllo della congestione.

* Errore e controllo della congestione;
* stabilimento della connessione: affidabilità, controllo della congestione, autenticazione, encryption, stato stabilito in un RTT

TCP (affidabiltità, stato del controllo della congestione) + TLS (autenticazione, stato crypto) = 2 strette di mano.

QUIC: affidabilità, controllo della congestione, autenticazione, stato crypto = 1 stretta di mana.

Appunti del 11/05/22

# Switching via memoria

Prima generazione di routers:

* tradizionali computers con lo switching sotto diretto controllo di CPU;
* il pacchetto viene copiato nella memoria di sistema;
* la velocità è limitata dalla banda di memoria (2 bus che attraversano per datagramma).

# Switching via bus

* il datagramma va della memoria della porta input alla memoria della porta output via bus condiviso;
* bus contention: switchando la velocità è limitata per banda del bus;

# Switching via rete interconnessa

* crossbar, o altre interconnessioni sono innestate inizialmente sviluppate per connettere i processori in multiprocessori;
* switch multistage: nxn switch da multipli stage di più piccoli switch;
* exploiting parallelism: - frammenta il datagramma in una lunghezza fissa delle celle in entrata; - switch cells attraverso la fabbrica, ri-assembla il datagramma in uscita;
* scalando, usando multipli switching in parallelo:
* speedup, scaleup via parallelismo;

# Buffer Management:

* Drop: quale pacchetto aggiungere, droppa quando i buffers sono pieni:
* drop coda: droppa quando arriva il pacchetto;
* priorità: droppa o rimuove on base alla priorità
* marcando: a quali pacchetti segnalare la congestione

# Packet Scheduling: FCFS

Lo scheduling del pacchetto: decide quale prossimo pacchetto mandare nel collegamento:

* il primo che arriva, è il primo ad essere servito
* priorità
* round robin
* in base alla pesantezza della coda

FCFS: i pacchetti sono trasmessi in base all’ordine di arrivo nella porta output:

* anche chiamato: il primo ad entrare è il primo ad uscire (FIFO).

# Scheduling policies: priority

Priorità dello scheduling:

* classificato in base all’arrivo di traffico, classificato per classe: ogni campo header può essere usato per la classificazione;
* manda il pacchetto dalla coda di più alta priorità che ha i pacchetti memorizzati: FCFS all’interno della classe di priorità.

# Scheduling policies: round robin

Scheduling Round Robin (RR) :

* classificato in base all’arrivo di traffico, classificato per classe: ogni campo header può essere usato per la classificazione;
* il server in modo ciclico e ripetitivamente scansiona le code della classe, mandando un completo pacchetto da ogni classe in turno.

# Scheduling policies: weighted fair queueing

Weighted Fair Queuing (WFQ):

* generalizza l’RR;
* ogni classe, i, ha un peso, w j piccolo, e ottiene l’ammontare della pesantezza in ogni ciclo: Wj / sommatoria Wj
* un minimo della larghezza della banda è garantito;

# Livello della rete: Internet

Host, router funzioni del livello della rete:

Dentro il livello della rete ci sono gli algoritmi sulla selezione del percorso: implementati nei protocolli del rounting o nel controller SDN;

C’é il protocollo IP:

* formato del datagramma
* inidirizzamento;
* le convenzione di maneggiamento dei pacchetti.

C’é il protocollo ICMP:

* reporting dei errori;
* “segnalazione” del router;

# IP Datagram format

Si trovano campi da 32 bits in cui nel primo:

Abbiamo dei campi riservati per:

* il numero della versione del protocollo IP;
* la lunghezza dell’header;
* il tipo di servizio: diffsery, ECN
* la lunghezza totale del datagramma;

nel secondo:

* ci sono diversi campi per la frammentazione e il riassemblamento

nel terzo:

* il tempo di vita;
* il livello più in alto (esempio: TCP o UDP);
* l’header del checksum

ed infine nei ultimi quattro livelli:

* l’indirizzo IP della sorgente (32 bits);
* l’indirizzo IP della destinazione (32 bits);
* le opzioni;
* il “pagamento del caricamento” del dato.

Overhead = quanto spendo per il payload

# IP fragmentation/reassembly

I collegamenti hanno un MTU (Maximum Trasmission Unit – massima trasmissione per unità) – che è il più largo collegamento-livello del frame

# IP adressing: introduction

IP sta per Internet Protocoll.

Indirizzo IP: i 32-bit identifica una interfaccia di rete per ogni host o router;

Interfaccia: è la connessione tra host/router e il collegamento fisico:

* i router tipicamente hanno multiple interfacce;
* gli hosts tipicamente ne ha uno o due di interfacce.



# Subnets

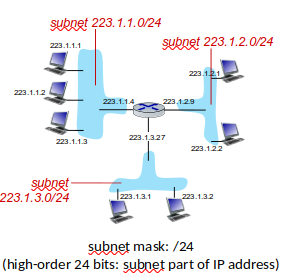
Cosa è un subnet?

È un dispositivo delle interfacce che può fisicamente raggiungere ognuno senza passare attraverso un router intervening.

Gli indirizzi hanno una struttura:

una parte di un subnet: disposto nello stesso subnet hanno comunemente un alto ordine di bits;

una parte del host: rimane poco ordine nei bits.



Recipiente per la definizione dei subnets:

stacca ogni interfaccia dal il suo host o dal router, creando “isole” di reti isolate:

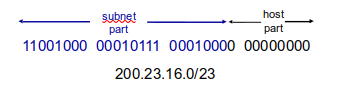
ogni rete isolata è chiamata subnet

cercare cosa serve la subnet mask

# IP addressing: CIDR

CIDR: Classless InterDomain Routing

* la porzione del subnet del indirizzo di una lunghezza arbitraria;
* formato dell’indirizzo: a.b.c.d/x, dove x è il numero di bits nella porzione del subnet del inidirizzo.



# IP addresses: how to get one?(1)

Come fa l’host ha ottenere l’indirizzo IP?

* Codificato dall’amministrazione di sistema in un file di configurazione;
* DHCP: Dynamic Host Configuration Protocol: dinamicamente ottiene l’indirizzo come server

# DHCP: Dynamic Host Configuration Protocol

Obbiettivo: l’host dinamicamente ottiene l’indirizzo IP dal server della rete quando “entra” nella rete

* può rinnovare la locazione dell’indirizzo in uso;
* permette il riuso dei indirizzi;
* supporto per gli utenti mobili che entrano/lasciano la rete.

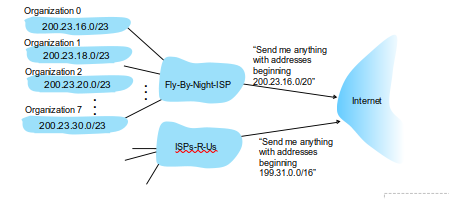
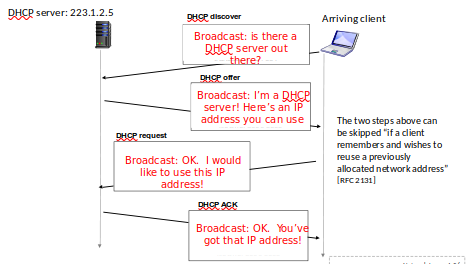
Anteprima DHCP:

* l’host trasmette un messaggio di scoperta DHCP [opzionale];
* il server DHCP risponde con un messaggio d’offerta DHCP [opzionale];
* l’host richiede l’indirizzo IP: messaggio di richiesta DHCP;
* il server DHCP manda l’indirizzo: messaggio di conoscenza DHCP.

# DHCP client-server scenario

Tipicamente, il server DHCP sarà localizzato in un router, servendo tutti i subnets di cui router è attaccato.

Sta arrivando il cliente DHCP e ha bisogno dell’indirizzo di questa rete.



# DHCP: more than IP addresses

DHCP può darti molto più che dove è allocato l’indirizzo IP nel subnet:

* l’indirizzo del primo salto router per il cliente;
* nome e indirizzo IP del server DNS;
* la maschera della rete.

# IP addresses: how to get one? (2)

Seconda domanda: come fa la rete ad ottenere una parte del subnet del indirizzo IP?

Ottiene una porzione allocata dal suo fornitore dallo spazio d’indirizzo del ISP.

Blocco del ISP: 11001000 00010111 00010000 00000000 200.23.16.0/23

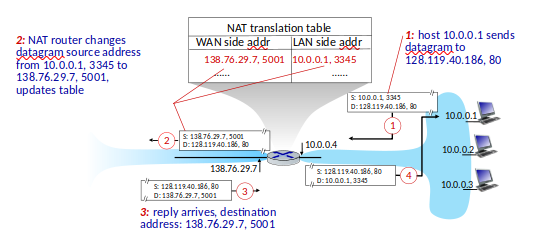
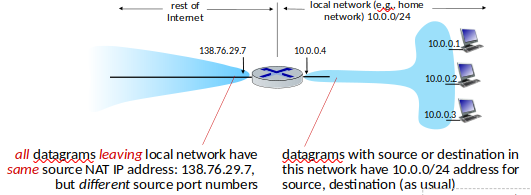
ISP può allocare fuori i suoi spazi dell’indirizzo in 8 blocchi.

# Hierarchical addressing: route aggregation

L’indirizzamento gerarchico permette efficiente pubblicità del informazione del routing.

- L’organizzazione 1 si muove dal Fly-By-Night al ISPs-R-Us;

- ISPs-R-Us adesso pubblicizza un più specifico cammino all’organizzazione 1

* network address translaction
* IPv6

# NAT: network address translaction

NAT: tutti i dispositivi in una rete locale condivide solo UN indirizzo IPv4 per quanto riguarda il mondo esterno

Tutti i dispositivi in una rete locale hanno indirizzi a 32-bits in uno spazio dei indirizzi IP (privato) che può solo essere usato nella rete locale.

Vantaggi:

* hai bisogno solo di UN indirizzo IP dal fornitore ISP per tutti i dispositivi;
* puoi cambiare gli indirizzi del host nella rete locale senza farlo sapere al resto del mondo;
* sicurezza: i dispositivi all’interno della locale net non sono indirizzabili in modo diretto, visibile fuori dal mondo.

Implementazione: il router del NAT deve:

* i datagrammi uscenti: rimpiazzo (fonte dell’indirizzo IP, numero della porta) di ogni datagramma uscente al (indirizzo IP del NAT, nuova numero della porta). I Clienti e Servers remoti risponderanno usando (l’indirizzo IP del NAT, nuovo numero della porta) come indirizzo di destinazione;
* ricorda (in una tavola di traduzione NAT) ogni (fonte dell’indirizzo IP, numero della porta) al (indirizzo IP del NAT, nuova numero della porta) coppia di traduzione;
* datagrammi che stanno arrivando: rimpazza (l’indirizzo IP del NAT, nuovo numero della porta) in campi destinazione di ogni datagrammo che sta arrivando con il corrispondente (fonte dell’indirizzo IP, numero della porta) immagazzinato nella tavola del NAT.

# IPv6: motivazione

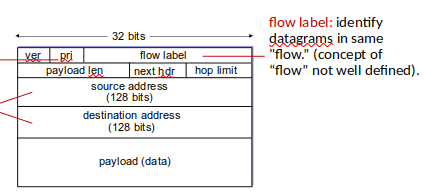
* Iniziale motivazione: 32-bit lo spazio dell’indirizzo dell’IPv4 potrebbe essere completamente allocato;
* motivazione addizionale:
  + velocità nel processare/andare avanti: 40-bytes di “riparazione” nella lunghezza dell’header;
  + abilita la differenza sul trattamento del livello della rete sul “flusso”

# IPv6 formato del datagramma

All’indirizzo dell’IPv6 vengono dati 128 bit.

Cosa manca rispetto all’IPv4:

* non c’è il checksum;
* non c’è la frammentazione/riassemblamento;
* non ci sono le opzioni (disponibili come livello più alto)



# Transizione da IPv4 a IPv6

Come fa la rete a operare con routers mischiati (IPv4 e IPv6)?

Attraverso il tunneling: il datagramma IPv6 gli importa come caricamento (payload) nel datagramma IPv4 tra i routers IPv4.

Il tunnelinfg è spesso usato in modo più ampio in altri contesti (4G/5G).

**Appunti in più:**

L’IPv6 non ha il checksum e non fa la frammentazione.

Il congestion avoidance dimezza la finestra.

Il “slow start threhold” è ciò che divide la fase di slow start dalla fase AIMD.

Il protocollo ARP (Address Resolution Protocol) serve per conoscere il MAC address, una volta noto l’indirizzo IP di destinazione.

Il CRC (Cyclic Redundancy Check) è il metodo di calcolo del checksum.

Il checksum ha lunghezza fissa.

Il router ha i primi 3 livelli (fisico, data link e network); mentre gli switch ha solo i primi 2 livelli (fisico e data link).

L’architettura SDN è l’architettura di un server centralizzato e di solito ha un controller logico per il control plane (il control plane si occupa del disegnamento della topologia della rete).

Il protocollo che viene utilizzato per proteggere i dati HTTPS è SSL.

Il protocollo che invia messaggi di errore ed è utilizzato dai comandi ping e tracert è ICMP.

Domanda: Quando si progettano i servizi di trasporto, è necessario assicurarsi che:

Risposta: Siano verificate le affermazioni elencate.

Domanda: Una comunicazione si dice affidabile quando:

Risposta: Tutte le affermazioni si devono verificare contemporaneamente.

Domanda: TCP stabilisce una sessione con un processo :

Risposta: three way handshake.

Domanda: La programmazione dei Socket è un importante elemento nello sviluppo delle

applicazioni di rete. Per un Socket, quali delle seguenti affermazioni non sono

valide:

Risposta: Nessuna delle affermazioni sopra elencate.

Domanda: Quale delle seguenti affermazioni **NON È CORRETTA** per il codice CRC (Cyclic

Redundancy Check)?

Risposta: Il CRC può rilevare errori ed essere utilizzato per l’integrità dei dati.

Domanda: Il protocollo IPv6 manda solo pacchetti criptati per sicurezza.

Risposta: False

Domanda: Il Protocollo TCP Implementa:

Risposta: nessuna delle risposte è corretta.

Domanda: Si consideri la figura: una connessione TCP e considerata fair se:

Risposta: Nessuna delle risposte é corretta ? Maybe o lascia in bianco