



Università
degli Studi
della Campania
Luigi Vanvitelli

Reti di Calcolatori e Cybersecurity

Livello Trasporto - UDP

Ing. Vincenzo Abate

Livello trasporto

In una rete di computer, il compito del livello trasporto è quello di consentire alle molteplici applicazioni eseguite in un end-system lo scambio di informazioni attraverso il servizio offerto dal livello rete.

Consente alle applicaz. di scambiarsi informaz.
Ho programmi esecuz. che devono comunicare.

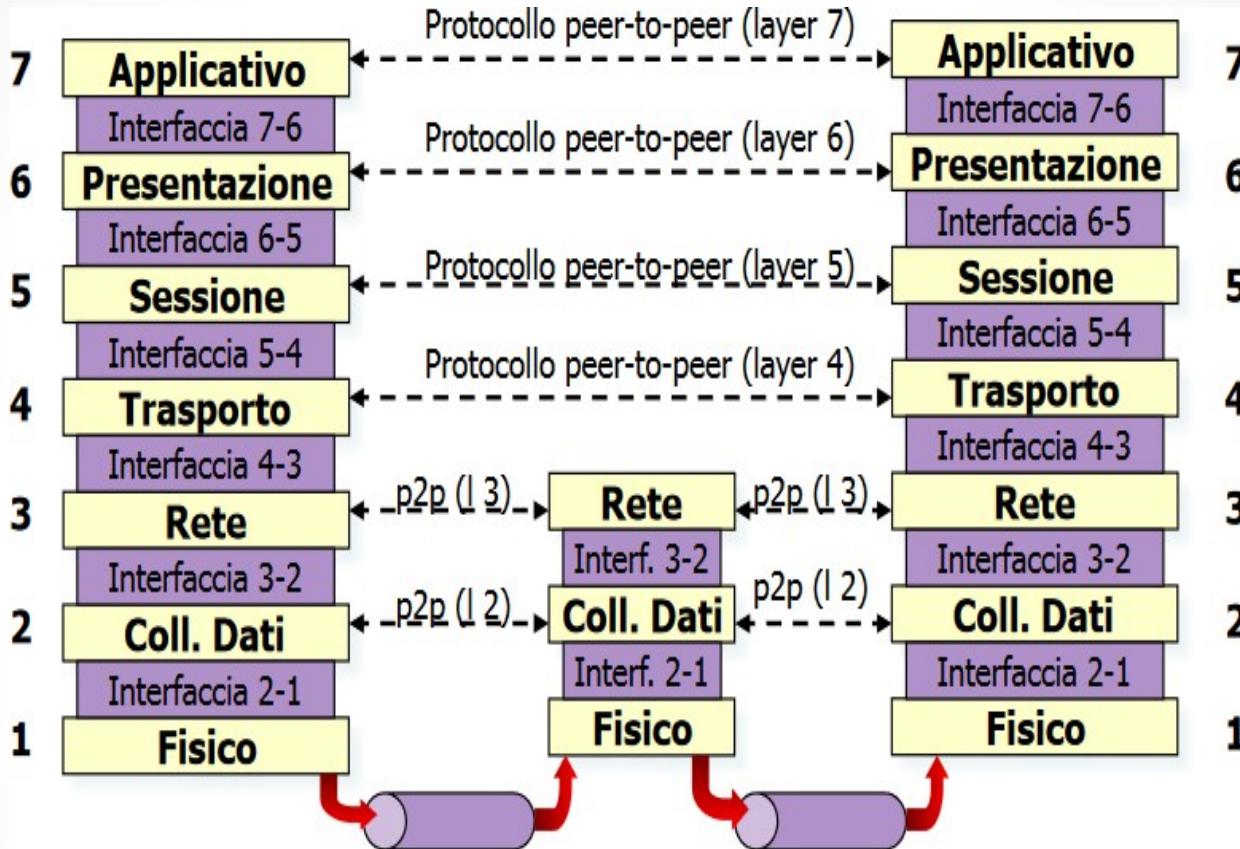
uso
del livello
trasporto

Protocollo



Livello trasporto

agisce solo



Il livello **trasporto** agisce solo negli end-system

In uno stesso momento, in un terminale possono essere attivi molteplici flussi di

comunicazioni *Flussi di comunicaz. tra applicaz. => App su un host parla ad app su un altro host.*
Ciascuno di questi flussi sarà trattato distintamente dal livello trasporto

Livello trasporto

- Offre un canale di comunicazione logica tra applicazioni attive su differenti host
- Differenze tra livello trasporto e livello rete:
 - Network-layer: trasferimento dati tra end-system
 - Transport layer: trasferimento dati tra processi.
Naturalmente necessita dei servizi offerti dal livello rete.

↑ A livello Rete

Da trasporto da host a host a trasporto da processo a processo

App devono parlare fra loro → macchine che parlano fra loro →

Livello trasporto

Livello Trasporto
Livello Rete
Livello Data-Link
Livello Fisico



In particolare:

- controllo degli errori
- sequenza ordinata
- controllo di flusso
- controllo di congestione



Affidabile

Non Affidabile *

Il livello rete offre un servizio inaffidabile, quindi:

Il livello trasporto deve ovviare:

- aumentando l'efficienza
- aumentando l'affidabilità

Obiettivo:

Isolare i livelli superiori dai problemi dovuti all'uso di differenti tecnologie di rete e dalle loro (eventuali) imperfezioni

* Prob. che andrà bene

Livello trasporto

I protocolli di Livello Trasporto sono realizzati al di sopra del Livello Rete, quindi è necessario gestire:

- Apertura della connessione (setup)
- Memorizzazione dei pacchetti all'interno della rete
- Un numero elevato di connessioni ...
 - Multiplexing e Demultiplexing

↳ Cioé, delle info che arrivano al livello rete vengono passate alla questa socket.
Al contrario arrivano al corretto destinatario.

Multiplexing e Demultiplexing

I diversi programmi in esecuzione concorrente all'interno di un terminale (end-system) sono detti processi (nella terminologia dei sistemi operativi)

Il compito del livello trasporto è quello di consegnare i messaggi applicativi contenuti nei pacchetti che arrivano dal livello rete al processo al quale sono destinati **DEMULTIPLEXING**

Questa funzione di smistamento (demultiplexing) è esplicata al livello trasporto attraverso un'informazione ausiliaria contenuta nell'header dei protocolli di livello trasporto: i **port number** A livello nello uso numero ip. A livello trasporto vedi come consegnare le cose al corretto processo al numero di port

Multiplexing e Demultiplexing

Demultiplexing: inoltrare i segmenti ricevuti al corretto processo cui i dati sono destinati

Multiplexing:

Raccogliere i dati provenienti dalle applicazioni, imbustare i dati con un header appropriato (per il de-multiplexing)

L'host riceve datagrammi IP:

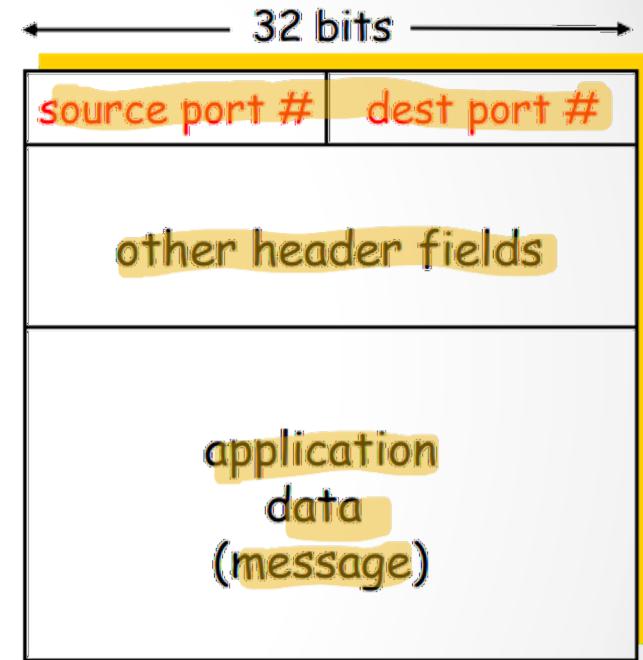
ogni datagramma ha Indirizzo IP sorgente, IP di destinazione

info supplementare

ogni datagramma contiene 1 segmento del livello di trasporto

ogni segmento ha numero di porta di origine e destinazione

L'host utilizza indirizzi IP e porta numeri per indirizzare il segmento verso la socket appropriata



TCP/UDP segment format

Demux Connectionless

Protocollo UDP; demux viene effettuato così

Socket UDP identificata da 2-tupla:

Per id. destinatario

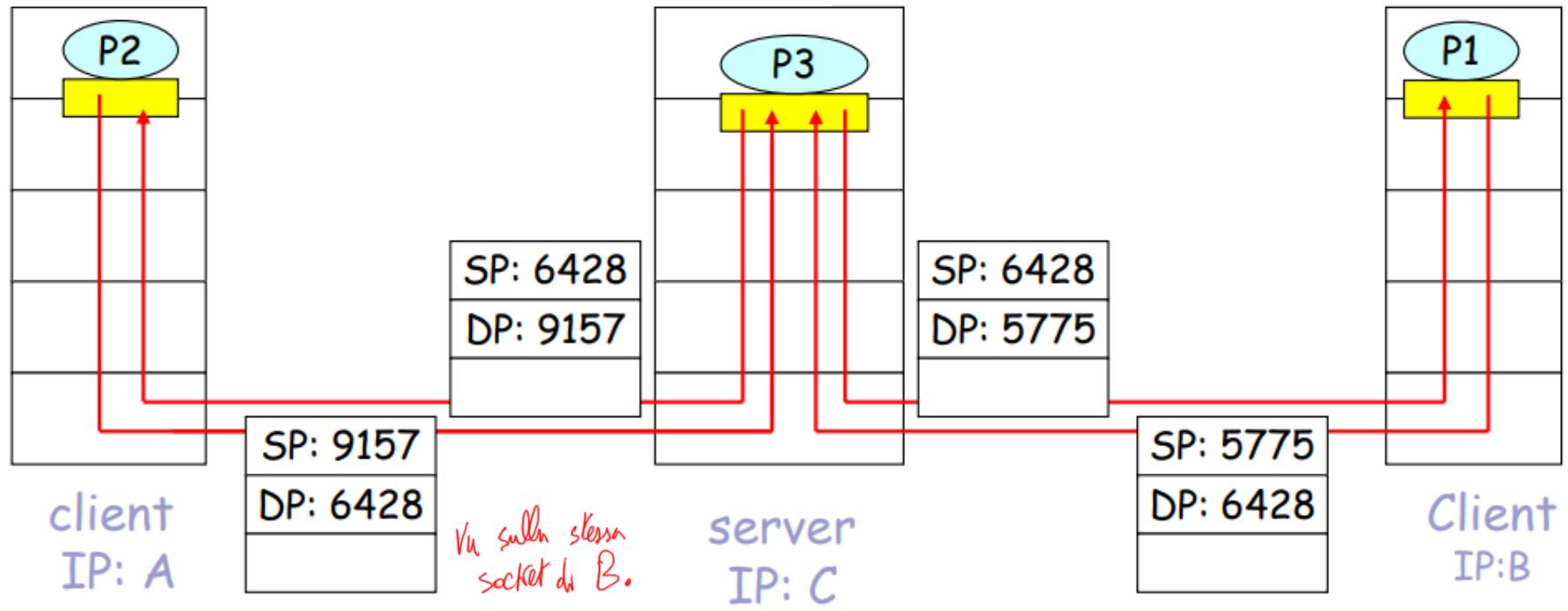
(indirizzo IP di destinazione, numero di porta di destinazione)

Quando l'host riceve un segmento UDP:

- controlla la porta di destinazione nel segmento
- indirizza il segmento UDP alla socket con quel numero di porta
Socket: processo di esecuzione

I Datagrammi IP con IP di origine e/o numeri di porta sorgente diverso sono diretti verso la stessa socket

Demux Connectionless



SP fornisce « l'indirizzo di ritorno »

Con UDP non ho altro modo.
UDP non è ovviato alla connessione
Non m'importa niente se chi è m'ha

Demux Connection-oriented

La socket TCP identificata con 4 tuple:

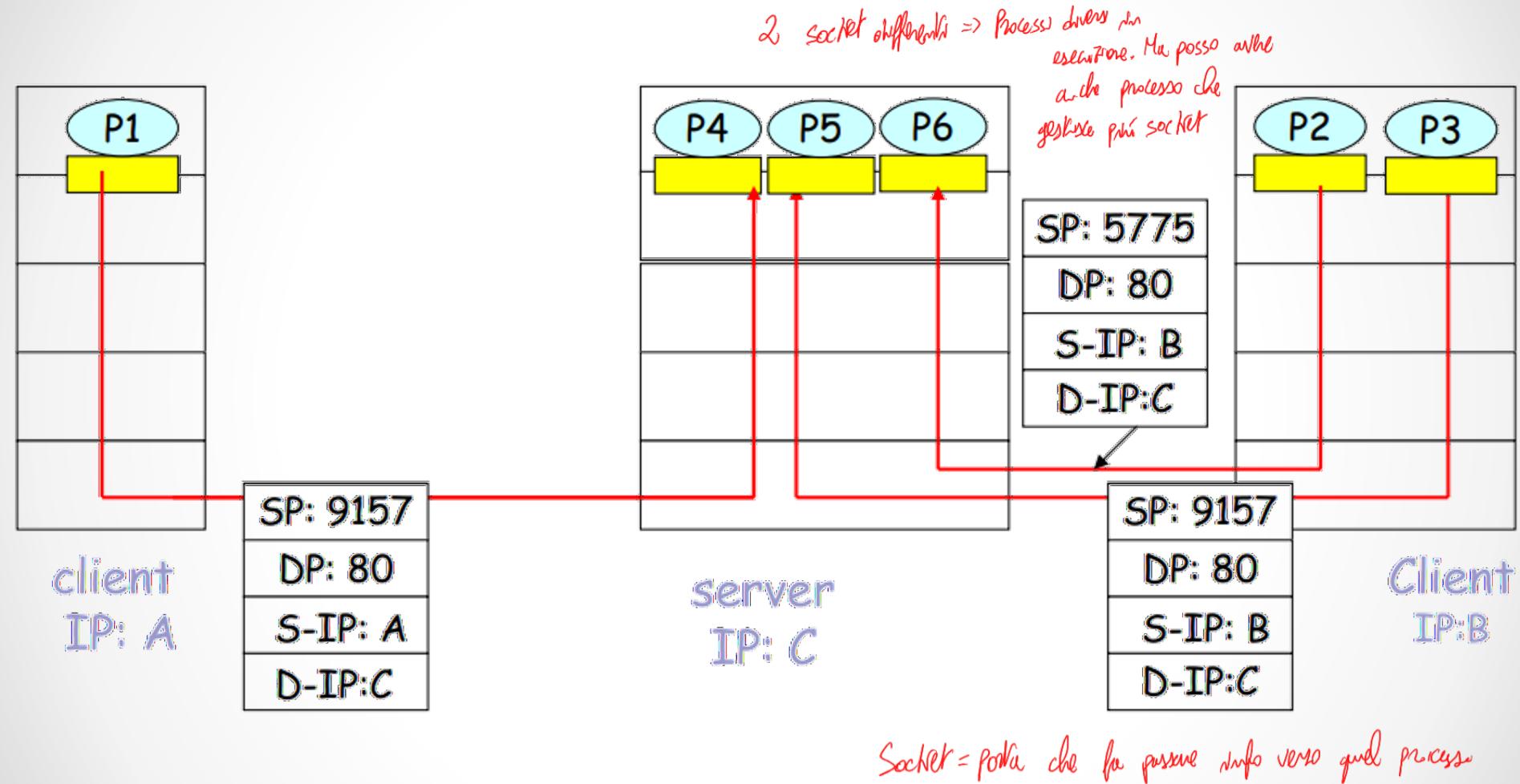
Devo gestire connessioni

(IP origine, porta di origine, IP destinazione, porta di destinazione)

pknew

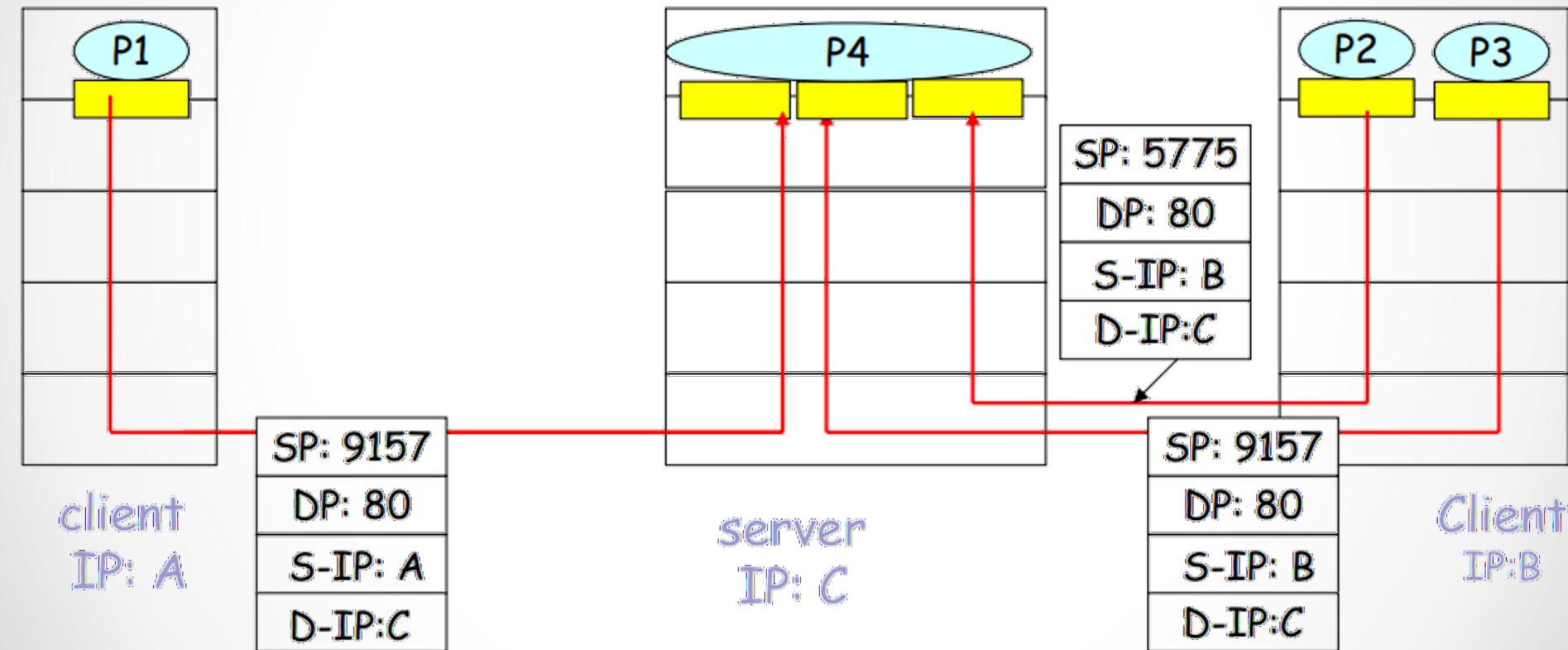
- L'host `recv` utilizza i quattro valori per dirigere segmento verso la socket appropriata
- L'host del server può supportare diverse socket TCP simultanee: ciascuna identificata dalla propria tupla a 4
- I server Web hanno socket diverse per ogni client
- HTTP non persistente avrà diversa socket per ogni richiesta

Demux Connection-oriented



Socket = porta che fa passare info verso quel process.

Demux Connection-oriented Threaded Web server



UDP

Nasce come protocollo semplice

Protocollo di trasporto Internet "senza fronzoli"

- Servizio "**best-effort**", con UDP i segmenti possono essere:
 - perduto ↳ Fa quello che deve fare al meglio che può
 - consegnato a app non in ordine
- senza connessione: nessun handshaking tra mittente UDP e destinatario
- ogni segmento UDP gestito indipendentemente da altri

Perché è stato introdotto UDP?

- non è necessaria la fase di inizializzazione (setup) che introduce delay (Es: DNS è basato su UDP) ↳ APP che deve avere bassa latenza. UDP asincrona
- semplice: sender e receiver non devono conservare informazioni di stato (stateless)
- intestazione di dimensioni contenute: basso overhead
- controllo della congestione assente: le applicazioni possono inviare alla velocità desiderata
 - utile per alcune applicazioni
 - rischioso per la rete

↑ Satanna la banda

POSITIVA e NEGATIVA

↳ Sempre legato al tipo di app: UDP utile per streaming

Non ho controlli particolari
App scelgono la velocità (no controllo)

UDP

Ampiamente usato per applicazioni multimediali:

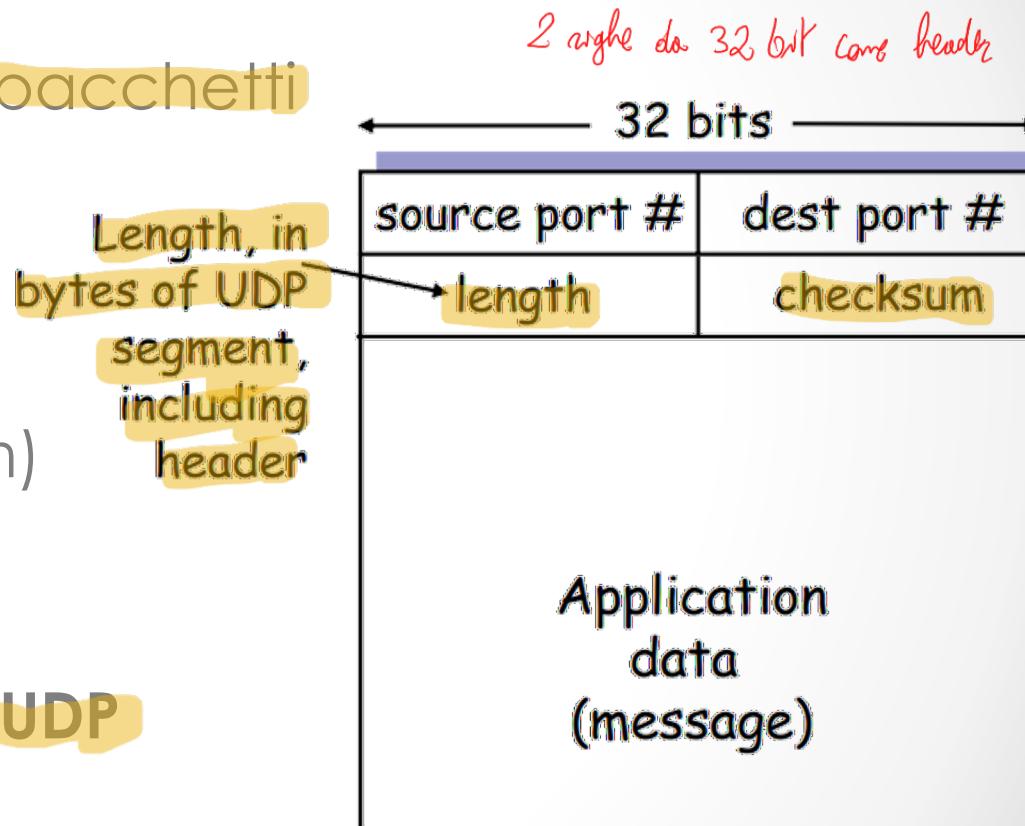
- tolleranti alle perdite di pacchetti
- sensibili ai ritardi

Altre applicazioni:

- DNS
- NFS (Network File System)
- SNMP (Simple Network Management Protocol)

Domanda: si può rendere UDP affidabile?

- Gestione degli errori
- Conferma di avvenuta ricezione



Checksum UDP

Campo da 16 bit che viene calcolato; mittente prende pacco e lo calcola sommando.

Obiettivo: rilevare "errori" (bit alterati) nel segmento trasmesso

Mittente:

Tratta il contenuto del segmento come una sequenza di interi espressi su 16 bit

checksum: complemento ad 1 della somma (in complemento ad 1) dei numeri da 16 bit che costituiscono il segmento UDP ^{NOT}

Incluso uno pseudoheader che include gli indirizzi IP sorgente e destinazione

Il mittente pone il valore della checksum nel campo checksum del segmento UDP

Ricevente:

calcola la somma in complemento ad 1 dei campi del segmento ricevuto compresa la checksum

Risultato composto da tutti 1 ?



NO: errore!

Pacchetto arriva con la checksum:

Somm tutto + checksum = 2xchecksum
complemento

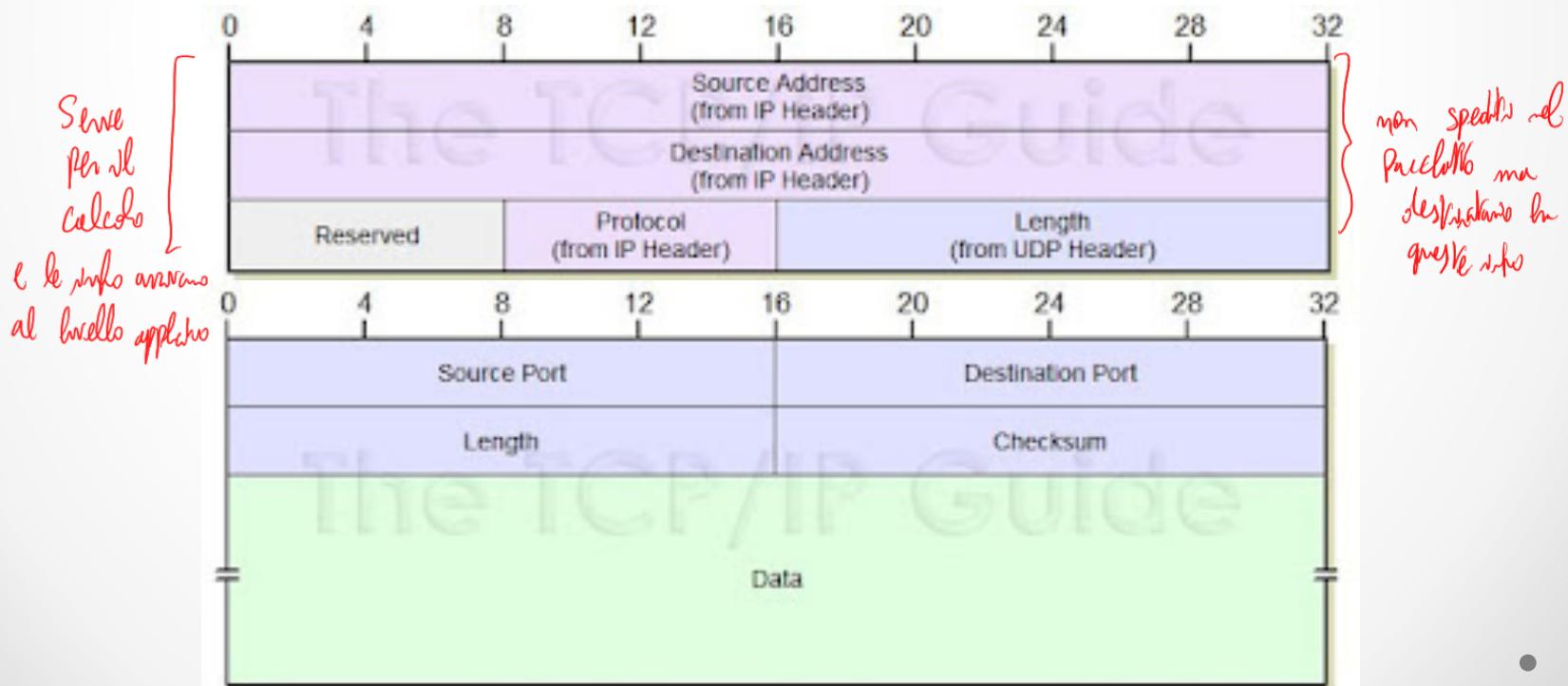
SI: nessun errore rilevato... il che non vuol dire che non vi siano stati errori ...

Non viene mandato, serve per calcolo

Checksum UDP: Pseudo Header

La checksum viene calcolata anteponendo al datagramma UDP uno pseudo-header che contiene gli indirizzi IP sorgente e destinazione e la lunghezza del datagramma

Lo pseudo-header (come dice il nome) viene aggiunto ai soli fini del calcolo della checksum, non viene trasmesso



Checksum UDP: Esempio

Quando si sommano i numeri, potrebbe essere necessario considerare un riporto

Esempio: somma due numeri interi a 16 bit

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Workaround necessario altrimenti avremmo in risultato più di 16 bit (non ammissibile per una checksum)

6bit da riporto. Il riporto in meno significativo e somma

Checksum UDP: Esempio

Pseudo header starts here			
	Decimal	Binary	Hex
Source IP	192.168 0.31	1100 0000 1010 1000 0000 0000 0001 1111	C0 A8 00 1F
Destination IP	192.168. 0.30	1100 0000 1010 1000 0000 0000 0001 1110	C0 A8 00 1E
Reserved/UDP protocol	0/17	0000 0000 0001 0001	00 11
Padding/Length	0/10	0000 0000 0000 1010	00 0A
Pseudo header ends here so we will add the real UDP header to this			
UDP Source Port	20	0000 0000 0001 0100	00 14
UDP destination Port	10	0000 0000 0000 1010	00 0A
UDP Length	10	0000 0000 0000 1010	00 0A
UDP Data	Hi	0100 1000 0110 1001	48 69
Now that we have all that information let's add			
		1 1100 1010 0011 1001	1 CA 39
Notice in our previous entry our values exceed 16 bits (2 bytes). This will not work since our checksum has to be 16 bits. To get to 16 bits we will expand the results from 1 to become 32 bits. Thus we will prepend hex 000 to 1 CA 39. We will also find the binary value of 000 and add it to the binary column.			
		1 1100 1010 0011 1001	00 01 CA 39
Now that we have the 32 Bit value we take the upper half 00 01 and add them to the lower half CA 39			
		0000 0000 0000 0001 + 1100 1010 0011 1001	00 01 + CA 39
		1100 1010 0011 1010	CA3A
We're getting there. Now that we have the above value, we need to find its one's complement. To do this we interchange the 0s and the 1s of the result above			
		0011 0101 1100 0101	35 C5