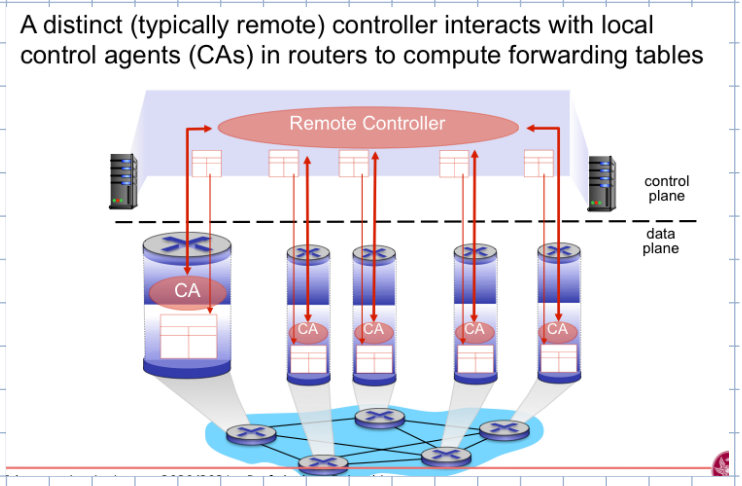
SOFTWARE DEFINED NETWORKING (SDN)

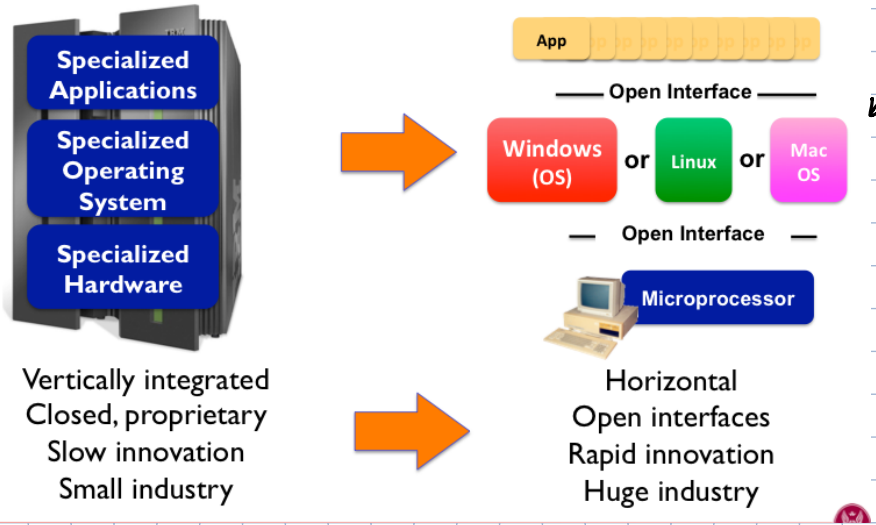
Abbiamo visto internet come un insieme di router interconnessi e ciascun router fa inoltro dei pacchetti e fa instradamento (insieme agli altri router provvede a creare e poi ad aggiornare le tabelle di inoltro).
Con SDN viene centralizzato il processo di interazione tra i router.



L'SDN separa la parte di inoltro da quella decisionale. Quest'ultima parte viene astratta in un remote controller (un'entità centralizzata) che raccoglie le informazioni di tutta la rete, dentro ogni router c'è un software che si chiama agent.

Il motivo principale per centralizzare è la flessibilità. Ad esempio immaginiamo di voler cambiare il criterio con cui viene formulato il problema di ottimizzazione (ossia trovare il cammino con costo minore), come se non volessi minimizzare più la somma dei costi. Questo cambio molto semplice, nell'approccio tradizionale significherebbe modificare il software di ciascun router della rete (la rete nel routing è un sistema autonomo). È oneroso da un punto di vista gestionale che non vengono incentivati i cambiamenti.
Se invece si centralizza tutto e si lascia solo il piano dati sui router (la possibilità di legger le tabelle), basta cambiare solamente il software del remote controlle.

Un esempio pratico è l'evoluzione dei computer:

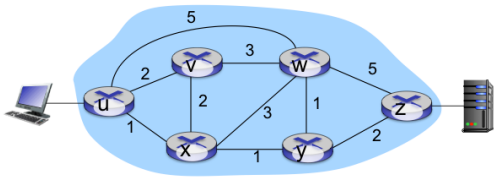


Vengono separati hardware e software in modo tale da incentivare i cambiamenti → questa separazione avviene grazie al sistema operativo

Questo disaccoppiamento ha consentito grandi evoluzioni. Tutto questo non ha reso più "efficienti" i computer ma ha dato la possibilità di far utilizzare a tutti un pc.
Questo discorso di separazione è stato trasferito sulla rete in modo tale da rendere internet è la rete simili all'evoluzione dei pc. In modo da separare il piano dati dalle applicazioni che usa il gestore di rete, questo avviene attraverso SDN.

Traffic engineering:

Cosa bisogna fare affinchè un traffico cambi cammino ? Bisogna cambiare i pesi in maniera tale che si modifichi anche il cammino ottimo.

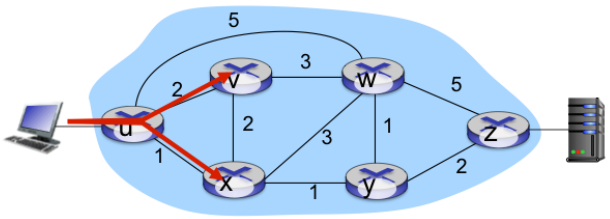


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, x-to-z traffic to flow *xwyz*?

A: need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

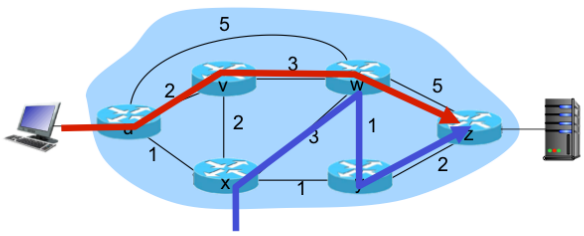
Link weights are only control "knobs": wrong!

Cosa bisogna fare affinchè si faccia load balancing?
Non si può fare con l'instradamento tradizionale a meno che non si crei un nuovo routing algothm.



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

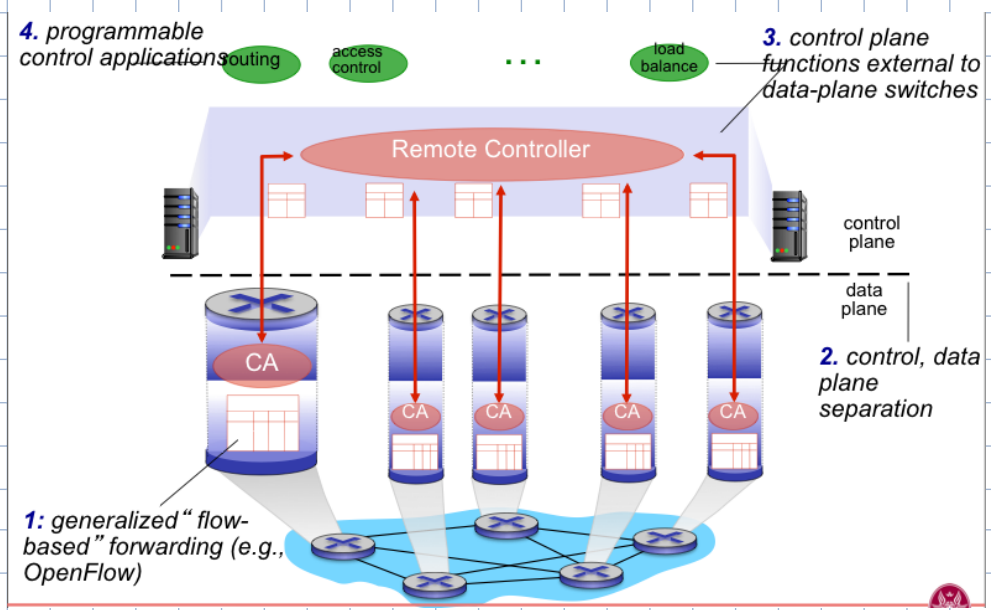
A: can't do it (or need a new routing algorithm)



Q: what if w wants to route blue and red traffic differently?

A: can't do it (with destination based forwarding, and LS, DV routing)

Tutto questo invece si può fare con SDN, è diviso in control plane e data plane.

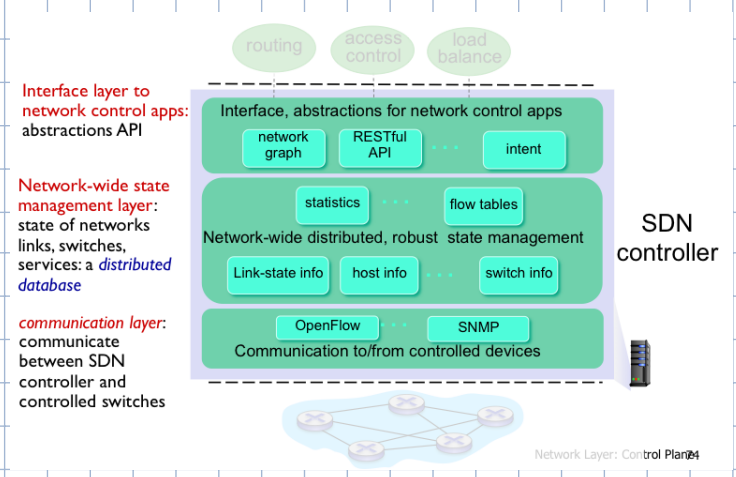


I nodi che fanno solo inoltro si chiama solo switch.

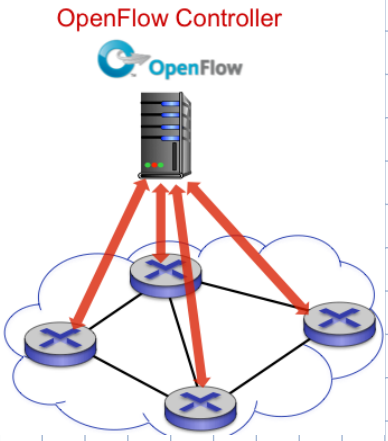
Il piano hardware è formato solo da switch che fanno inoltro generalizzato.
Poi il livello superiore è l'insieme di protocolli di software che presenta al livello superiore un'estrazione degli switch sottostanti.

Nel livello più alto ci sono le applicazioni di rete.

Dialogare con gli switch necessita di un protocollo di comunicazione. Il protocollo per fare il dialogo tra controller e agent si chiama OPENFLOW.



il cuore del sistema operativo consente in una parte di comunicazione (per comunicare con i sistemi operativi) ed è rappresentata dall'open flow,; una parte che contiene strutture dati e sistemi che manipola ossia contiene lo stato della rete
in ogni momento, tutte le info del sistema sono raccolte in una sorta di banca dati che viene aggiornata non appena ci sono notifiche, ci sono tutte le info per avviare il processo di routing.



I messaggi OPENFLOW che vanno dal controllore al switch (packet out) servono a raccogliere le caratteristiche del dispositivo. Altri messaggi OPENFLOW sono: asynchronous (switch to controller,packet-in), symmetric.

Controller to switch messages

- features: raccoglie le caratteristiche del dispositivo.
- Configure: il controllore setta i parametri di configurazione degli switch
- Modify-state: aggiunge, rimuove e modifica i flussi in entrata nelle tabelle di OPENFLOW.
- Packet-out : il controllore può mandare pacchetti specifici agli switch e dirgli cosa fare.

Switch to controller messages

Sono ad esempio:

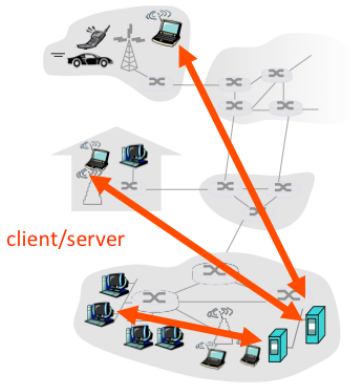
- packet in: vengono trasferiti i pacchetti al controllore. Vede i messaggi dal controllore.
- flow removed
- Port status

Sono dei messaggi di notifica che informano delle modifiche dello stato del sistema.

Il livello di trasporto

Prima facciamo un'introduzione su chi sta al di sopra (ossia la destinazione del trasporto)
Al di sopra c'è il livello applicativo che è di due tipi: peer to peer e client server.

Client-server architecture



server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

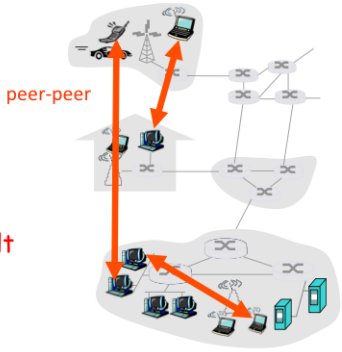
Il servente è sempre connesso e ha un indirizzo statico tale da facilitarne la raggiungibilità. Dà i comandi al client

I clients comunica con il server, può essere connesso a intermittenza, può avere indirizzi IP che cambiano, non comunicano tra di loro direttamente.

P2P

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

Highly scalable but difficult to manage



Ognuno svolge sia il ruolo di servente che di cliente. Un esempio sono tutti i sistemi di comunicazione che utilizziamo (come zoom e skype).

Ci sono delle applicazioni che utilizzano sia p2p che client-server come le app di messaggistica.

Di che tipo si servizio di trasporto ha bisogno un'applicazione?

Requisiti di integrità: l'integrità ci sono delle applicaziomi più tolleranti e altre invece che richiedono un trasferimento integro (come le applicazioni di file).

Requisiti di ritardo: il valore medio di tempo di trasferimento hanno impatto variabile in base al tipo di applicazione. Le applicazioni più rigide sono quelle che comportano il trasferimento di flussi di video, come ad esempio in videochiamata.

Requisiti di portata: quantià di bit trasferita. Alcune applicazioni richiedono una portata minima per essere effettivi, altre app utilizzano qualsiasi portata che gli viene offerto.

Requisiti di sicurezza

Esempio:

Transport service requirements of common applications

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Comunicazione tra processi applicativi

Comunicano tra di loro scambiandosi messaggi.
Chi provvede a trasferire i messaggi da una a l'altra è il processo applicativo.
Per ricevere i messaggi il processo deve avere degli identificatori (questo si fa attraverso l'indirizzo IP)

Addressing processes

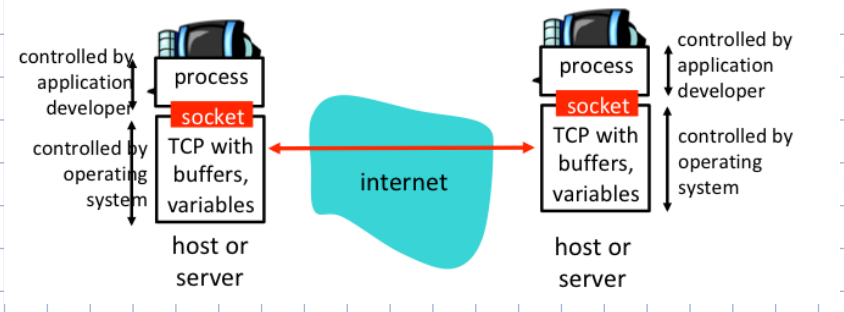
Il solo indirizzo IP non basta perché più processi can be running dallo stesso host.
Si aggiunge quindi il numero di porta.

Ciò che identifica il processo applicativo è : l'indirizzo IP per identificare l'host a cui far arrivare il pacchetto, e il numero di porta per vedere la destinazione dentro l'host.

- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80

Socket

Rappresenta l'interfaccia verticale tra livello applicativo e tutto quello che c'è sotto.
Sono delle porte attraverso cui passano, in entrambe le direzioni, i dati tra livello di trasporto e livello di applicazione.



Attraverso il socket l'applicazione può evocare il servizio TCP o UDP.

TCP: offre un servizio con connessione e servizio affidabile. ARQ, controllo di flusso e controllo della congestione. Il TCP non fornisce supporto per la qualità di servizio, sicurezza, e timing.

UDP: servizio senza connessione, non offre un servizio affidabile, non fa controllo di flusso, non fa controllo di congestione, timing o sicurezza.

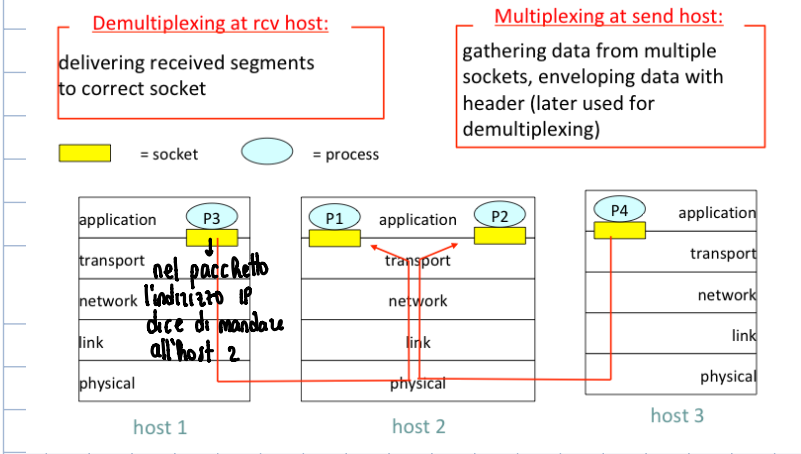
Allora perché è utile avere UDP?
Perché permette di avere maggiore manovra nel trasferimento, senza le regole di TCP.

Molte applicazioni hanno bisogno sempre delle stesse cose quindi è utile usare TCP. Altre invece che fanno a modo loro e cambiano con il tempo quindi usano UDP.

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

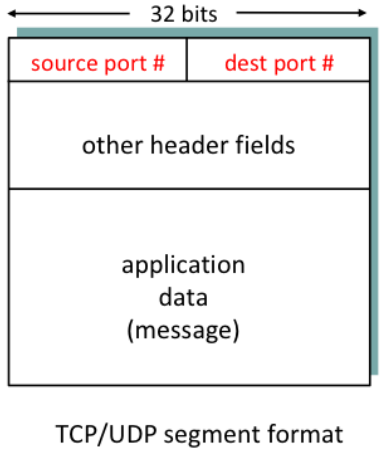
Riassumendo come si identifica un processo applicativo in internet?
Network layer: A livello di rete bisogna identificare l'host presso cui si trova, il livello di rete si occupare di gestire le comunicazioni tra host utilizzando gli indirizzi di livello di rete (indirizzi IP).
Transport layer:Questo però non basta perché serve il livello di trasporto che da un punto di vista applicativo si occupa di smistare i pacchetti che arrivano ad un host all'interno delle applicazioni di quell'host utilizzando i numeri di porta.

Multiplexing



Senza connessione il socket è formato da indirizzo IP e numero di porta.

Si prendono le informazioni dai diversi socket che dà informazione su quale host spedire e poi il numero di porta a quale applicazione.



TCP/UDP segment format

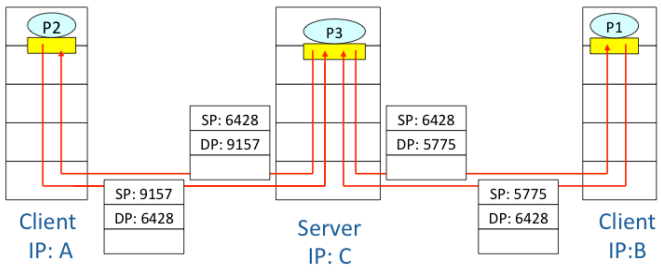
Sia TCP e UDP hanno un'intestazione e un carico pagante, l'informazione di controllo si trova tutta all'inizio.
L'intestazione è fatta dai primi 32 bit (16 bit per porta sorgente e 16 bit per porta di destinazione), il resto dell'intestazione è diversa per UDP e TCP.

Demutiplexing

Quando un host riceve un segmento UDP (senza connessione):
- controlla il numero di porta di destinazione nel segmento
- Dà il segmento UDP al socket con quel numero di porta.

senza connessione

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

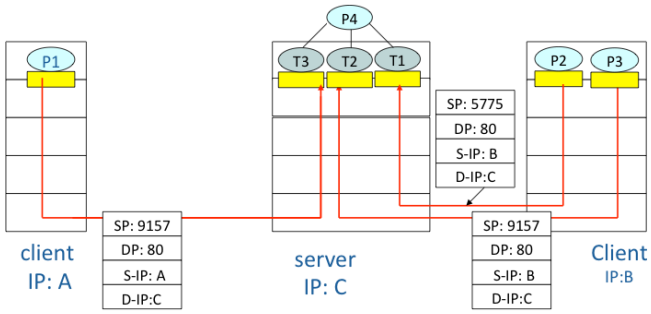


SP provides "return address"

Nel servizio con connessione il socket TCP identifica:
- Indirizzo IP di origine
- Il numero di porta di origine
- Indirizzo IP di destinazione
- Il numero di porta di destinazione

L'host ricevente usa tutti e 4 i valore per dirigere il segmento all'appropriato socket

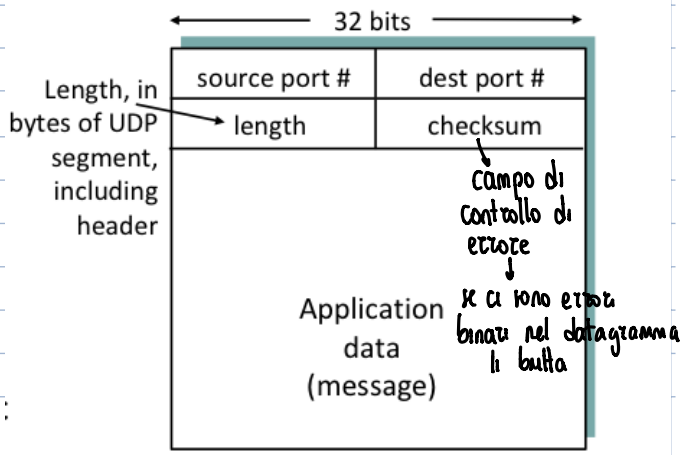
con connessione



Protocollo UDP

Il datagramma UDP è formato da un'intestazione fatta da 8 byte e il carico pagante.

UDP quando riceve un datagramma controlla attraverso il checksum che non ci siano errori binari, nel caso li butta.



UDP segment format

Protocollo TCP

(Integrare con il file word)