1 Lezione del 04-11-25

1.0.1 Tunneling

Continuiamo a parlare della coesistenza fra IPv4 e IPv6. Abbiamo già introdotto nella scorsa lezione l'opzione della *dual architecture*, cioè lo sviluppo di router che parlano sia la *"lingua"* dell'IPv4 che dell'IPv6.

Un'altro approccio valido è il **tunneling**: si incapsulano i datagrammi IPv6 all'interno di datagrammi IPv4, in modo che questi possano essere instradati da infrastruttura pensata per IPv4 (cioè ad esempio attraverso una rete Ethernet che supporta solo IPv4).

Chiaramente questo approccio porta a 2 visioni separate:

- La **visione logica**, che vede i router IPv4/v6 come connessi, appunto, da un *tunnel IPv4*;
- La **visione fisica**, che vede i router IPv4/v6 semplicemente connessi ad una rete di router IPv4, su cui instradano anziché datagrammi IPv6, datagrammi IPv4 che incapsulano nel loro campo payload datagrammi IPv6.

1.1 ARP

Il protocollo **ARP** (*Address Resolution Protocol*) è usato per associare indirizzi MAC (utili per l'indirizzamento su rete locale) ad indirizzi IP.

Ogni nodo IP (cioè host e router) ha una tabella ARP che contiene mappe IP/MAC per gli altri nodi "adiacenti" (e chiaramente un certo time-to-live di validità dell'associazione, in quanto ricordiamo gli IP sono variabili (e fino ad un certo punto lo sono anche i MAC)).

Il funzionamento di ARP è il seguente:

1. Poniamo che A voglia inviare un datagramma a B, e B non sia nella tabella ARP di A;
2. A invierà quindi un pacchetto di richiesta ARP in broadcast a tutti i nodi sulla rete locale;
3. B ricevera il pacchetto di richiesta ARP, e se questo è ben formato sarà l'unico a rispondere, inoltrando ad A il suo indirizzo MAC;
4. A potrà quindi inviare il datagramma a B. Inoltre, ci aspettiamo che A memorizzi il MAC ricevuto nella sua tabella ARP (associandolo appunto all'IP di B): in questo modo per successive trasmissione non dovrà ripetere la richiesta.

1.2 ICMP

Il protocollo **ICMP** (*Internet Control Message Protocol*) è stato già introdotto più volte. Viene usato dagli host e dai router per scambiarsi informazioni di livello network come notifiche, errori, ecc... Inoltre, supporta il meccanismo del `ping`, cioè i pacchetti di **echo**.

Il protocollo ICMP è sempre un protocollo di livello network, ma costruito su IP. Ha infatti 3 campi che vengono incapsulati nel datagramma IP:

- **Tipo** del messaggio;
- **Codice** del messaggio, che assieme al tipo permette di specificare completamente la natura del messaggio ICMP;

- **Header** e **primi 8 byte** del datagramma IP che ha causato la notifica.

Alcuni esempi di coppie tipo e codice ICMP sono i seguenti:

Tipo	Codice	Descrizione
0	0	Echo reply (ping)
3	0	Network destinazione irraggiungibile
3	1	Host destinazione irraggiungibile
3	2	Protocollo destinazione irraggiungibile
3	3	Porta destinazione irraggiungibile
3	6	Network destinazione sconosciuto
3	7	Host destinazione sconosciuto
8	0	Echo request (ping)
9	0	Route advertisement, usato dai router per pubblicizzare un percorso
10	0	Router discovery, usato dai router per presentarsi
11	0	TTL del datagramma scaduto
12	0	Header IP malformato

Riguardo a quanto detto su tipo e codice notiamo come, ad esempio, il tipo 3 è destinato ad errori di trasmissione (con il codice che discrimina *quale* fra diversi errori di trasmissione).

1.3 Forwarding generalizzato

Avevamo introdotto il modello **match plus action** per il forwarding: quando arriva un pacchetto, si fanno combaciare i bit con la tabella di forwarding (*match*) e quindi si fa una qualche *azione*. Nel **destionation based forwarding** (*forwarding basato su destinazione*), questa azione è di inoltrare i pacchetti verso l'indirizzo IP destinazione, basandosi solamente sull'indirizzo IP destinazione.

Nel **generalized forwarding** (*forwarding generalizzato*), invece, si cerca di usare tutti i campi dell'header IP per informare l'azione compiuta. Questa azione, infatti, potrà essere non solo l'*inoltrare* il pacchetto, ma anche lo *scartare*, *copiare*, *modificare* o *loggare* il pacchetto.

Chiaramente questo approccio prevede di fornirsi di un'astrazione diversa dalla tabella di forwarding, che chiamiamo **tabella di flusso**.

- Nella tabella di flusso associamo esplicitamente campi *match* a campi *action*.
- Il **flusso**, appunto, che questa tabella definisce sarà definito da diversi campi dell'header (di livello link, network e transport). L'esempio tipico è di prendere sia i campi sorgente che destinazione;
- Le **azioni**, come abbiamo già detto, sono più variegate di quelle previste dal modello basato su destinazione (possono prevedere di *scartare* pacchetti, *archiviarli*, ecc...);
- Occore definire una **priorità** per i pattern di match che si sovrappongono (come disambiguiamo su quale eseguire?);
- Infine dovemo prevedere **contatori** per quanti byte e quanti pacchetti coinvolgono le regole definite, a scopo di statistica.

1.3.1 OpenFlow

OpenFlow è un protocollo standard per il forwarding generalizzato.

Ogni entrata di una tabella di flusso OpenFlow ha 3 campi:

1. **Match**: può offrire un'operazione di match per diversi campi header MAC, IP e TCP/UDP;
2. **Action**: definisce diverse operazioni, fra cui:
   - Forwarding verso porte;
   - Drop di pacchetti (scarta pacchetti);
   - Modifica campi negli header di pacchetto;
   - Incapsula pacchetto e inoltra a *controller*;
   - Ecc...
3. **Stats**: permette di contare quanti pacchetti di un certo tipo sono stati analizzati, o quanti byte sono passati sul router, ecc...

L'astrazione match plus action permette di unificare tutta una serie di dispositivi diversi:

- **Router**: implementando regole di instradamento basate sui campi header IP;
- **Switch**: implementando regole di instradamento basate sui campi header MAC, magari appoggiandosi anche su azioni di *flooding* (trasmissioni in broadcast);
- **Firewall**: facendo match su indirizzi IP e numeri di porta TCP/UDP, e sfruttando azioni di drop o instradamento pacchetto;
- **NAT**: facendo match su indirizzi IP e numeri di porta TCP/UDP e sfruttando azioni di modifica dei pacchetti (riscrittura indirizzi UP e porte TCP/UDP).

Riassumendo, abbiamo quindi che l'astrazione *match plus action* è basata su operazioni di match su più di un campo header, e azioni più variegate rispetto al semplice inoltro. Permette di definire il cosiddetto *forwarding generalizzato*, e *OpenFlow* ne è un'implementazione standard. Abbiamo che attraverso questa operazione possiamo reliazzare reti effettivamente **programmabili**, cioè il cui comportamento può essere configurato, a livello di tabelle di flusso, da dispositivi di ordine superiore (che magari automatizzano la configurazione sulla base di informazioni di livello più alto).

1.4 Middlebox

Vengono detti **middlebox** tutti i dispositivi che non sono né host né router. Ad esempio, il *NAT* è un middlebox, come lo sono il *firewoll*, l'*IDS* (*Intrusion Detection System*), i *load balancer*, le *cache* e tutta un'altra serie di dispositivi ad uso speficico ad applicazioni (ad esempio si pensi all'infrastruttura richiesta dai *CDN*).

Le middlebox nascono come soluzioni proprietarie, e quindi a sorgente e specifiche *chiuse*. In seguito, ci siamo spostati verso le cosiddette *"whitebox"*, cioè hardware implementato attraverso API open source, programmabili attraverso il match plus action.

Questo permette di avvicinarsi al cosiddetto **SDN** (*Software Defined Networking*), cioè la centralizzazione del controllo e della configurazione dei dispositivi che compongono la rete, spesso da remoto (*cloud*, ecc...).

1.5 Comunicazione fra processi

Abbiamo visto finora la comunicazione fra **host**.

Studiando il livello application, però, abbiamo visto che veramente siamo interessati all'**IPC** (*Inter Process Communication*), cioè la comunicazione fra più processi all'interno della stessa o più macchine.

Possiamo basarci sui livelli fisici, datalink e network per realizzare il trasporto fra macchine, e ci basiamo su un ultimo livello, il livello **transport**, per realizzare l'IPC. Ricordiamo che i protocolli di questo livello sono **TCP** (*Transmission Control Protocol*) e **UDP** (*Universal Datagram Protocol*). Oltre alla comunicazione diretta (almeno dal loro punto di vista) fra processi, vedremo come i protocolli di livello transport possono implementare servizi come:

- **Multiplexing** e **demultiplexing** di più messaggi sulla stessa linea;
- **Reliable data transfer**, già visto in 10.3 al livello datalink;
- Controllo **flusso** e **congestione**.

Il compito dei protocolli di livello transport è quindi quello di permettere la comunicazione *logica* fra più processi applicazione in esecuzione su macchine diverse. Ricordiamo, *logica*, in quanto la comunicazione effettiva avviene attraverso l'intero stack protocollare (fisico, datalink, network e quindi transport).

Al livello transport la comunicazione avviene in **segmenti** passati al livello network. Questo perché il livello transport non vede pacchetti, ma vede *byte*: si vuole infatti creare l'astrazione di un *flusso* continuo di byte, e un segmento non è altro che una parte di questo flusso (che viene quindi diretta in rete sotto forma di pacchetti).

Il livello transport dell'host mittente divide quindi i messaggi di livello application in più segmenti, che vengono ricomposti dal livello transport dell'host destinatario per essere quindi consegnati al relativo livello application, realizzando effettivamente questa astrazione di *byte stream* fra livelli application (cioè fra processi).