DATALINK LAYER

E' il livello 2 dello stack TCP/IP ed ha il seguente COMPITO:

Trasferire frames da un nodo ad un altro nodo adiacente su un link!

- Ci sono link diversi, ognuno con protocolli e servizi differenti (e.g. Ethernet, Wifi, ...)

- SERVIZI:

1) FRAMING e ACCESSO AL LINK:
   - Incapsulare i datagrammi in frames, aggiungendo header e trailer
   - Gestire l'accesso al canale, se il mezzo è condiviso
   - Usare indirizzi "MAC" nell'header dei frames per i nodi source e dest.

2) COMUNICAZIONE AFFIDABILE:
   - Già visto come fare per il livello 4; i principi sono gli stessi!
   - * Ma, PERCHÉ gestire l'AFFIDABILITÀ sia a LV. 2 che a LV. 4?
     1. Perché i livelli sono INDIPENDENTI
     2. A livello 3 un router può scartare un datagramma per congestione!

3) CONTROLLO DI FLUSSO: Rate trasmissivo controllato dal nodo ricevente e trasmettente

4) ERROR DETECTION: Receiver in grado di RILEVARE errori, dovuti ad attenuazione del segnale o rumore

5) ERROR CORRECTION: Grazie a particolari codici e correzione d'errore, il receiver è in grado di CORREGGERE eventuali bit flipped nei frames ricevuti
   ↳ Questo servizio non è presente a livello 4!

6) HALF-DUPLEX e FULL-DUPLEX: Con Half-Duplex si può trasmettere oppure ricevere, ma non entrambe contemporaneamente.

- DOV'È IMPLEMENTATO IL LINK LAYER?

Si trova in tutti gli host e i router! In particolare, è implementato tramite la NIC = Network Interface Card (Schede di Rete).

Le NIC sono collegate al bus (o ai bus) di sistema e sono una combinazione di hardware, software e firmware.

- PARITY CHECKING:
  - 1 Parity Bit → Permette di RILEVARE 1 solo errore
  - Two Dimensional Parity Bit → 1 Parity Bit per ogni riga e colonna → Permette di rilevare e CORREGGERE 1 solo errore

• MULTIPLE ACCESS LINKS:

Ci sono sostanzialmente 2 tipi di link (a noi interessa il secondo):

1) POINT-TO-POINT:
   - PPP per accessi dial-up, ad esempio rete telefonica
   - point-to-point link tra uno switch Ethernet e un host

2) BROADCAST:
   - cavo o mezzo condiviso
   - ad esempio, il vecchio Ethernet con cavo condiviso
   - upstream HFC (Hybrid Fiber Coax)
   - 802.11 wireless LAN, cioè Wi-fi

* Su un mezzo condiviso e broadcast si possono generare COLLISIONI!

  ↳ C'è bisogno di PROTOCOLLI DI ACCESSO MULTIPLO per decidere quando un nodo può trasmettere.
  Notare che informazioni sulla condivisione e l'accesso al canale devono essere trasmesse usando il canale stesso!

• MULTIPLE ACCESS PROTOCOLS:

Algoritmo distribuito per la condivisione del canale, cioè che determini quando un nodo possa trasmettere.
Sono protocolli MAC = Medium Access Control. Ce ne sono 3 categorie:

- CHANNEL PARTITIONING: segue il principio della rete a circuito, suddividendo la banda in slot assegnati ai nodi ad uso ESCLUSIVO!
- RANDOM ACCESS: ogni nodo trasmette quando vuole, ci sono collisioni, ma è in grado di recepire e recuperare alle collisioni (Ethernet, ...)
- TAKING TURNS: i nodi prendono dei turni per trasmettere, nodi con più dati da trasmettere prendono dei turni più lunghi (Reti Token Ring).

• CHANNEL PARTITIONING

1. TDMA = Time Division Multiple Access
2. FDMA = Frequency Division Multiple Access
3. CDMA = Code Division Multiple Access

} Stessi principi del TDM e FDM in reti a circuito

* Slot inutilizzati finiscono per essere IDLE

- **RANDOM ACCESS PROTOCOLS**
  - Quando un nodo deve trasmettere, lo trasmette al massimo data rate del canale R.
  - Possono esserci COLLISIONI!
  - I diversi protocolli si differenziano su:
    - Come rilevare le collisioni
    - Come recuperare dalle collisioni (e.g. via delayed retransmission)
  - Esempi di protocolli: slotted ALOHA, ALOHA, CSMA, CSMA/CD, CSMA/CA.

SLOTTED ALOHA

Assunzioni:

1. Tutti i frame hanno stessa grandezza
2. Tempo diviso in slot uguali → SLOTTED
3. Un nodo inizia a trasmettere solo all'inizio di uno slot
4. I nodi sono SINCRONIZZATI
5. Se c'è una collisione, tutti i nodi la rilevano

Operazioni:

- Quando un nodo ottiene un frame da trasmettere, lo trasmette nello slot successivo;
  - Se non c'è collisione: il nodo può inviare un altro frame nello slot successivo
  - Se c'è collisione: il nodo ritrasmette il frame in ogni slot successivo con PROBABILITÀ p, fino al successo.

- **PRO e CONTRO:**

PRO	CONTRO
• Un singolo nodo attivo può trasmettere continuamente all'intero rate del canale	• Collisioni, si perdono slots
• Altamente decentralizzato: c'è bisogno solo di sincronizzare gli slots dei nodi	• Slots vuoti
• Semplice	• I nodi dovrebbero essere in grado di rilevare le collisioni in un tempo minore del tempo di trasmissione
	• Richiede sincronizzazione dei clock

- **SLOTTED ALOHA EFFICIENCY:**

Valutiamone l'efficienza come la percentuale di tempo (slots) che il canale è occupato per trasmissioni che vanno a buon fine rispetto al totale.

Supponiamo che ci siano N nodi con sempre frames da trasmettere, e che ognuno trasmette in ogni slot con probabilità p.

$$P(\text{successo}) = p(1-p)^{N-1} \longrightarrow \text{successo in 1 slot}$$

- La probabilità che un qualsiesi nodo abbia successo è: $Np(1-p)^{N-1}$
- La probabilità p^* che massimizza l'efficienza è $\boxed{p^* = \frac{1}{N}}$, per cui si ha: $\left(1-\frac{1}{N}\right)^{N-1}$
- Facciamo il limite per $N \to +\infty$: $\quad k = -N$

$$\lim_{N\to+\infty}\left(1-\frac{1}{N}\right)^{N-1} = \lim_{N\to+\infty}\left(1+\frac{1}{(-N)}\right)^{N} \overset{\downarrow}{=} \lim_{k\to-\infty}\left[\left(1+\frac{1}{k}\right)^{k}\right]^{-1} = \frac{1}{e} \simeq 0{,}37$$

$\Rightarrow$ **L'efficienza (throughput) è del 37%!** $\longrightarrow$ È bassissima!

- **PURE (UNSLOTTED) ALOHA:**

È SENZA SINCRONIZZAZIONE!

↳ Ci può essere collisione anche con lo "slot" precedente!

Valutiamone l'efficienza:

$$P(\text{successo}) = p(1-p)^{N-1}\cdot(1-p)^{N-1} \Rightarrow \text{Qualsiesi successo} = Np(1-p)^{N-1}\cdot(1-p)^{N-1}$$

- Per massimizzare l'efficienza, scegliamo $\boxed{p^* = \frac{1}{2N-1}}$ ottenendo:

$$\frac{N}{2N-1}\left(1-\frac{1}{2N-1}\right)^{2(N-1)}$$

- Facciamone il limite per $N \to +\infty$:

$$\lim_{N\to+\infty}\left(1-\frac{1}{N}\right)^{2N-1} = \lim_{N\to+\infty}\left(1+\frac{1}{(-N)}\right)^{2N} = \lim_{k\to-\infty}\left[\left(1+\frac{1}{k}\right)^{k}\right]^{-2} = \frac{1}{e^2} = 0{,}137$$

$$x = 2N-1 \Rightarrow N = \frac{x+1}{2} \quad ; \quad 2(N-1) = x-1$$

$$\lim_{x\to+\infty}\frac{x+1}{2x}\left(1-\frac{1}{x}\right)^{2x-1} = \lim_{x\to+\infty}\underbrace{\frac{x+1}{2x}}_{\to\frac{1}{2}}\cdot\underbrace{\frac{x}{x-1}}_{\to 1}\cdot\underbrace{\left(1-\frac{1}{x}\right)^{x}}_{\to\frac{1}{e}} \simeq \frac{1}{2e} = 18.4\%$$

$\Rightarrow$ **L'efficienza è del 18,4%**

- Non richiede sincronizzazione, ma è ancora peggio di Slotted ALOHA!

CSMA

- CSMA = Carrier Sense Multiple Access

1) CARRIER SENSE: prima di trasmettere si fa SENSING per verificare che il canale sia IDLE.
   Se il canale è occupato, si fa DEFER della trasmissione.

* Possono ancora esserci collisioni!
Infatti, il ritardo di propagazione (d_{prop}) può far sì che 2 nodi si accorgano in ritardo che stanno trasmettendo entrambi.

2) COLLISION DETECTION (CD): quando ci si accorge di collidere, si fa ABORT della trasmissione e poi il DEFER così da ridurre il tempo di inutilizzabilità del canale.

- In reti Wireless (come Wifi), la forza del segnale ricevuto verrebbe sovrastata da quella del segnale trasmesso ed è difficile fare il sensing!
  Si fa COLLISION AVOIDANCE → CSMA/CA.

- "TAKING TURNS" MAC PROTOCOL:

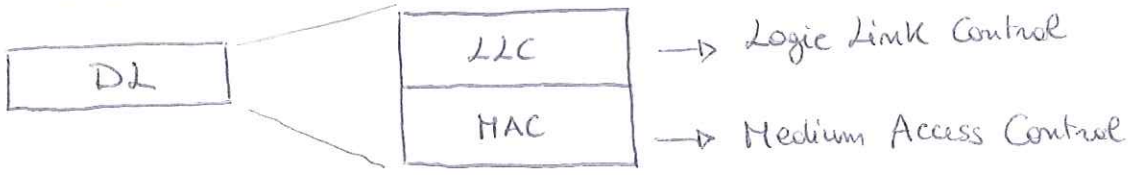
1) POLLING: c'è un nodo master che a turni invita i nodi slaves a trasmettere, una specie di Round-Robin.
   - polling overhead
   - latenze
   - singolo punto di rottura (master)

2) TOKEN PASSING: c'è un TOKEN di controllo che i nodi, disposti ad anello, si passano sequenzialmente
   - token overhead
   - latenze
   - singolo punto di rottura (token)

- Protocolli molto usati nel Bluetooth e nelle ormai desuete reti Token Ring della IBM.

MAC ADDRESS

Il livello Datalink è suddiviso in 2 parti:



- LLC fa controllo d'errore, di flusso, ...
- MAC si occupa dell'accesso al mezzo fisico e dei protocolli che lo gestiscono

- Gli indirizzi IP sono a 32-bit, non dipendono dalle sottoreti → INDIRIZZI LOGICI !

- **MAC ADDRESS**:
  - anche detto indirizzo FISICO, LAN o Ethernet
  - è a 48-bit (6 Byte)
  - sono presenti nelle memorie ROM delle NIC, ma sono anche settabili via software
  - ce n'è 1 per ogni interfaccia / scheda di rete!

RUOLO: identificare un nodo in una LAN

BROADCAST: indirizzo ff:ff:ff:ff:ff:ff → Invia a tutti i nodi della LAN

- Sono indirizzi di tipo FLAT → (PORTABILI) da una LAN all'altra
  Gli indirizzi IP non lo sono !!!
- I MAC addresses sono gestiti dall'IEEE e vengono acquistati dai costruttori di hardware/software per poterli assegnare; sono UNIVOCI !
  Tipicamente, i primi 3 Byte identificano il costruttore e gli altri 3 Byte la scheda.

ARP: ADDRESS RESOLUTION PROTOCOL

Come ottenere l'indirizzo MAC a partire dall'indirizzo IP?

- Ogni nodo IP, host o router che sia, ha una ARP TABLE, in cui mantiene i mappings tra gli indirizzi IP e i MAC address che ha imparato, associati con un time-to-live (soft state). Il ttl è tipicamente di 20 minuti.

* NOTA: Ha senso voler conoscere un MAC address solo di nodi che sono nelle proprie LAN! Altrimenti, bisognerà passare per forza per IP, quindi individuare il MAC del router (della LAN propria).

- **ARP: stessa LAN**

Il nodo A vuole contattare B, presente nella sua stessa LAN, ma non ha B nella ARP table.

1) A invia un pacchetto di query ARP contenente l'IP di B in BROADCAST nella LAN:
   - dest MAC address: ff:ff:ff:ff:ff:ff → Ricevuto da tutti i nodi.
2) B lo riceve e risponde ad A con il proprio MAC address: il frame viene inviato direttamente all'indirizzo MAC di A (unicast)
3) A salva il mapping <IP di B, MAC di B, ttl> nella ARP table, associandogli un ttl.

* ARP è "PLUG & PLAY": nessuna configurazione necessaria → SELF-LEARNING dei nodi.

- **ARP : LAN diverse :**

A vuole contattare B, di cui conosce l'indirizzo IP, ma B è in un'altra LAN.
A conosce anche l'IP del router R.

1. A si chiede se B appartenga o meno alla sua stessa sottorete IP. Vedendo che non vi appartiene, tramite protocollo IP, scopre di dover contattare il router R.
2. Decidere il next-hop (R) e vedere che appartiene alla propria sottorete
3. Ricavare l'indirizzo MAC di R:
   - 3.1 Se il mapping è presente nella arp table, usarlo
   - 3.2 Se non è presente, inviare ARP request in broadcast nella LAN per ottenerlo
4. Inviare sul link verso R un frame con IP source = IP di A, IP dest = IP di B, MAC source = MAC di A, MAC dest = MAC di R.

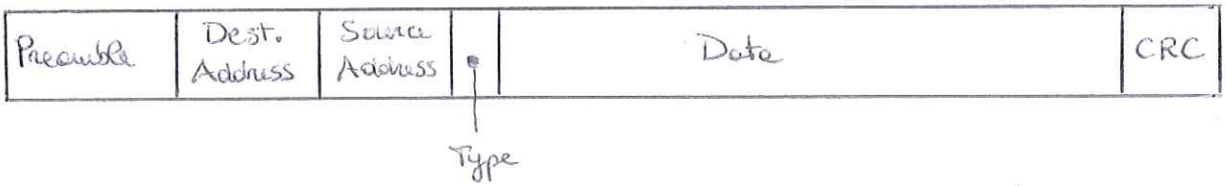
* NOTA: il 1° passo non è il broadcast!!! Bensì è chiedersi se B è nella propria sottorete IP!!!

- **ETHERNET :**

Tecnologia semplice, economica ed efficiente.

- TOPOLOGIA A BUS: Cavo coassiale condiviso. Legacy Ethernet (Metcalf, '90s). Tutti i nodi sono nello stesso dominio di collisione.
- TOPOLOGIA A STELLA: C'è uno (SWITCH) al centro e ogni nodo parla con lo switch. Cavi doppi per trasmissione e ricezione tra ogni host e lo switch → COLLISIONI EVITATE!

- **FRAME:**



- Preambolo: 8 Byte, di cui i primi 7 sono "10101010" e l'8° è "10101011". Si usa per far sì che il receiver si sincronizzi con il clock del sender (Codifica di Manchester)
- Address: 6 Byte ciascuno, quanto appunto il MAC Address, al contrario di IP, qui va prima la destinazione! Se la NIC riceve un frame in cui il Dest Address combacia con il proprio o con quello broadcast, passa il datagram ma a livello 3, altrimenti lo scarta
- Type: Indica il protocollo di livello superiore, tipicamente IP!

- CRC: Cyclic Redundancy Check, usato dal receiver per il controllo d'errore. Se qualcosa va male, il frame viene scartato → UNRELIABLE!

- **CARATTERISTICHE DI ETHERNET:**
  - **CONNECTIONLESS**: le NICs non fanno handshaking e non stabiliscono una connessione prima di trasmettere
  - **UNRELIABLE**: le NICs non inviano ACK o NAK:
    - lo stream di datagrammi passati a livello 3 può avere dei buchi
    - i buchi verranno riempiti se si usa TCP, altrimenti rimarranno
  - Utilizza come protocollo MAC il **CSMA/CD** con **EXPONENTIAL BACKOFF**.

- **Eth CSMA/CD ALGORITHM:**

1) La NIC riceve un datagramma dal livello 3 e lo incapsula in un frame
2) Se la NIC rileva che il canale è IDLE, inizia la trasmissione. Se sente il canale BUSY, aspetta finché non diventa IDLE e ritrasmette
3) Se la NIC trasmette l'intero frame senza rilevare un'altra trasmissione, allora è ok!
4) Se la NIC avverte un'altra trasmissione mentre trasmette, allora fa l'ABORT della trasmissione e invia un SEGNALE DI JAM
5) Dopo l'abort, la NIC entra nella fase di EXPONENTIAL BACKOFF:

   dopo la m-esima collisione, la NIC sceglie random un numero $K \in \{0, \dots, 2^m - 1\}$.
   La NIC attende $K \cdot 512$ bit times, prima di ritornare a trasmettere.

   Se $m > 10$, K viene comunque scelto come se m fosse 10, cioè $K \in \{0, \dots, 1023\}$.

- **SEGNALE DI JAM**: 48 bits → Serve per essere sicuri che tutti gli altri nodi che stanno trasmettendo avvertano la collisione

- **BIT TIME**: 0,1 μs per Ethernet a 10 Mbps; per $K = 1023$, wait time ≃ 50 ms
  È il tempo per trasmettere 1 bit (512 bit ≃ 1 slot).

- **EFFICIENCY:**

Se t_{prop} = max ritardo di propagazione tra 2 nodi nella LAN
t_{trans} = tempo di trasmissione del più grande frame possibile

$$efficiency = \frac{1}{1 + 5\frac{t_{prop}}{t_{trans}}}$$

- Tende → ad 1 per:
  - $t_{prop} \to 0$: molto più probabile
  - $t_{trans} \to \infty$: improbabile

! Performance migliori di ALOHA! Inoltre, è semplice, economico e decentralizzato!

• INTERCONNESSIONE DI SEGMENTI LAN:

- **HUB**: è sostanzialmente un ripetitore di segnale. Permette di estendere le distanze massime tra i nodi della LAN, ma aumenta le dimensioni del collision domain!
  Rispetto al coassiale fisso, permette una manutenzione fisica più agevole in caso di guasti.

- **BRIDGE**: è un antenato dello switch; aveva molte meno porte ed era più lento, ma il funzionamento è lo stesso.

- **SWITCH**:
  1) Separano i DOMINI DI COLLISIONE, dividendo la rete LAN in segmenti; cioè evita che tutti i nodi nella LAN collidano, filtrando i pacchetti da trasmettere su altri segmenti della LAN, facendo a sua volta CSMA/CD
     ↳ Ethernet con Switch → EVITA COLLISIONI!
  2) Fa anche da repeater, ricostruendo il segnale
  3) Fa lo storage dei frames da inviare in BUFFERS (gli hub no!)
  4) Evitano che frames vengano trasmessi in segmenti della LAN dove non serve, tramite MAC table e un processo di self-learning.
     Gestisce il ROUTING all'interno della LAN, salvando in una switch table il mapping tra host raggiungibili (MAC address) e l'interfaccia tramite il quale raggiungerli, con un TTL associato!
     * Questi mapping vengono impostati tramite un processo di learning: si fa il learning solo del Source MAC Address dei frames che arrivano!

• FILTERING / FORWARDING:

Quando un bridge riceve un frame:

Cerca nella bridge table usando il MAC dest address:

```
if entry found for destination
  then{
    if dest on segment from which frame arrived
      // butta il frame, non c'è bisogno di inoltrarlo
      then drop the frame
      else forward the frame on interface indicated
  }
  else flood // broadcast forwarding, except on receiving interface
```

- Se un frame è destinato alla stessa interfaccia da cui proviene, viene scartato, in quanto non c'è bisogno di inoltrarlo!

- Quando non si ha una entry nella bridge table per la destinazione, così come quando si ha un frame broadcast, si invia il pacchetto in BROADCAST su tutti i Segmenti LAN, eccetto quello da cui proviene.
  In questo modo, il frame raggiungerà di certo la destinazione; tuttavia, si farà il learning solo se la destinazione dovesse rispondere!

* ATTENZIONE: (Diff. con IP) L'indirizzamento non è logico e/o gerarchico come in IP !!! Qui si ha 1 entry nella bridge table per OGNI MACCHINA!

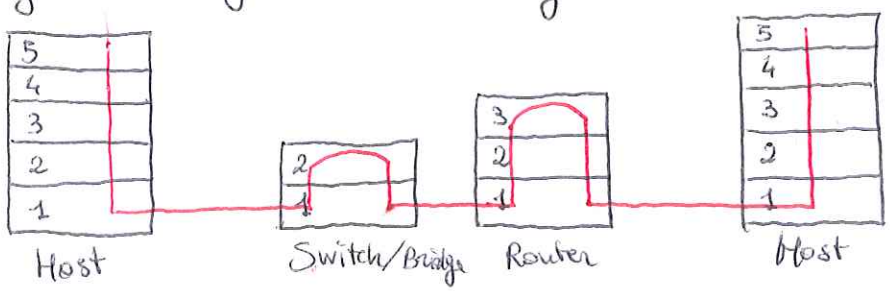
- BRIDGES SPANNING TREE:

Per aumentare l'affidabilità, è desiderabile avere più percorsi alternativi nella rete. Tuttavia, percorsi multipli si traducono in CICLI nella rete! In una LAN di livello 2, specialmente con frames inviati in broadcast (ARP request ad esempio), si avrebbe che i pacchetti vengono rimpallati all'infinito tra gli switch e gli hub
↳ BROADCAST STORM e Pacchetto di Chernobyl!

SOLUZIONE: gli switch/bridge usano un algoritmo di SPANNING TREE, che permette loro di comprendere la topologia della rete e scegliere quali interfacce disabilitare per renderla un albero, spanning tree appunto, e quindi aciclica!

- NOTA: Isolare i domini di collisione contribuisce ad aumentare il throughput totale della rete, poiché si possono avere più comunicazioni in parallelo!
- Gli switch/bridge sono "PLUG & PLAY" → Si accendono e funzionano, self learning, senza configurazione

- SWITCHES VS ROUTERS:
  - Entrambi sono dispositivi STORE-AND-FORWARD:
    - routers di livello di rete (3)
    - switches di livello di collegamento (2)
  - I routers mantengono una routing table, implementano algoritmi di routing
  - Gli switches mantengono una switch table, implementano il filtering dei frames, eseguono l'algoritmo di learning.



- ROUTERS VS BRIDGES:

- BRIDGES:

PRO	CONTRO
◦ Le operazioni del bridge sono più semplici e richiedono meno processamento del pacchetto ◦ Le bridge tables sono self learning	◦ Tutto il traffico è limitato allo spanning tree, nonostante ci sia altra banda disponibile ◦ I bridges non offrono protezione da BROADCAST STORM, si possono effettuare attacchi tramite ICMP o UDP

- ROUTERS:

PRO	CONTRO
◦ Supporta topologie arbitrarie, i cicli sono gestite col TTL e da buoni protocolli di routing ◦ Fornisce protezione da broadcast storm	◦ Richiede configurazione dell'indirizzo IP (No plug-&-play) ◦ Richiede un maggiore processamento dei pacchetti

- Bridges buoni per reti piccole; Routers buoni per reti grandi.

Routers vs. Bridges

Routers + and -

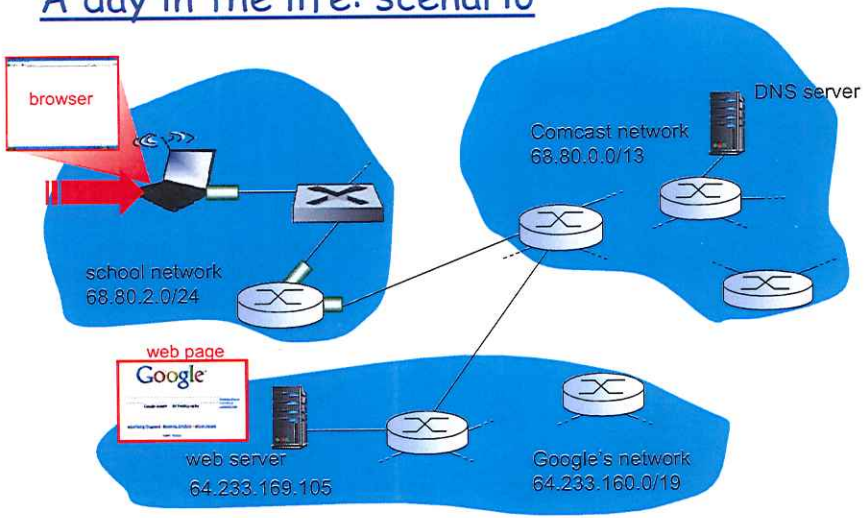
+ arbitrary topologies can be supported, cycling is limited by TTL counters (and good routing protocols)
+ provide protection against broadcast storms
- require IP address configuration (not plug and play)
- require higher packet processing

- bridges do well in small (few hundred hosts) while routers used in large networks (thousands of hosts)

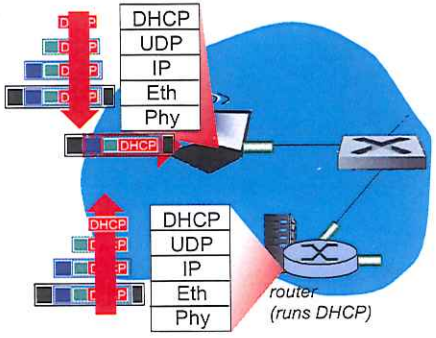
Synthesis: a day in the life of a web request

- journey down protocol stack complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

A day in the life: scenario

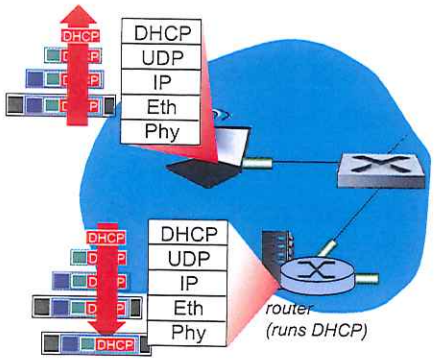


A day in the life... connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request *encapsulated* in *UDP*, encapsulated in *IP*, encapsulated in *802.3* Ethernet
- Ethernet frame *broadcast* (dest: FFFFFFFFFFFF) on LAN, received at router running *DHCP* server
- Ethernet *demuxed* to IP demuxed, UDP demuxed to DHCP

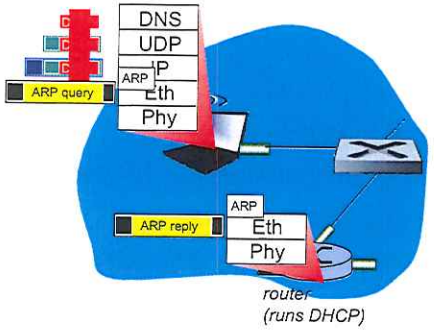
A day in the life... connecting to the Internet



- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (*switch learning*) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

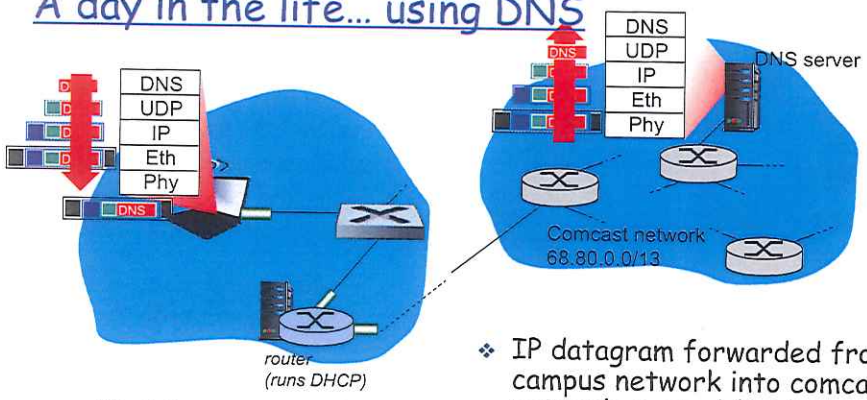
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life... ARP (before DNS, before HTTP)



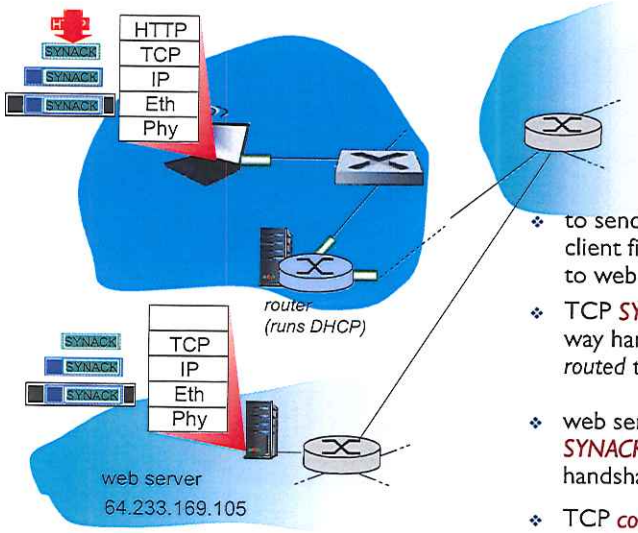
- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
- *ARP query* broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

A day in the life... using DNS



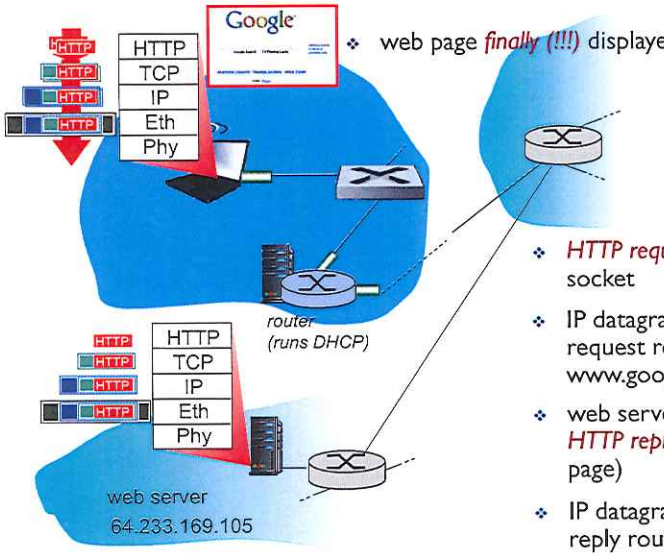
- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into comcast network, routed (tables created by *RIP, OSPF, IS-IS* and/or *BGP* routing protocols) to DNS server
- demux'ed to DNS server
- DNS server replies to client with IP address of www.google.com

A day in the life...TCP connection carrying HTTP



- to send HTTP request, client first opens *TCP socket* to web server
- TCP *SYN segment* (step 1 in 3-way handshake) *inter-domain routed* to web server
- web server responds with *TCP SYNACK* (step 2 in 3-way handshake)
- TCP *connection established!*

A day in the life... HTTP request/reply



- web page *finally (!!!)* displayed

- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with *HTTP reply* (containing web page)
- IP datagram containing HTTP reply routed back to client