1.

Servicio	Protocolo	Puerto por defecto
Web	HTTP	80
SSH	SSH	22
DNS	UDP/TCP	53
Web Seguro	HTTPS	443
POP3	POP3	110
IMAP	IMAP	143
SMTP	SMTP	25

✅ En Linux
Los servicios y sus puertos por defecto están descritos en el archivo:
- /etc/services
✅ En Windows
Windows guarda esta información en el archivo:
- C:\Windows\System32\drivers\etc\services

2. **Multicast** es una técnica de transmisión de datos en redes que permite enviar **un solo paquete** desde un origen a **múltiples destinos seleccionados**, en lugar de enviar una copia por cada receptor como en el unicast. Multicast funciona sobre el protocolo UDP (User Datagram Protocol). UDP **no requiere conexión** previa y **no garantiza entrega**, lo que es ideal para aplicaciones que prefieren rendimiento sobre fiabilidad.
   No se puede adaptar fácilmente, y prácticamente **no es viable usar TCP con multicast**. Las razones son:
   1. **TCP es orientado a conexión**: establece una conexión 1 a 1 (unicast), no 1 a N.
   2. **Control de flujo y congestión por conexión individual**: tendría que mantener y sincronizar múltiples conexiones activas, lo que es **ineficiente y complejo**.
   3. **Confirmación y reenvío**: cada destino debería confirmar recepción (ACK), lo que generaría **"tormenta de ACKs"** (implosión de acuses de recibo).
3. 🔁 Modo **Activo**
   - Cliente se conecta al **puerto 21** del servidor (canal de control).
   - Cliente **abre un puerto aleatorio** y se lo informa al servidor con el comando `PORT`.

- El servidor se conecta **desde su puerto 20** al puerto informado del cliente para establecer el **canal de datos**.
  Desventaja: Muchos firewalls en el cliente **bloquean conexiones entrantes**, por lo que este modo suele fallar en redes modernas.
  🔒 Modo **Pasivo**
- Cliente se conecta al **puerto 21** del servidor (canal de control).
- Cliente envía el comando `PASV`.
- El servidor **abre un puerto aleatorio** y le informa al cliente.
- El cliente se conecta a ese puerto del servidor para el canal de datos.
  Ventaja: El cliente siempre inicia las conexiones, por lo que **funciona mejor con NAT y firewalls**.
  🆚 Diferencias con otros protocolos de aplicación (HTTP, SSH, etc.)
- **HTTP, SSH, SMTP, DNS**: usan **una sola conexión TCP** (por lo general).
- **FTP**: usa **dos conexiones separadas**, lo que lo hace más complejo de manejar (especialmente con NAT/firewalls).
- Además, FTP no encripta datos ni credenciales por defecto → se recomienda **FTPS o SFTP** para mayor seguridad.

4. **Host A** envía los segmentos 0, 1, 2 (con error), 3, 4, 5.
   Host B recibe:
   - 0 → ✅ responde con ACK 0
   - 1 → ✅ responde con ACK 1
     Host B envía ACK 2 tres veces seguidas después del error en la trama 2, como señal de que está esperando dicha trama.
5. La ventana de envío debe ser como máximo la mitad del espacio total de numeración de secuencia. Si el número de secuencia usa **k bits**, entonces hay un total de **2^k valores posibles** (números de secuencia de 0 a 2^k-1). **Selective Repeat permite recibir y almacenar paquetes fuera de orden**, y los números de secuencia se **reutilizan cíclicamente** (mod 2k2^k2k). Si el tamaño de la ventana fuera mayor que 2k−12^{k-1}2k−1, el receptor **no podría distinguir entre un paquete nuevo y uno retransmitido**, ya que el número de secuencia se habría **reutilizado dentro de una ventana activa**.
6. Los valores se ven en los campos del detalle del paquete:
   - **Source:** `172.20.1.100`
   - **Destination:** `172.20.1.1`
   - **Protocol:** TCP
   - **Source port:** `11111`
   - **Destination port:** `41749`
   - **Sequence number:** `1047471501`
   - **Acknowledgment number:** `3933822138`

- **Flags:** `SYN, ACK`
- **Window size:** `5792`
  Valores de la línea borrada:
- **Source:** `172.20.1.1`
- **Destination:** `172.20.1.100`
- **Info:** `41749 > vce [ACK] Seq=3933822138 Ack=1047471502

7. Información a tener en cuenta:
   - El **SYN** consume **1 unidad de secuencia**.
   - Cada byte de datos enviados incrementa el número de secuencia.
   - Los ACK **no consumen secuencia**, pero su número de secuencia se mantiene en el último utilizado.
   - El **FIN** también **consume 1 unidad de secuencia**.

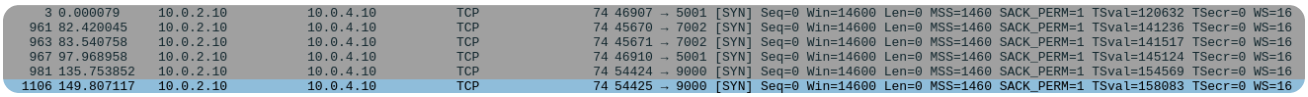
Time	10.0.0.10 10.0.1.10	Comment
1360	(54762) ----------SYN----------> (10000)	Seq = 0
1360	(54762) --------SYN,ACK-------> (10000)	Seq = 0 Ack = 1
1360	(54762) ---------ACK----------> (10000)	Seq = 1 Ack = 1
3581	(54762) ----PSH,ACK-Len:7----> (10000)	Seq = 1 Ack = 1
3581	(54762) ----------ACK----------> (10000)	Seq = 1 Ack = 8
8796	(54762) ----PSH,ACK-Len:9----> (10000)	Seq = 8 Ack = 1
8797	(54762) ----------ACK----------> (10000)	Seq = 1 Ack = 17
14382	(54762) ----PSH,ACK-Len:5----> (10000)	Seq = 17 Ack = 1
14382	(54762) ----------ACK----------> (10000)	Seq = 1 Ack = 22
15190	(54762) --------FIN,ACK--------> (10000)	Seq = 22 Ack = 1
15190	(54762) --------FIN,ACK--------> (10000)	Seq = 1 Ack = 23
15190	(54762) ----------ACK----------> (10000)	Seq = 23 Ack = 2

8. RTT (Round-Trip Time) es el tiempo que tarda un paquete en ir desde el emisor al receptor y volver con una respuesta.
   🔁 En TCP, este valor es crucial para:
   Estimar el tiempo de espera de retransmisiones (RTO),
   Calcular el desempeño de la red,
   Adaptar el tamaño de la ventana (control de congestión).
   RTT = Tiempo de respuesta - Tiempo de envío
   La **opción TCP Timestamp** fue introducida para mejorar el cálculo del RTT y apoyar el control de congestión. Esta opción agrega dos campos en el encabezado TCP: **TSval**

(Timestamp Value) --> El tiempo actual del emisor del segmento y **TSecr** (Timestamp Echo Reply) --> Copia del último TSval recibido del otro host

- El host A envía un segmento con `TSval = t1`.
- El host B responde con un ACK y **TSecr = t1**, y su propio `TSval = t2`.
- Cuando A recibe ese ACK, puede calcular el **RTT = tiempo_actual - TSecr**.
  el **TSecr** actúa como un "espejo" del tiempo original, permitiendo que el emisor estime cuánto tardó en completarse el viaje ida y vuelta.

9. A-



B-

(IP: port) fuente	(IP: port) destino
10.0.2.10:46907	10.0.4.10:5001
10.0.2.10:45670	10.0.4.10:7002
10.0.2.10:45671	10.0.4.10:7002
10.0.2.10:46910	10.0.4.10:5001
10.0.2.10:54424	10.0.4.10:9000
10.0.2.10:54425	10.0.4.10:9000

C- ![[Pasted image 20250521093638.png]]
Las conexiones exitosas tienen los flags SYN y ACK en 1, las fallidas tienen los flags RST y ACK en 1.
D- La conexión la inicia 10.0.2.10:46907. El cliente es 10.0.2.10.46907 y el servidor es 10.0.4.5001.
3-way handshake:
![[Pasted image 20250521094115.png]]
Los ISN intercambiados son:
Para el segmento 3 son: ISN relativo = 0; ISN raw = 2218428254
![[Pasted image 20250521094328.png]]
Para el segmento 4 son: ISN relativo = 0; ISN raw = 1292618479
![[Pasted image 20250521094438.png]]
Para el segmento 5 son: ISN relativo = 1; ISN raw = 2218428255
![[Pasted image 20250521094536.png]]
El MSS que se negocia (en segmentos SYN y SYN-ACK) es 1460:
![[Pasted image 20250521095131.png]]
El host que envía mayor cantidad de datos es el 10.0.2.10:46907, podemos verlo examinando los números de secuencia de los segmentos que envía, crece

```
muchísimo durante la conexión.
E-![[Pasted image 20250521100106.png]]
Cantidad de datos que lleva (24 bytes):
![[Pasted image 20250521100133.png]]
ACK de los datos:
![[Pasted image 20250521100400.png]]
Se confirman 24 bytes y se espera el 25.
F- Lo inicia 10.0.2.10:46907, segmento con los flags FIN, PSH, ACK:
![[Pasted image 20250521100758.png]]
```

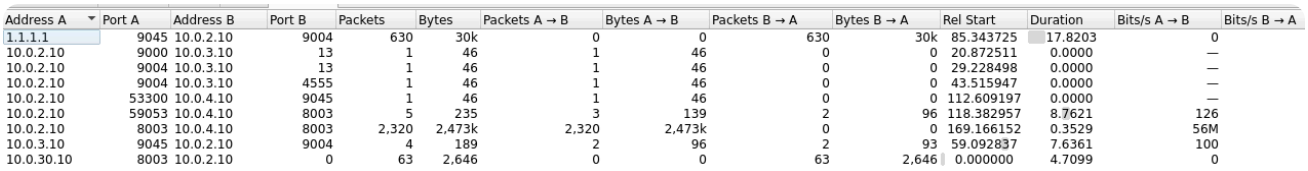
10. El **control de flujo** es un mecanismo del protocolo **TCP** diseñado para evitar que el emisor (sender) sobrecargue al receptor (receiver) enviándole más datos de los que puede procesar o almacenar.
    A- **Lo activa el receptor**. **Forma de activación:** el receptor indica cuántos bytes está dispuesto a recibir a través del campo **Window Size (Tamaño de Ventana)** en la cabecera TCP de cada segmento. Este valor se conoce como **ventana de recepción (receive window)**.
    Por ejemplo, si el receptor pone `Window Size = 1000`, le está diciendo al emisor que puede enviar hasta 1000 bytes más antes de necesitar confirmación.
    B- **Evita que el emisor sature la memoria (buffer) del receptor.** En otras palabras, evita la **pérdida de datos** y **retransmisiones innecesarias** debidas a que el receptor no pueda almacenar todo lo que el emisor le envía. Es crucial en redes heterogéneas donde el emisor puede ser mucho más rápido que el receptor.
    C- El control de flujo **está activo durante toda la conexión TCP**.
    - **Se ajusta dinámicamente** según el estado del receptor.
    - Puede **limitarse (casi detenerse)** si el receptor anuncia una ventana de tamaño 0 (cero), lo cual indica que no puede recibir más datos por el momento.
    - Se **"desactiva" (o se relaja)** cuando el receptor vuelve a anunciar una ventana mayor a 0, indicando que ha liberado espacio en su buffer y puede recibir más datos.
11. El **control de congestión** en TCP es un mecanismo **implementado por el emisor** para evitar **saturar la red** enviando más datos de los que puede manejar.
    A- **Lo activa el emisor (host que envía los datos). Es un mecanismo siempre activo** en las conexiones TCP modernas.
    Disparadores típicos para ajustar o modificar el comportamiento del control de congestión:
    - **Pérdida de paquetes:** inferida por la falta de ACKs o por la llegada de ACKs duplicados (por ejemplo, 3 ACKs duplicados).
    - **Timeout (retransmisión):** si el emisor no recibe un ACK a tiempo.
    - En ciertos algoritmos también puede considerar **variaciones en el RTT (Round Trip Time)**.

B- Evita la congestión de la red controlando la cantidad de datos en tránsito. Ayuda a mantener el rendimiento de la red sin provocar pérdidas masivas ni colapsos de routers o enlaces. Evita que múltiples conexiones sobrecarguen simultáneamente el mismo tramo de red.
C-

Característica	Slow Start	Congestion Avoidance
¿Cuando se usa?	Al inicio de la conexión o tras pérdida grave	Cuando se alcanza un umbral (ssthresh)
Crecimiento de la ventana	Exponencial (duplica por RTT)	Lineal (aumenta 1 MSS por RTT)
Objetivo	Rápida exploración de la capacidad inicial	Evitar congestión una vez estimada la red
Fin de la fase	Cuando el tamaño de la ventana es mayor o igual a ssthresh	Puede continuar indefinidamente

12. A-

En total hay 9 comunicaciones:



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
1.1.1.1	9045	10.0.2.10	9004	630	30k	0	0	630	30k	85.343725	17.8203	0	
10.0.2.10	9000	10.0.3.10	13	1	46	1	46	0	0	20.872511	0.0000	—	
10.0.2.10	9004	10.0.3.10	13	1	46	1	46	0	0	29.228498	0.0000	—	
10.0.2.10	9004	10.0.3.10	4555	1	46	1	46	0	0	43.515947	0.0000	—	
10.0.2.10	53300	10.0.4.10	9045	1	46	1	46	0	0	112.609197	0.0000	—	
10.0.2.10	59053	10.0.4.10	8003	5	235	3	139	2	96	118.382957	8.7621	126	
10.0.2.10	8003	10.0.4.10	8003	2,320	2,473k	2,320	2,473k	0	0	169.166152	0.3529	56M	
10.0.3.10	9045	10.0.2.10	9004	4	189	2	96	2	93	59.092837	7.6361	100	
10.0.30.10	8003	10.0.2.10	0	63	2,646	0	0	63	2,646	0.000000	4.7099	0	

B- Se ve con los mensajes ICMP que se envían a las no exitosas.
C-Aunque UDP es un protocolo sin conexión (connectionless), se puede implementar un modelo cliente/servidor de la siguiente forma:

- Servidor UDP:
  - Escucha en un puerto fijo esperando datagramas.
  - Usa una aplicación que "bindea" ese puerto (por ejemplo, DNS escucha en el puerto 53/UDP).
  - Responde a cada datagrama recibido con otro datagrama al cliente.
- Cliente UDP:
  - Envía un datagrama al servidor con una petición (ej: consulta DNS).
  - Espera una respuesta (aunque no hay garantía de recibirla).

D- ✅ Servicios y aplicaciones que usan UDP:

- DNS (Domain Name System)
  - Puerto: 53/UDP
  - Función: Resolver nombres de dominio a direcciones IP.
  - Requerimientos: Baja latencia, ya que las consultas deben ser rápidas. Puede tolerar reenvíos si la respuesta no llega.

- **DHCP (Dynamic Host Configuration Protocol)**
  - **Puertos**: 67 (servidor) y 68 (cliente)
  - **Función**: Asignar automáticamente direcciones IP a dispositivos.
  - **Requerimientos**: Comunicación simple entre cliente y servidor sin necesidad de una conexión establecida.
- **VoIP (Voice over IP) – como SIP, RTP**
  - **Puerto común**: variable (RTP entre 16384-32767)
  - **Función**: Transmisión de voz en tiempo real.
  - **Requerimientos**: Muy baja latencia; la pérdida ocasional de paquetes es preferible a los retrasos (no se retransmiten paquetes perdidos).
- **Streaming de video/audio (en tiempo real)**
  - **Ejemplos**: IPTV, transmisiones en vivo, Zoom, WebRTC.
  - **Requerimientos**: Fluidez continua. Prefieren omitir paquetes dañados antes que retrasar el flujo por retransmisión.
- **Juegos en línea**
  - **Función**: Envío de acciones del jugador y actualizaciones del estado del juego.
  - **Requerimientos**: Reacción instantánea. La pérdida ocasional es tolerable, pero los retrasos no.
- **TFTP (Trivial File Transfer Protocol)**
  - **Puerto**: 69/UDP
  - **Función**: Transferencia simple de archivos (a menudo usado en dispositivos de red).
  - **Requerimientos**: Simpleza, bajo overhead, especialmente útil en dispositivos con pocos recursos.
- **NTP (Network Time Protocol)**
  - **Puerto**: 123/UDP
  - **Función**: Sincronización de hora entre sistemas.
  - **Requerimientos**: Rápido y simple, no crítico si un paquete se pierde ocasionalmente.

E-El protocolo **UDP (User Datagram Protocol)** realiza un **control de errores mínimo**. Hace **suma de verificación (Checksum)**:

- UDP **incluye un campo de checksum de 16 bits** en su cabecera.
- Este checksum se calcula sobre:
  - La cabecera UDP.
  - Los datos del datagrama.
  - Una "pseudo-cabecera" que incluye partes de la cabecera IP (como IP origen/destino y el protocolo).
- **Función**: Detectar errores de transmisión (bits corruptos) en los datos o en la cabecera. Si el checksum no concuerda en el destino, el datagrama se **descarta** silenciosamente.

No hay reintentos ni notificaciones de error. No garantiza entrega, no realiza control de flujo ni de congestión, no hace ACKs de recibo ni mantiene conexión.
F- En UDP el puerto origen puede ser 0 si no se necesita una respuesta, puede simplemente enviar información.
G- La dirección IP y el puerto es: 10.0.2.10:9004. Se mandan 4 y 7 bytes

```
79 59.092837    10.0.2.10          10.0.3.10          UDP              46 9004 → 9045 Len=4
80 62.173832    10.0.3.10          10.0.2.10          UDP              49 9045 → 9004 Len=7
```

13. A- `hping3 -p 3306 --udp 10.100.25.135`
    - Envía un datagrama **UDP** al puerto **3306** (MySQL).
    - En la imagen, el puerto 3306 aparece en estado `LISTEN` para **TCP**, no UDP, por lo que el host responderá con un **ICMP Port Unreachable**.

    B- `hping3 -S -p 25 10.100.25.135`
    - Envía un segmento **TCP con flag SYN** al puerto **25** (SMTP).
    - En la imagen: No aparece ningún servicio escuchando en el puerto 25, por lo que el host **responderá con RST-ACK**, indicando que no hay nadie escuchando en el puerto 25.

    C- `hping3 -S -p 22 10.100.25.135`
    - Envía un **SYN TCP** al puerto 22 (SSH).
    - En la imagen: `tcp LISTEN *:22`, el puerto está abierto. El host **responderá con SYN-ACK**, indicando que el puerto está abierto.

    D- `hping3 -S -p 110 10.100.25.135`
    - Envía un **SYN TCP** al puerto 110 (POP3).
    - En la imagen: No aparece ningún servicio escuchando en el puerto **110**, por lo que el host **responderá con RST-ACK**, indicando que no hay nadie escuchando en el puerto 110.

    E- La cantidad de conexiones establecidas es: 3 (ya que entre las conexiones (aquellas que tienen estado ESTAB) hay 2 pares que son una misma conexión pero bidireccional, por lo que solo cuentan como 1 cada una)