



Universidad de
SanAndrés

I310 - Redes y Comunicación

**Trabajo Práctico D:
Desarrollo de aplicaciones distribuidas con BSD
sockets**

Ignacio Gremes, Timoteo Menceyra y Felipe Viaggio

25 de noviembre de 2025

Ingeniería en Inteligencia Artificial

1. Pregunta 1: Transferencia de archivo de 20 kB en diferentes escenarios

Se solicitó transferir un archivo de 20 kB utilizando el protocolo UDP stop & wait implementado, en tres escenarios con diferentes latencias.

Archivo de prueba:

- Tamaño: 20 kB = 20,480 bytes

Escenarios:

1. **LAN:** Red local virtual (RTT ~ 2 ms)
2. **Internacional:** Delay 90 ms cada sentido (RTT ~ 180 ms)
3. **Lunar:** Delay 1.28 s cada sentido (RTT ~ 2.56 s)

1.1. Deducción de la fórmula teórica

Cálculo del número de chunks:

El protocolo divide el archivo en chunks de 1,470 bytes (MAX_DATA_SIZE):

$$N_{chunks} = \left\lceil \frac{20,480}{1,470} \right\rceil = 14 \text{ chunks} \quad (1)$$

Número de RTTs necesarios:

El protocolo consta de 4 fases, cada una requiriendo un RTT:

- HELLO (autenticación): 1 RTT
- WRQ (nombre del archivo): 1 RTT
- DATA (14 chunks con stop & wait): 14 RTTs
- FIN (finalización): 1 RTT

Total: $N_{RTTs} = 1 + 1 + 14 + 1 = 17$ RTTs

Fórmula del tiempo de transferencia:

$$T_{\text{total}} = 17 \times RTT \quad (2)$$

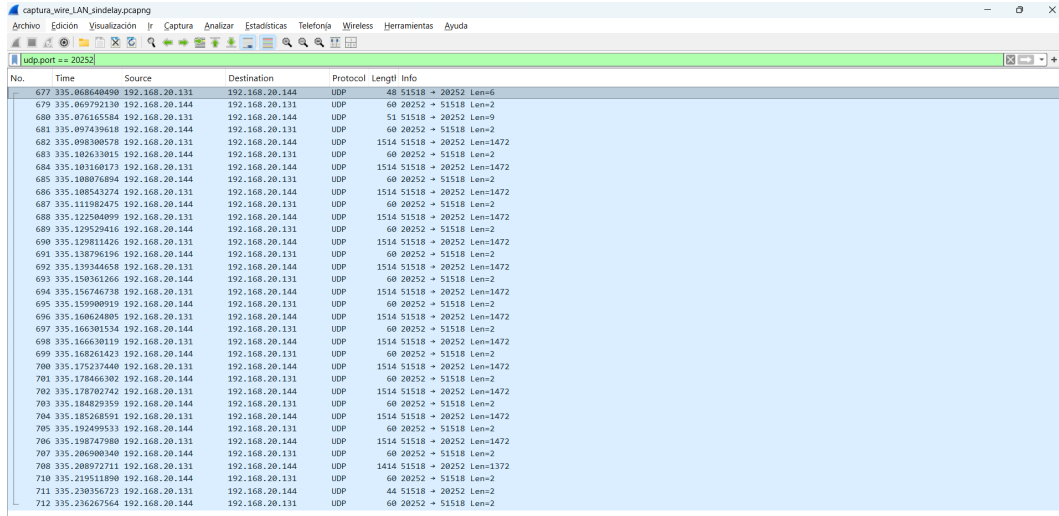
Fórmula generalizada para cualquier archivo de tamaño F bytes:

$$T_{\text{total}} = \left(\left\lceil \frac{F}{1,470} \right\rceil + 3 \right) \times RTT \quad (3)$$

1.2. Resultados experimentales

Escenario 1 - LAN:

- RTT: 2 ms
- Tiempo teórico: $17 \times 2 = 34$ ms
- Tiempo medido en Wireshark: 168 ms



No.	Time	Source	Destination	Protocol	Length	Info
677	3.35.068640490	192.168.20.131	192.168.20.144	UDP	48	151518 → 20252 Len=6
679	3.35.069792130	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
680	3.35.076165584	192.168.20.131	192.168.20.144	UDP	53	151518 → 20252 Len=9
681	3.35.097439618	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
682	3.35.098300578	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
683	3.35.102633015	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
684	3.35.103160173	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
685	3.35.108076894	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
686	3.35.108543274	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
687	3.35.111982475	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
688	3.35.122584099	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
689	3.35.129529416	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
690	3.35.129811426	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
691	3.35.138796196	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
692	3.35.139344658	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
693	3.35.150361366	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
694	3.35.156746738	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
695	3.35.159900919	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
696	3.35.160624805	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
697	3.35.166301534	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
698	3.35.166630119	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
699	3.35.168261423	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
700	3.35.175237440	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
701	3.35.178466302	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
702	3.35.178702742	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
703	3.35.184829359	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
704	3.35.185268591	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
705	3.35.192499533	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
706	3.35.198747980	192.168.20.131	192.168.20.144	UDP	1514	151518 → 20252 Len=1472
707	3.35.206000140	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
708	3.35.208972711	192.168.20.131	192.168.20.144	UDP	1414	151518 → 20252 Len=1372
710	3.35.219511890	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2
711	3.35.230356723	192.168.20.131	192.168.20.144	UDP	44	151518 → 20252 Len=2
712	3.35.236267564	192.168.20.144	192.168.20.131	UDP	60	20252 → 151518 Len=2

Figura 1: Captura Wireshark - Escenario LAN

Análisis: El protocolo es eficiente en LAN donde la latencia es despreciable. El tiempo real es mayor que el teórico por el procesamiento del sistema operativo y la escritura del archivo a disco.

Escenario 2 - Internacional:

- RTT: 180 ms
- Tiempo teórico: $17 \times 180 = 3,060$ ms = 3.06 s
- Tiempo medido en Wireshark: 3.34 s

No.	Time	Source	Destination	Protocol	Length	Info
303	126.728884686	192.168.20.131	192.168.20.144	UDP	48	54483 → 20252 Len=6
306	127.086106874	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
307	127.087177877	192.168.20.131	192.168.20.144	UDP	52	54483 → 20252 Len=10
308	127.215872293	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
309	127.217247384	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
310	127.406072268	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
311	127.407197985	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
313	127.592725098	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
314	127.592828087	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
315	127.782825683	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
316	127.784157346	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
317	127.969334878	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
318	127.972153216	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
319	128.162672312	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
321	128.171680032	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
322	128.359310145	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
323	128.362139333	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
324	128.549155143	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
325	128.552864859	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
327	128.742292925	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
328	128.743974539	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
329	128.932442274	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
330	128.932550998	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
332	129.122794555	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
333	129.124156033	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
334	129.309051132	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
335	129.310160985	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
336	129.495498853	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
337	129.496409547	192.168.20.131	192.168.20.144	UDP	1514	54483 → 20252 Len=1472
339	129.688859183	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
340	129.688964554	192.168.20.131	192.168.20.144	UDP	1414	54483 → 20252 Len=1372
341	129.875832556	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2
342	129.877195438	192.168.20.131	192.168.20.144	UDP	44	54483 → 20252 Len=2
344	130.065894653	192.168.20.144	192.168.20.131	UDP	60	20252 → 54483 Len=2

Figura 2: Captura Wireshark - Escenario Internacional

Análisis: El protocolo ya empieza a mostrar ineficiencia. Con latencia de 90ms en cada sentido, transferir 20 kB tarda más de 3 segundos. El canal está la mayor parte del tiempo esperando ACKs en lugar de transmitiendo datos.

Escenario 3 - Lunar:

- RTT: 2.56 s
- Tiempo teórico: $17 \times 2,560 = 43,520$ ms = 43.52 s
- Tiempo medido en Wireshark: 44.05 s

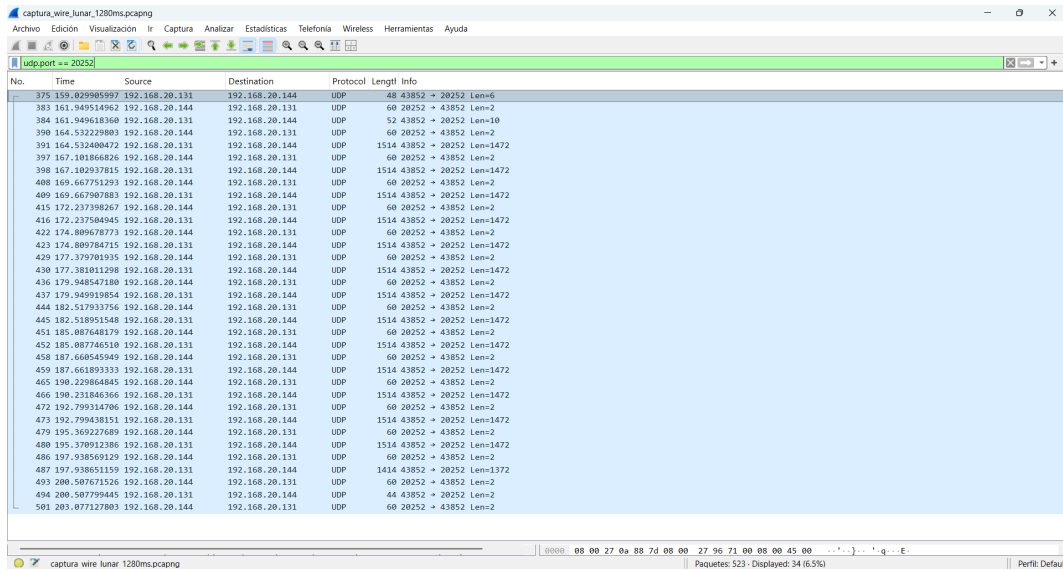


Figura 3: Captura Wireshark - Escenario Lunar

Análisis: El protocolo es totalmente inadecuado para enlaces de alta latencia. Transferir apenas 20 kB tarda casi 45 segundos porque después de cada paquete de 1470 bytes hay que esperar 2.56 segundos para la confirmación. El canal está prácticamente todo el tiempo ocioso esperando ACKs en lugar de transmitiendo datos.

1.3. Comparación de resultados

Cuadro 1: Comparación de escenarios

Escenario	RTT	T. Teórico	T. Real
LAN	2 ms	34 ms	168 ms
Internacional	180 ms	3.06 s	3.34 s
Lunar	2.56 s	43.52 s	44.05 s

Conclusiones:

- La fórmula $T = 17 \times RTT$ predice correctamente el tiempo de transferencia
- El protocolo stop & wait es apropiado solo para redes LAN de baja latencia
- Con latencia alta, el canal está ocioso la mayor parte del tiempo esperando ACKs
- Para enlaces de alta latencia se requieren protocolos con ventana deslizante

2. Pregunta 2: ¿Recomendaría este protocolo para actualizar firmware de 16 MB desde Buenos Aires a Virginia (EEUU)?

No, definitivamente no lo recomendaría.

Cálculo del tiempo de transferencia:

La latencia típica Buenos Aires - Virginia es $RTT \sim 250$ ms. Para 16 MB (16,777,216 bytes):

$$N_{chunks} = \left\lceil \frac{16,777,216}{1,470} \right\rceil = 11,411 \text{ chunks} \quad (4)$$

$$N_{RTTs} = 1 + 1 + 11,411 + 1 = 11,414 \text{ RTTs} \quad (5)$$

$$T_{total} = 11,414 \times 250 \text{ ms} \approx \boxed{48 \text{ minutos}} \quad (6)$$

El tiempo de 48 minutos para transferir solo 16 MB es extremadamente lento. Como vimos en la Pregunta 1, el protocolo stop & wait tiene el problema fundamental de que debe esperar la confirmación de cada paquete antes de enviar el siguiente. Con 11,411 chunks y una latencia de 250 ms, esto hace que el canal esté la mayor parte del tiempo ocioso esperando ACKs en lugar de transmitiendo datos.

Además, actualizar firmware es una operación crítica donde el dispositivo queda vulnerable durante toda la transferencia. Si en esos 48 minutos hay un corte de red o cualquier problema, la transferencia falla y hay que empezar desde cero porque nuestro protocolo no tiene forma de reanudar. Con más de 11,000 transmisiones, la probabilidad de que algo salga mal aumenta considerablemente.

Para este tipo de transferencias largas sobre enlaces con latencia alta, se necesitan protocolos más eficientes como TCP que usan ventanas deslizantes y pueden enviar múltiples paquetes sin esperar confirmación de cada uno. Esos protocolos completarían la misma transferencia en aproximadamente 2-3 minutos, siendo más de 20 veces más rápidos y con mecanismos de recuperación ante errores.

3. Pregunta 3: ¿Por qué TCP no es adecuado para aplicaciones de tiempo real con packet loss?

Respuesta: TCP no es adecuado para aplicaciones de tiempo real en entornos con pérdida de paquetes.

El problema principal es que cuando TCP detecta la pérdida de un paquete, debe retransmitirlo antes de entregar los datos siguientes a la aplicación (head-of-line blocking). Esto genera retrasos impredecibles de 300-700ms con solo 5 % de pérdida. Además, el jitter crece significativamente ante pérdida, volviendo impredecibles los tiempos de entrega. Para aplicaciones de tiempo real como videoconferencias es preferible tener delay constante que variaciones impredecibles, incluso si eso significa perder algunos paquetes.

Ejemplos de entornos con alta pérdida incluyen redes inalámbricas como 4G/5G debido al movimiento de usuarios que causa interferencias, y comunicaciones satelitales por la gran distancia y condiciones atmosféricas.