Clase Práctica : Protocolos Punto a Punto

Tomás F. Melli

August 2025

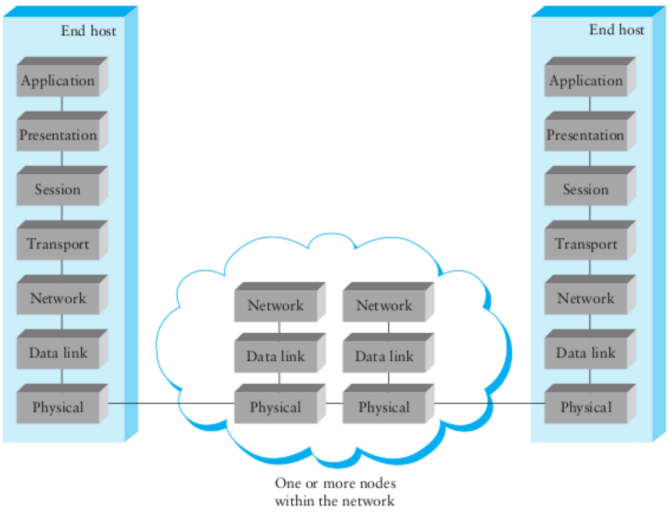
Índice

1 Arquitectura en capas

Ya hablamos de la transmisión de información la clase pasada, pero para que realmente se entiendan dos dispositivos, deben enfrentar la **vulnerabilidad al ruido impulsivo** que generar errores de transmisión. Dando lugar a una serie de complejidades como:

- **Control de errores** : si se produjo un error, cómo lo detectamos ? Cómo lo corregimos ?
- **Control de flujo** : cómo regulamos la velocidad a la que un emisor envía datos para no saturar al receptor?
- **Framing** : cómo agrupamos los bits que mandamos a través del **caño serial** de manera que el receptor entienda dónde empieza y dónde termina el mensaje ? Agregamos información de control ? Dónde ?
- **Proveer servicio a la capa superior** : se puede ofrecer un servicio **confiable** (la capa de enlace detecta errores y los corrige) o **no confiable** (la capa de enlace solo detecta errores)

Es en este momento en que aparece la **arquitectura en capas** para resolver todas estas complejidades. Cada capa ofrece servicios a la capa superior y utiliza los servicios de la capa inferior. Este enfoque modular simplifica el diseño, facilita la interoperabilidad entre diferentes tecnologías y permite aislar problemas en un nivel sin afectar todo el sistema. El modelo de referencia es el **OSI (Open Systems Interconnection)** que se organiza en **7 capas** desde la más cercana al usuario hasta la más cercana al medio físico de transmisión.

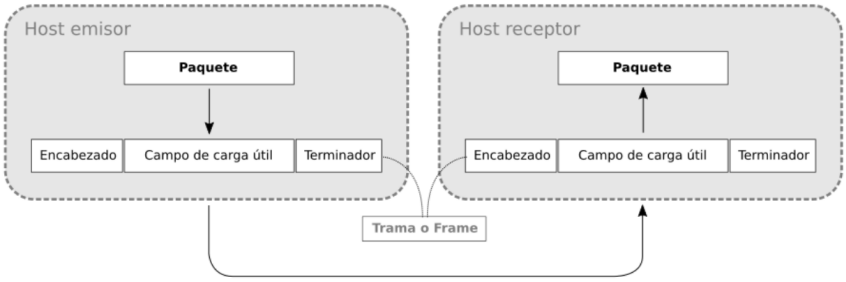


Brevemente describimos qué hace cada una (de abajo hacia arriba) :

- **Physical** : Se encarga de la transmisión real de bits a través del medio físico (cables, fibra óptica, radio, etc.).
- **Data Link** : Garantiza una comunicación libre de errores entre dos nodos directamente conectados. Se ocupa de la detección/corrección de errores y del control de acceso al medio (MAC).
- **Network** : Determina la ruta que seguirán los datos en la red. Se encarga de direccionamiento lógico (como las direcciones IP) y del enrutamiento entre nodos.
- **Transport** : Proporciona una comunicación extremo a extremo confiable. Gestiona la segmentación de datos, el control de flujo y la corrección de errores. Protocolos como TCP, UDP
- **Session** : Administra y sincroniza el diálogo entre aplicaciones. Establece, mantiene y termina sesiones de comunicación.
- **Presentation** : Se encarga de la traducción y transformación de datos. Incluye compresión, encriptación y conversión de formatos (por ejemplo, entre diferentes sistemas de codificación).
- **Application** : Es la más cercana al usuario. Proporciona servicios de red a las aplicaciones, como correo electrónico, transferencia de archivos, navegación web.

2 Framing

El Framing es el proceso de agrupar un flujo continuo de bits en unidades reconocibles llamadas **frames**.



2.1 Ejercicio 1

Dado un enlace punto a punto a la luna de 1 Mbps con un delay de 1.25 segundos:

a. ¿Cuántos bits entran en el enlace?

b. Asumiendo que se separan los bits en frames de largo fijo de 1 Kb, ¿cuántos frames entran en el enlace?

Resolución

a. La ecuación que determina cuántos bits entran en el canal está dada por:

$$C_{vol} = V_{tx} \times Delay$$

Dados $V_{tx} = 1\,\text{Mbps}$ y $Delay = 1.25\,\text{s}$, tenemos que:

$$C_{vol} = 1.25\,\text{Mb}$$

Es decir, entran 1.25 millones de bits en el canal.

b. Para frames de largo fijo de 1 Kb tenemos:

$$\frac{1.25\,\text{Mb}}{1\,\text{Kb}} = 1250 \text{ frames}$$

2.2 Framing por Encapsulamiento

Cuando tenemos un flujo continuo de bits en un enlace punto a punto, el receptor necesita saber cómo separar esos bits en unidades lógicas (frames). O sea, dónde empieza y dónde termina. Existen varias técnicas :

2.2.1 Largo fijo

Cada frame tiene siempre la misma longitud en bits/bytes. El receptor simplemente cuenta los bits: cada "N" bits corresponde a un frame. Lo bueno es que es muy simple. Lo malo es que no se adapta bien a mensajes de diferente tamaño, ya que si el largo es enorme y la data que mandamos es despreciable en comparación al tamaño del frame desperdiciamos espacio.

2.2.2 Largo en el encabezado

Cada frame comienza con un campo especial (header o encabezado) que indica el tamaño del frame. El receptor lee el encabezado, sabe cuántos bits/bytes debe recibir, y así separa cada frame. Lo bueno es que es flexible, cada frame puede tener un tamaño distinto. Lo malo es que por sí solo, si se corrompe la información de ese header, vamos a necesitar implementar un mecanismo de detección de errores como **CRC (Cyclic Redundancy Check)**. Más abajo lo vemos mejor :).

2.2.3 Delimitadores con *bit-stuffing*

Cada frame empieza y termina con una secuencia especial de bits.¿qué pasa si esa secuencia aparece dentro de los datos? Se usa **bit-stuffing**, el transmisor inserta un bit adicional (normalmente un 0) dentro de la carga útil para evitar que se confunda con el delimitador. El receptor, al ver la secuencia, elimina ese bit extra para recuperar los datos originales. Lo bueno es que es muy robusto, permite frames de tamaño variable sin desperdicio. Lo malo es que requiere procesamiento adicional (insertar/quitar bits).

2.3 Eficiencia del frame

Con esto de agregar información de control, queremos evaluar qué tan eficiente es mandar información en un frame, para ello surge esta cuenta :

$$\eta_{\texttt{frame}} = \frac{\texttt{largo de los datos}}{\texttt{largo total del frame}}$$

Donde el **largo de los datos** es la longitud del **payload** o sea, la información útil, y el **largo del frame** incluye payload + encabezado + CRC + otros campos de control. Por tanto, lo que hace esta cuenta es *qué fracción del frame ocupan los datos útiles.*

2.4 Ejercicio 2

Calcule la eficiencia del frame tomando en cuenta solo el overhead impuesto por las siguientes técnicas de framing:

a. Largo fijo.

b. Campo de 16 bits en el encabezado indicando el largo del frame.

c. Delimitadores de 8 bits usando bit-stuffing.

2.5 Resolución

a. Siendo

$$\eta_{\text{frame}} = \frac{\text{largo de los datos}}{\text{largo total del frame}}$$

en este caso tenemos que:

$$\eta_{\text{frame}} = \frac{\text{largo de los datos}}{\text{largo de los datos}} = 1$$

b. Para campos en el encabezado tenemos que:

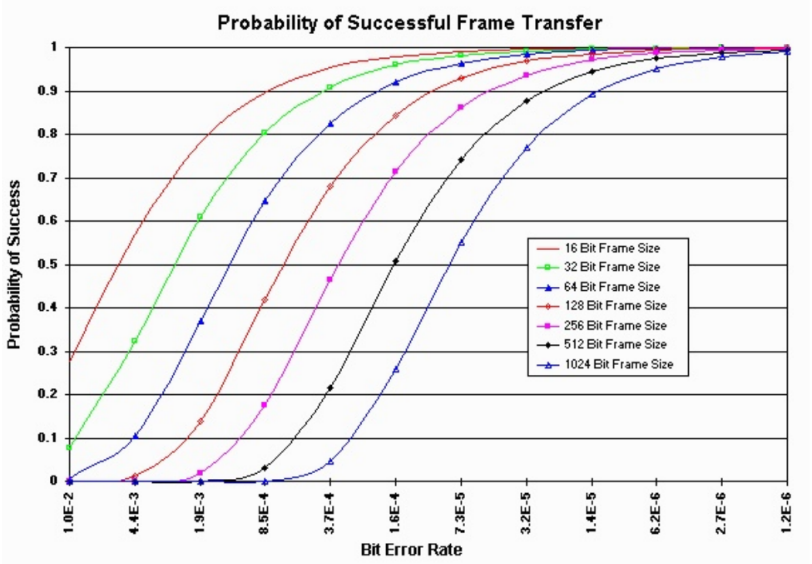
$$\eta_{\text{frame}} = \frac{\text{largo del frame} - |\text{camposize}|}{\text{largo del frame}} = \frac{\text{largo del frame} - 16}{\text{largo del frame}}$$

c. Este último caso es variable, ya que depende de la cantidad de bits que haya que incorporar al frame debido al bit-stuffing. Siendo n la cantidad de bits, nos queda:

$$\eta_{\text{frame}} = \frac{\text{largo del frame} - 8 - n}{\text{largo del frame}}$$

2.6 Frames de largo fijo : probabilida de error

Cuando hablamos de frames de largo fijo, cada frame tiene un número fijo de bits, digamos L bits. Si hay errores de transmisión, por ejemplo un bit puede cambiar con probabilidad p, entonces la probabilidad de que el frame completo llegue sin errores depende directamente de cuántos bits tiene el frame.



La probabilidad de que cada bit llegue bien es $1 - p$. Como todos los bits deben estar correctos para que el frame llegue bien, la probabilidad de que el frame completo llegue sin errores es:

$$P_{frame\ correcto} = (1 - P)^L$$

El tema es que si usamos frames largos tendremos menor probabilidad de éxito, porque hay más bits que pueden fallar. Pero si usamos frames cortos, aunque tengamos mayor probabilidad de éxito, necesitaremos más encabezados y control, lo que aumenta el overhead.

3 Control de errores

El control de errores es un conjunto de técnicas que permiten detectar y, en algunos casos, corregir errores en los datos transmitidos o almacenados. Esto asegura la integridad y confiabilidad de la información. Vamos a describir brevemente algunas de las técnicas.

- **Bit de paridad**:
  Se agrega un bit adicional a un bloque de datos para que el número total de bits en "1" sea par (paridad par) o impar (paridad impar).
  Permite detectar errores simples de un solo bit, pero no puede corregirlos ni detectar errores múltiples con certeza.
- **CRC (Cyclic Redundancy Check)**:
  Calcula un valor de verificación basado en la división polinómica de los datos por un polinomio generador.
  Es muy eficaz para detectar errores, no corrige errores por sí mismo, pero puede indicar cuándo retransmitir datos.
- **Checksum**:
  Suma de los valores de los datos (por bytes o palabras) que se envía junto con el bloque de datos.
  Permite detectar errores simples, pero es menos fiable que CRC para detectar errores.
- **Código de Hamming**:
  Introduce bits de control adicionales que permiten detectar y corregir errores de un solo bit en un bloque de datos.
  Permite cierta corrección automática sin necesidad de retransmisión.
- **Reed-Solomon**:
  Código de corrección de errores que trabaja con bloques de símbolos, no solo bits.
  Muy usado en CDs, DVDs y transmisión digital de televisión, porque puede corregir errores múltiples dentro de un bloque.
- **MD5 (Message Digest 5)**:
  Función de hash criptográfico que genera un resumen de los datos (digest).
  Permite detectar cambios o corrupciones en los datos, pero no corrige errores.
  Usado principalmente para verificación de integridad de archivos y mensajes.

4 Retransmisiones

En las comunicaciones digitales, no siempre todos los datos llegan correctamente al receptor debido a errores o pérdidas de paquetes. Las retransmisiones son mecanismos que **permiten volver a enviar datos que no fueron recibidos correctamente**, asegurando la integridad de la información y la fiabilidad del enlace. Existen principalmente dos tipos:

4.1 Explícitas

Se utilizan mensajes de control específicos (por ejemplo, ACK/NACK) enviados por el receptor al emisor. Cuando el receptor detecta un error o una pérdida de datos, envía un mensaje solicitando la retransmisión de los datos afectados. Este método es muy confiable, porque el emisor sabe exactamente qué datos deben reenviarse.

4.2 Implícitas

No se requiere un mensaje de control adicional del receptor. El emisor supone que los datos se perdieron si no recibe una confirmación dentro de un cierto tiempo (**timeout**). Cuando ocurre el timeout, se reenvían automáticamente los datos. Es más simple que la retransmisión explícita, pero depende de la estimación del tiempo de espera, y puede generar retransmisiones innecesarias si el timeout es muy corto.

5 Tipos de servicio

En redes de comunicación, los tipos de servicio definen **cómo se envían los datos y qué nivel de fiabilidad se garantiza**. Dependiendo de la aplicación y del protocolo, se pueden elegir distintos mecanismos para asegurar o no la entrega correcta de los datos. Los principales tipos de servicio son :

5.1 Sin conexión y sin reconocimiento

Los datos se envían sin establecer previamente una conexión ni asegurarse de que lleguen correctamente. No hay mensajes de confirmación (ACK) ni retransmisiones automáticas. Es rápido y simple, pero no garantiza la entrega de información.

5.2 Sin conexión y con reconocimiento

Los datos se envían sin establecer una conexión previa, pero el emisor asegura la correcta recepción mediante mensajes explícitos de reconocimiento (ACKs). Permite detectar y retransmitir datos perdidos, aunque no mantiene un estado de sesión.

5.3 Orientado a conexión

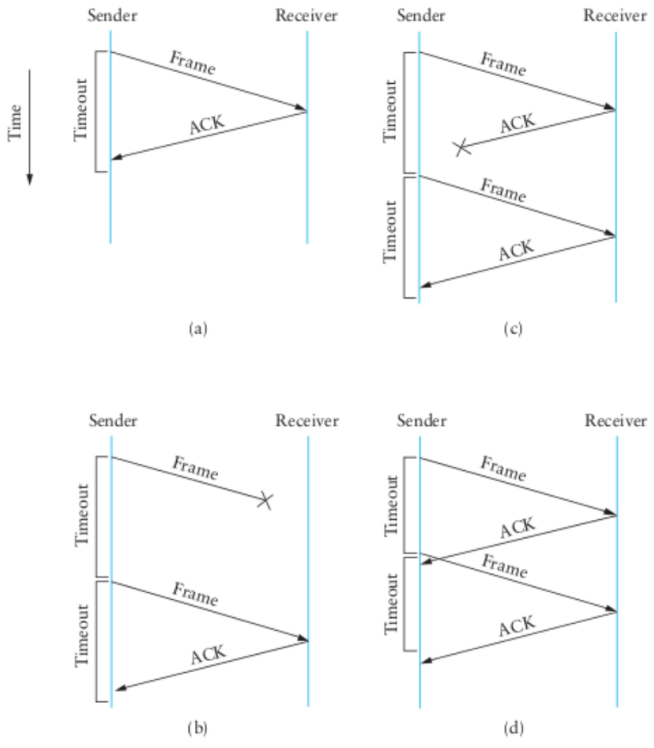
Antes de enviar datos, se establece una conexión o sesión entre emisor y receptor. Además de asegurar la correcta recepción de los datos, se mantiene un **estado** de la conexión durante toda la comunicación.

6 Transmisión confiable

Una transmisión confiable es un mecanismo o conjunto de protocolos que aseguran que los datos enviados desde un emisor lleguen correctamente al receptor, incluso si ocurren errores, pérdidas o duplicaciones durante la transmisión. Veamos ejemplos :

6.1 Stop and Wait

Es un protocolo de retransmisión confiable usado a nivel de enlace o transporte. Funciona bloque a bloque, es decir, el emisor envía un frame (o paquete) y espera un ACK antes de enviar el siguiente. Si el ACK no llega dentro de un tiempo límite (timeout), el emisor reenvía el mismo frame. Como se ve a continuación :

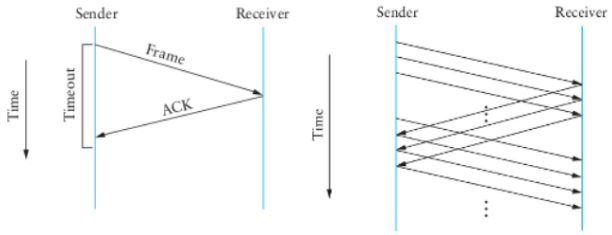


Entoces, la secuencia es

1. El emisor envía un frame con número de secuencia.
2. El receptor recibe el frame y verifica si llegó correctamente (por CRC o checksum).
3. Si el frame es correcto, el receptor envía un ACK al emisor.
4. El emisor recibe el ACK y envía el siguiente frame.
5. Si ocurre un error o no llega el ACK antes del timeout, el emisor retransmite el frame.

6.2 Sliding Window

Es un protocolo de transmisión confiable que permite enviar múltiples frames antes de recibir ACKs, en lugar de enviar un frame y esperar cada confirmación como en Stop-and-Wait. Se basa en una ventana deslizante que controla cuántos frames pueden estar "en viaje" sin ser reconocidos (recibir el ACK). Como se ve en la imagen :



El funcionamiento es el siguiente :

1. El emisor tiene una ventana de tamaño N. (Puede enviar hasta N frames consecutivos sin esperar ACK)
2. El receptor también tiene una ventana, indicando qué frames puede aceptar.
3. Cuando el receptor recibe correctamente un frame, envía un ACK (o ACK acumulativo).
4. La ventana se "desliza", los frames confirmados se eliminan de la ventana y se abren posiciones para nuevos frames.
5. Si un frame se pierde o llega con error, solo los frames a partir de ese punto se retransmiten (según la variante del protocolo: Go-Back-N o Selective Repeat, más abajo está explicado).

Tenemos dos métricas que nos permiten analizar el comportamiento de este protocolo:

- **Ventana de emisión** : La ventana de emisión es el número máximo de frames que el emisor puede enviar sin recibir ACKs.

  $$\text{SWS} = \frac{V_{\text{tx}} \cdot \text{RTT}}{|\text{Frame}|}$$

  donde:

  - V_{tx} = velocidad de transmisión del canal (bits/s)
  - RTT = tiempo de ida y vuelta (Round Trip Time) del enlace
  - $|\text{Frame}|$ = tamaño de cada frame en bits

- **Condición para enviar un nuevo frame** : No se puede enviar cualquier frame arbitrariamente; hay que respetar la ventana:

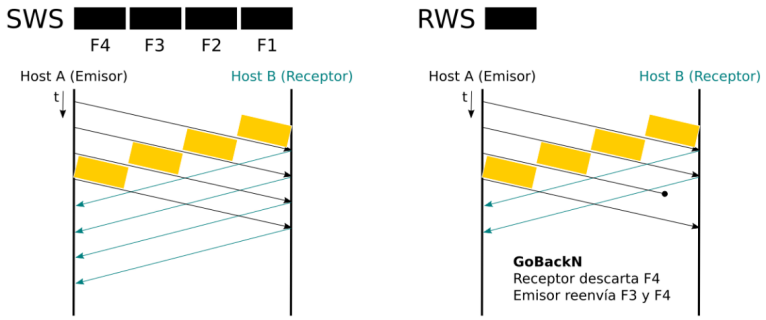
  $$\text{ÚltimoFrameEnviado} \leq \text{ÚltimoFrameReconocido} + \text{SWS}$$

  Esto significa que el emisor puede enviar frames dentro de la ventana activa, pero debe esperar ACKs si se alcanza el límite. Así se evita sobrecargar el receptor y mantener el orden correcto de los frames.

6.3 Variantes

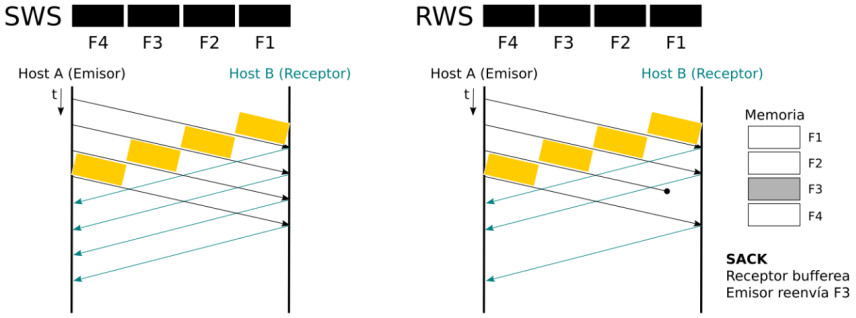
6.3.1 Go Back N

Si un frame se pierde, se retransmiten todos los frames posteriores, incluso si algunos llegaron bien.



6.3.2 Selective Repeat

Solo se retransmiten los frames que fallaron, más eficiente que Go-Back-N en canales con errores frecuentes.



6.4 Ejercicio 3

Diseñe posibles conjuntos de frames para los siguientes tipos de protocolos, asumiendo que se detectan errores usando CRC (No hace falta aclarar el largo de los campos):

1. Stop & Wait
2. Sliding Window con GoBackN usando Piggybacking
3. Sliding Window con ACK Selectivo

Resolución

1. Para el primer caso podríamos definir un frame para el emisor y otro para el receptor de la siguiente forma:
   - Emisor: #SEQ(1bit); Datos; Checksum
   - Receptor: #ACK(1bit); Checksum
2. Para el segundo caso, al haber Piggybacking todos los frames pueden llegar a tener que cumplir simultáneamente los roles de emisión y recepción. Entonces proponemos un único frame como el siguiente:
   - #SEQ; #ACK; Datos; Checksum
3. Para el último caso, a diferencia del punto anterior tenemos reconocimento selectivo. Para estos casos se recomienda, por un tema de eficiencia, que el frame receptor reconozca los frames tanto acumulativa como individualmente. Entonces nos quedarían dos frames como los siguientes:
   - Emisor: #SEQ; Datos; Checksum
   - Receptor: #ACK; #SACK; Checksum

6.5 Ventana de emisión o SWS ((Send Window Size)

Cuando transmitimos datos mediante protocolos de ventana deslizante (Sliding Window), no siempre conviene enviar un frame y esperar la confirmación antes de continuar (como en Stop-and-Wait). En su lugar, podemos aprovechar mejor el canal enviando varios frames en tránsito a la vez, hasta un cierto límite llamado **ventana de emisión**. El tamaño de esta ventana depende de las características del canal y del protocolo:

- **Velocidad de transmisión** (Vtx): cuántos bits por segundo podemos enviar.
- **RTT (Round Trip Time)**: el tiempo que tarda un frame en ir y volver con su ACK.
- **Tamaño de frame** ($| Frame |$): cuántos bits tiene cada frame.

Para hacer este cálculo, la fórmula es

$$SWS = \frac{V_{tx} \times RTT}{| Frame |}$$

Donde, $V_{tx} \times RTT$ nos dice la cantidad de bits que podemos inyectar en el canal durante el tiempo que tarda en llegar un ACK. Y luego, divir lo anterior por el tamaño de frame nos dice cuántos frames completos podemos enviar en ese lapso.

6.6 Ventana de recepción o RWS (Reception Window Size)

El receptor también define su propia ventana, que es la cantidad de frames que puede almacenar sin perderlos:

$$F_{RWS} = \begin{cases} SWS & \texttt{si hay SACK (Selective Acknowledgment)} \\ 1 & \texttt{si no lo hay (Go-Back-N)} \end{cases}$$

Si hay SACK, el receptor puede guardar varios frames fuera de orden y luego reordenarlos, por lo que conviene que su ventana de recepción sea tan grande como la del emisor.
Si no hay SACK (caso Go-Back-N), el receptor descarta cualquier frame que llegue fuera de orden. Entonces con un buffer de 1 solo frame es suficiente, porque nunca necesita almacenar más.

6.7 Número de secuencia mínimo necesario

El último detalle es que los números de secuencia deben ser suficientes para distinguir entre frames "viejos" y "nuevos" (**reencarnaciones**). La condición es:

$$\#frames \geq SWS + RWS$$

Si el rango de números de secuencia es menor, el receptor podría confundir un frame retransmitido (viejo) con uno nuevo, lo que generaría errores en la comunicación.

6.8 Eficiencia

En los protocolos de transmisión de datos (como Stop-and-Wait, Go-Back-N o Selective Repeat), la eficiencia mide qué tan bien se aprovecha el canal. Durante un tiempo, el emisor está transmitiendo datos. Luego, debe quedarse bloqueado esperando ACKs antes de poder seguir enviando. Cuanto mayor sea la proporción de tiempo útil (transmitiendo) frente al tiempo ocioso (esperando), más eficiente es el protocolo.
Definimos la siguiente fórmula : Se define como:

$$Eficiencia = \frac{T_{tx}(V)}{RTT(F)}$$

donde:

- $T_{tx}(V)$ = tiempo que tarda en transmitirse una ventana completa de datos (es decir, SWS frames).
- $RTT(F)$ = tiempo de ida y vuelta (Round Trip Time) para un frame (desde que se envía hasta que llega el ACK).

En otras palabras, la eficiencia es el cociente entre el tiempo aprovechado transmitiendo y el tiempo total del ciclo transmisión + espera.

- Si la eficiencia es cercana a 1 (100%), significa que el emisor está transmitiendo casi todo el tiempo, sin quedarse bloqueado.

- Si la eficiencia es muy baja, el emisor pasa demasiado tiempo esperando, desaprovechando la capacidad del canal.

Para aumentar la eficiencia, buscamos estar bloqueados lo menos posible. Esto se logra:

- Aumentando el tamaño de la ventana (SWS), de forma que mientras esperamos ACKs podamos seguir enviando frames.
- Reduciendo el RTT (por ejemplo, usando un canal más rápido o con menos latencia).
- Optimizando el tamaño de frame, para que la relación transmisión/espera sea más favorable.

6.9 Ejericicio 4

Un protocolo sobre un enlace punto a punto de 1Mbps y 0.25 segundos de delay, trabaja con ventana deslizante con GoBackN usando frames de largo fijo 2Kb y un CRC de 16bits para detectar errores.

1. Calcule cuáles son los tamaños de ventana de emisión y recepción óptimos.
2. ¿Cuántos bits hacen falta para secuenciar los frames?
3. Calcule cuánto tiempo es necesario para transmitir 20Mb de datos asumiendo que no hay errores.

Resolución

1. Es un protocolo de ventana deslizante GoBackN. Esto nos indica que por definición $RWS = 1$. Para el tamaño de la ventana de emisión tenemos:

$$SWS = \frac{V_{tx} \cdot 2 \cdot Delay}{|Frame|} = \frac{1 \text{ Mbps} \cdot 2 \cdot 0,25 \text{ s}}{2 \text{ Kb}} = 250 \text{ frames}$$

2. $SWS + RWS \leq \#Frames \;\; \Rightarrow \;\; \#SEQ \geq 251$ Con 8 bits alcanza para secuenciar los frames.

3. Veamos que nos piden transmitir 20Mb. Pero además sabemos que no hay errores en la transmisión. Entonces, como ya calculamos el tamaño de ventana que maximiza la eficiencia del protocolo, podemos asumir que la eficiencia se aproxima a 1. Con todos estos datos, podemos asumir que podemos transmitir todo el tiempo, que no vamos a bloquearnos esperando reconocimientos y que no va a haber errores.

   Además, tenemos que tener en cuenta la eficiencia del frame, debido a que no todos los bits que componen el frame son utilizados para los datos. La cantidad de bits del campo de datos en cada frame sería:

   $$|Datos| = |Frame| - |\#Seq| - |CRC| = 2000 - 8 - 16 = 1976$$

   Ahora bien, si para transmitir 1976 bits necesitamos un frame, entonces para transmitir 20Mb necesitaríamos:

   $$\frac{20\,000\,000}{1976} = 10121,45 \;\; \Rightarrow \;\; 10122 \text{ frames}$$

   Finalmente, bajo estas condiciones, transmitir 20Mb no es otra cosa que el:

   $$Delay(10122 \text{ frames}) = T_{tx}(10122 \text{ frames}) + T_{prop}$$

   $$= \frac{2000 \cdot 10122 \text{ bits}}{1\,000\,000 \text{ bps}} + 0,25 \text{ s} = 20,244 \text{ s} + 0,25 \text{ s} = 20,494 \text{ s}$$

   Si a esto le sumamos el delay del último ACK, la respuesta final es:

   $$20,744 \text{ s}$$

6.10 Ejercicio 5

Un protocolo confiable punto a punto que usa sliding window, opera sobre un canal de 10 Mbps, usa SACK y un frame emisor de 5Kb como el siguiente:
#SEQ (8 bits) ; Datos ; Checksum

1. Proponga un frame para el receptor.
2. ¿Cuál es el valor de delay para el cual el protocolo presenta un 100 % de eficiencia?
3. Si el delay fuera de 1 seg, ¿Cuántos bits deberían ocupar los números de secuencia de manera de maximizar la eficiencia?

Resolución

1. Al no presentarse requerimientos que ameriten el uso de piggybacking, no vamos a usar esta técnica para el framing. El protocolo va a ser asimétrico, en el sentido de que el frame emisor y el receptor van a ser distintos.

   El número de secuencia del frame emisor ocupa 8 bits, esto implica que tanto el campo ACK del receptor como el campo SACK deben ocupar la misma cantidad de bits: 8 bits para ACK y 8 bits para SACK. Esto es porque es necesario poder referenciar en los reconocimientos (ACKs) a cada frame que envía el emisor. Nota: no es necesario para la correctitud del algoritmo de ventana deslizante usar SACK y ACK cuando hay reconocimientos selectivos. Sin embargo se estila usar de este modo porque mejora el desempeño del protocolo.

   El frame receptor propuesto sería entonces:

   $$\#ACK(8 \text{ bits}); \#SACK(8 \text{ bits}); Checksum(16 \text{ bits})$$

2. Para encontrar el valor de delay óptimo hay que tratar de llenar el canal y que no se desperdicie tiempo estando bloqueado sin poder transmitir. Veamos los datos que tenemos:

   Tenemos 256 frames posibles porque el campo de #SEQ es de 8 bits. Pero, para evitar el problema de las reencarnaciones, solo 128 frames podrán ser enviados pues se usa SACK $\Rightarrow SWS = 128 = RWS$.

   Luego, usamos los datos que tenemos en la fórmula de eficiencia para el caso en cuál es óptima:

   $$1 = \frac{T_{tx}(SWS)}{RTT(F)} = \frac{T_{tx}(F) \cdot SWS}{Delay(Fe) + Delay(Fr)}$$

   $$Delay(5Kb) + Delay(32 \text{ bits}) = \frac{5\text{Kb}}{10 \text{ Mbps}} \cdot 128$$

   Resolviendo se obtiene que aproximadamente:

   $$Delay \approx 0.031 \text{ s}$$

   es el valor que maximiza la eficiencia del protocolo.

3. Para resolver este ejercicio se recalcula el tamaño de la ventana de emisión para ajustarla al delay requerido:

   $$SWS = \frac{V_{tx} \cdot 2 \cdot Delay}{|Frame|}$$

   Reemplazando los datos y resolviendo se obtiene que $SWS = 4000$. Luego, $\#Frames \geq 8000$ y tomando el techo de $\log_2(8000)$ se obtiene que se necesitan 13 bits.