Instituto Tecnológico de Costa Rica

Centro Académico de Alajuela

IC-7602: Redes

Proyecto:

Simulador de Protocolos

Estudiante:

William Gerardo Alfaro Quiros – 2022437996

René Sánchez Torres – 2020051805

Jose Andrés Vargas Serrano – 2019211290

Jose Alexander Artavia Quesada – 2015098028

Profesor:

Ing. Juan Manuel Sánchez Corrales

Fecha de entrega:

6 de octubre, 2025

Semestre II

Introducción **3**
Diseño **4**
Protocolo Utopía 5
Protocolo Stop-and-Wait 6
Protocolo PAR 7
Protocolo Sliding Window de 1 Bit 8
Protocolo Go-Back-N 9
Protocolo Selective-Repeat 10
Análisis **11**
Conclusiones **13**
Bibliografía **14**

Introducción

Antes de profundizar en la capa de enlace y en los protocolos que se utilizan en esta, es importante mencionar algunos aspectos fundamentales en los cuales se basa el modelo de comunicación. La interacción entre dispositivos es un campo de estudio amplio y complejo, que puede resultar difícil de comprender únicamente desde la teoría. Por esta razón, es más sencillo desarrollar inicialmente modelos, diagramas y representaciones funcionales del sistema, con el propósito de comprender de manera más efectiva los retos que implica este proceso.

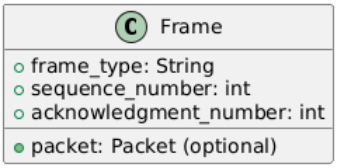
Inicialmente se asume que la capa física, la de enlace de datos y de red funcionan como procesos independientes que intercambian mensajes entre sí. En una implementación común, parte de las funciones de la capa de enlace y de la capa física se delegan al hardware especializado como una tarjeta de red, mientras que las demás funciones, junto con la capa de red, se ejecutan en la unidad central de procesamiento principal, siendo el enlace de datos gestionado generalmente por un controlador. Sin embargo, existen arquitecturas alternativas, como la ejecución total en hardware o en software. Considerar las tres capas como procesos separados facilita la comprensión del modelo y resalta la independencia de cada una (Tanenbaum & Wetherall, 2011).

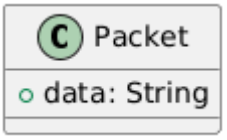
Desde el punto de vista de la capa de enlace, el paquete recibido desde la capa de red contiene únicamente datos, y cada bit debe entregarse íntegro a la capa de red del destino. La interpretación de este paquete, como la posible presencia de encabezados, no es responsabilidad de la capa de enlace. Cuando esta recibe un paquete, lo encapsula en un frame, le agrega un encabezado y un tráiler con la información de control. De esta manera, un frame está compuesto por el paquete original, los datos de control en el encabezado y un checksum en el tráiler. Luego de esto, el frame se envía al dispositivo receptor. Para simplificar el diseño de los protocolos, se asume la existencia de rutinas que permiten a la capa física enviar y recibir frames, encargándose éste de manera automática de calcular, añadir o verificar la checksum, usando por ejemplo el algoritmo CRC, sin que sea necesario incorporarlo de forma explícita en el protocolo (Tanenbaum & Wetherall, 2011).

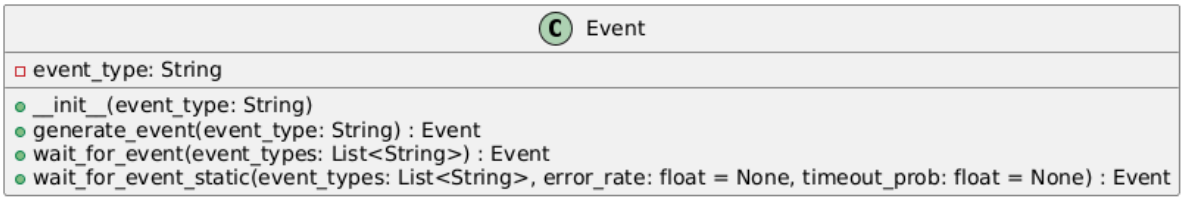
Para este proyecto se implementará un simulador con seis protocolos diferentes: tres enfocados a la comunicación en un solo sentido, los cuales son: Utopía, Stop-and-Wait y PAR, y otros tres dirigidos para la comunicación bidireccional, los cuales serían: Sliding Window de 1 bit, Go-Back-N y finalmente Selective-Repeat. Estos protocolos serán desarrollados mediante el lenguaje de programación Python versión 3.13.

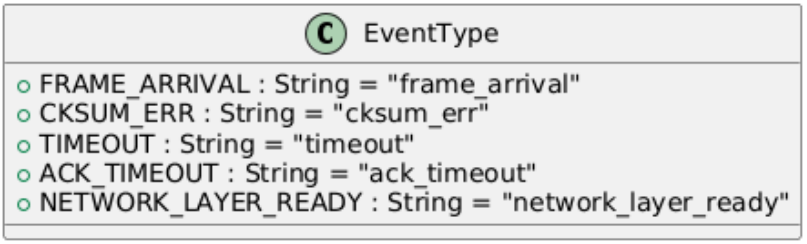
Diseño

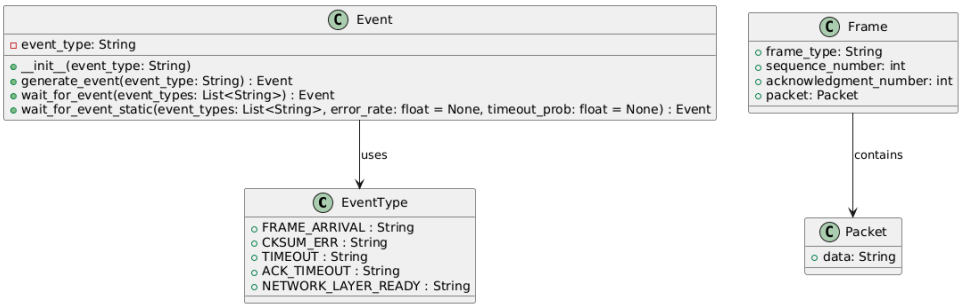
El objetivo fue construir un simulador de capa de enlace que permita comparar, bajo el mismo canal configurable, seis variantes: Utopía, Stop-and-Wait, PAR, Sliding Window de 1 bit, Go-Back-N (GBN) y Selective Repeat (SR). No se usaron sockets; el “medio” es una estructura en memoria. Los eventos de llegada, error y timeout se generan con probabilidades ajustables y se controlan desde la interfaz (error_rate, timeout_prob y step_delay). El modelo común de datos es simple: Frame (frame_type, sequence_number, acknowledgment_number y un Packet opcional con data) y Event/EventType (FRAME_ARRIVAL, CKSUM_ERR, TIMEOUT, ACK_TIMEOUT, NETWORK_LAYER_READY). Con esto se garantiza que todos los protocolos hablen el mismo “idioma” y que los experimentos sean comparables.





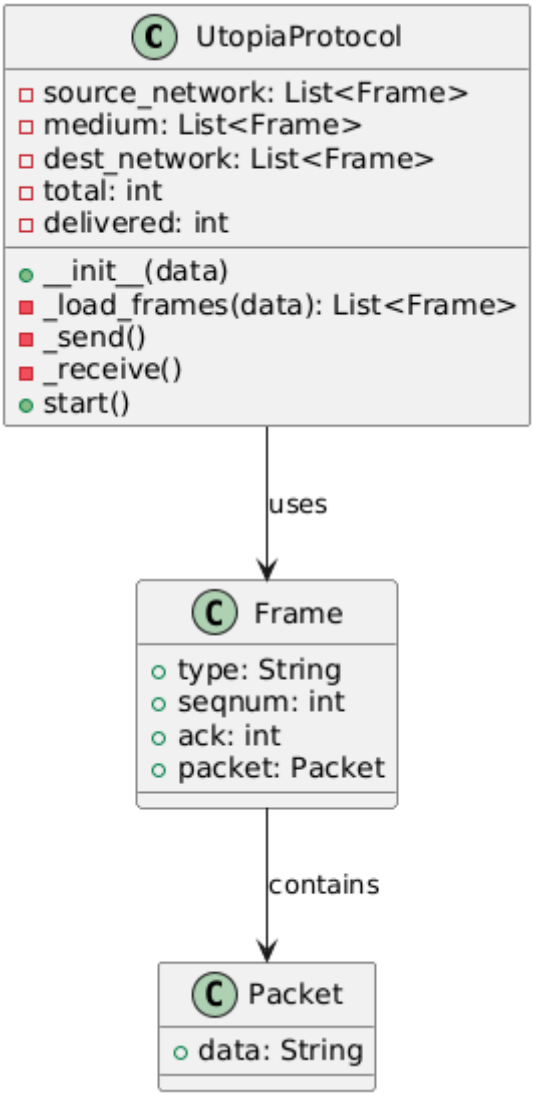






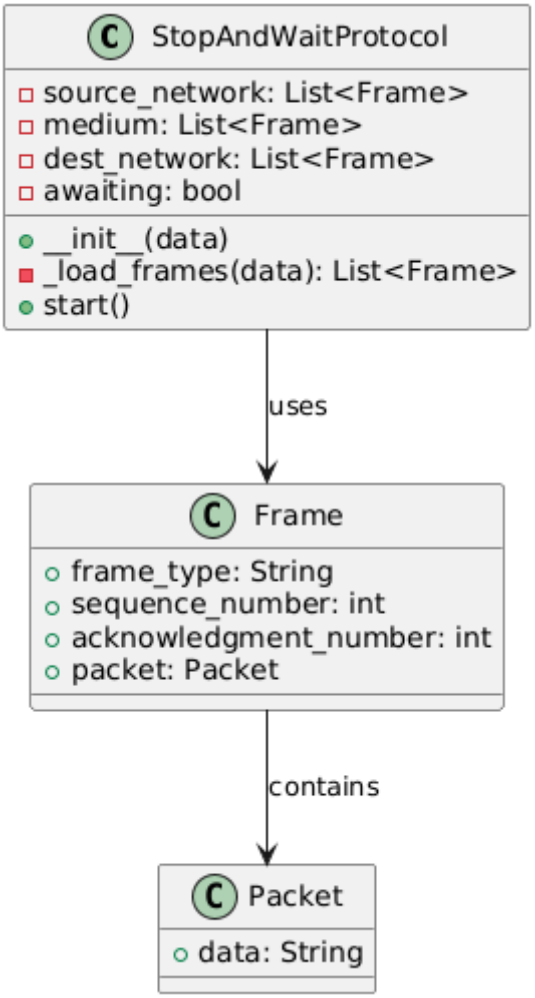
Protocolo Utopía

El diagrama muestra una ruta A -> medio -> B sin fallas. Las listas source_network, medium y dest_network ayudan a seguir el recorrido de cada frame. No hay ACK ni temporizadores; sirve como línea base para validar formato de tramas, conteo de entregas (total/delivered) y mensajes en consola. Impacto: verificación rápida de que la tubería A-medio-B y la GUI/CLI están bien conectadas.



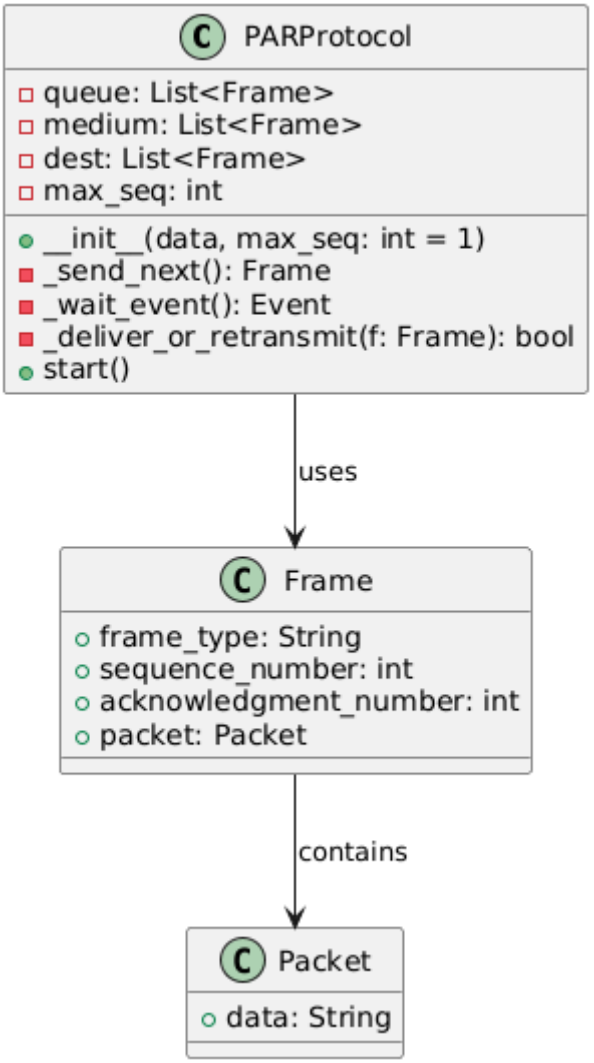
Protocolo Stop-and-Wait

El diagrama de StopAndWaitProtocol muestra tres listas para seguir cada trama: source_network, medium y dest_network. La bandera awaiting indica si hay un frame “en vuelo”. La secuencia típica es: cargar las tramas desde los datos (load_frames(data)), tomar la primera, enviarla al medio, poner awaiting en true y esperar el evento de llegada. No se usa ACK explícito; la entrega en dest_network se toma como confirmación y entonces awaiting vuelve a false para habilitar el siguiente envío. No hay ventana (tamaño 1 de facto) ni temporizadores; el supuesto es canal sin pérdida para esta variante. El impacto en el diseño fue clarificar el control mínimo de estado que necesita una transmisión confiable 1 a 1 y dejar una base directa para extender a PAR (donde se añaden pérdidas y reintentos).



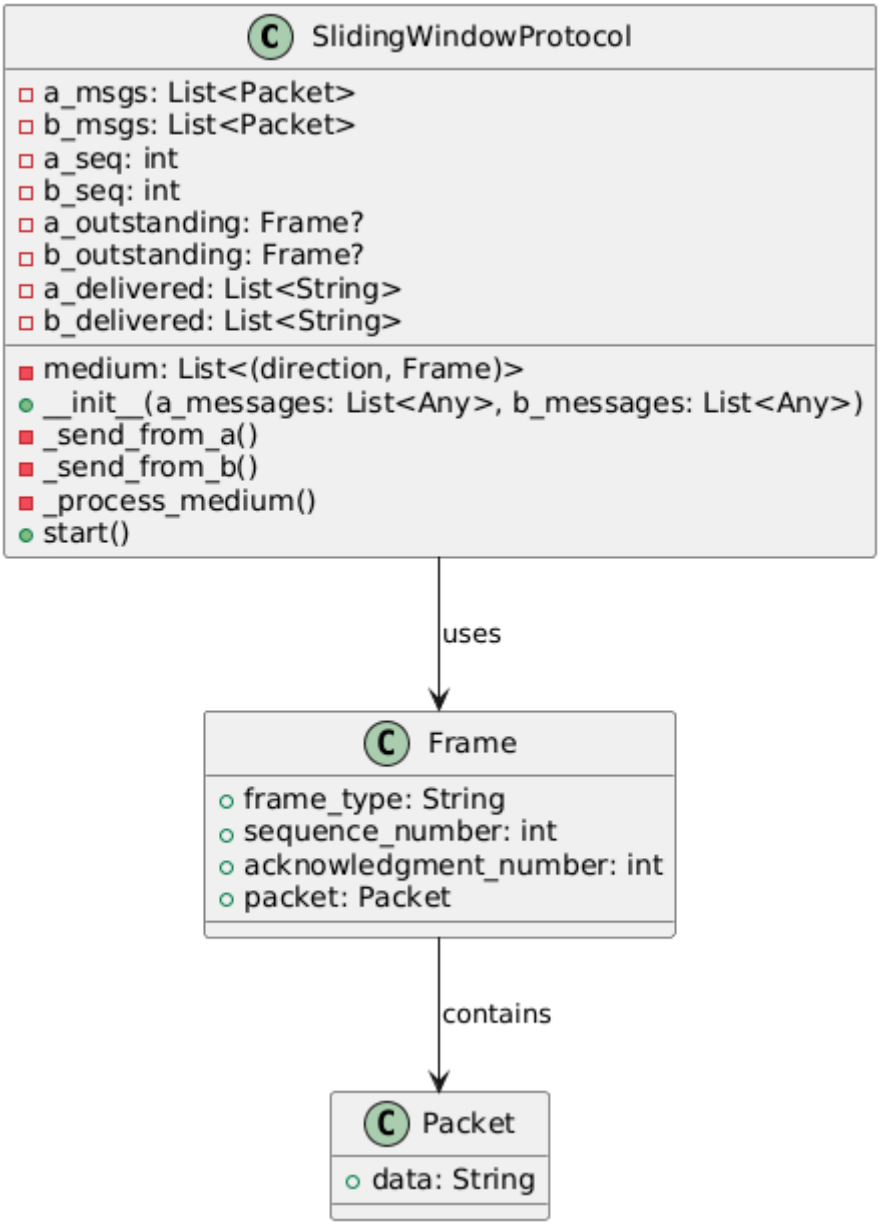
Protocolo PAR

El protocolo usa una cola (queue) y envía de a un frame. Si el evento reporta CKSUM_ERR o TIMEOUT, se retransmite la misma secuencia; si llega bien, se avanza. El campo max_seq permite limitar el rango de secuencias (por defecto 1, estilo alternante). Decisión de diseño: tratar el acuse como “llegó sin error”. Impacto: se observa con claridad cómo la tasa de pérdida dispara reintentos



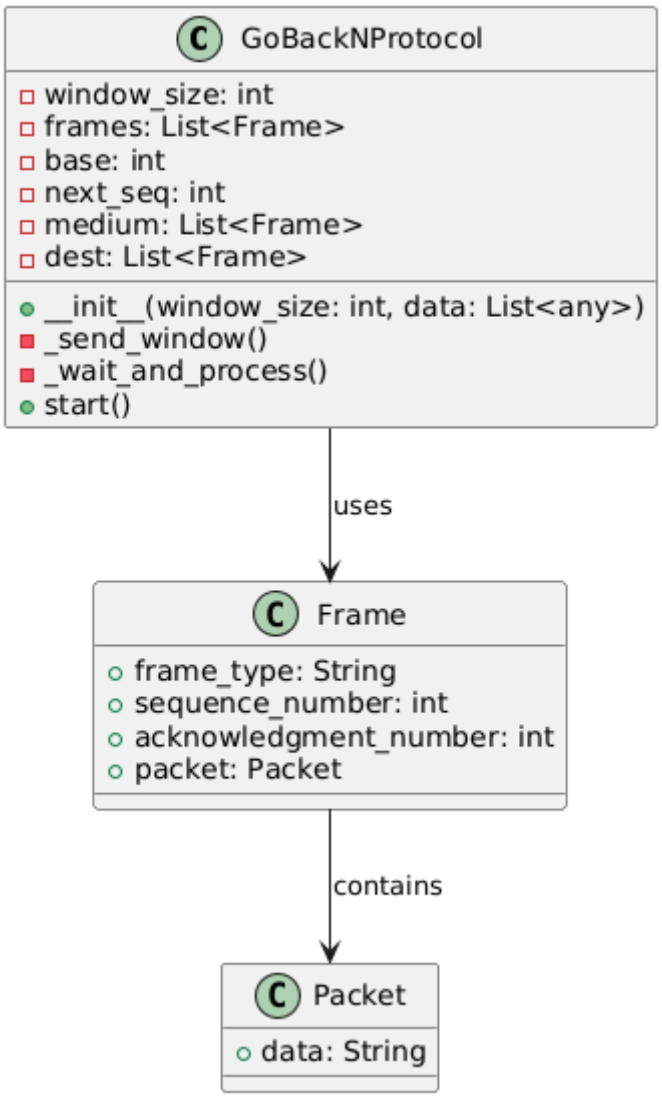
Protocolo Sliding Window de 1 Bit

El diagrama incluye a_msgs y b_msgs para simular tráfico en ambos sentidos al mismo tiempo. Hay un outstanding por lado y un número de secuencia por extremo (a_seq, b_seq). El ACK viaja por piggyback en acknowledgment_number. Supuestos: ventana de tamaño 1 y alternancia 0/1 (alternating bit). Impacto: mejor uso del canal que unidireccional, sin la complejidad de ventanas grandes



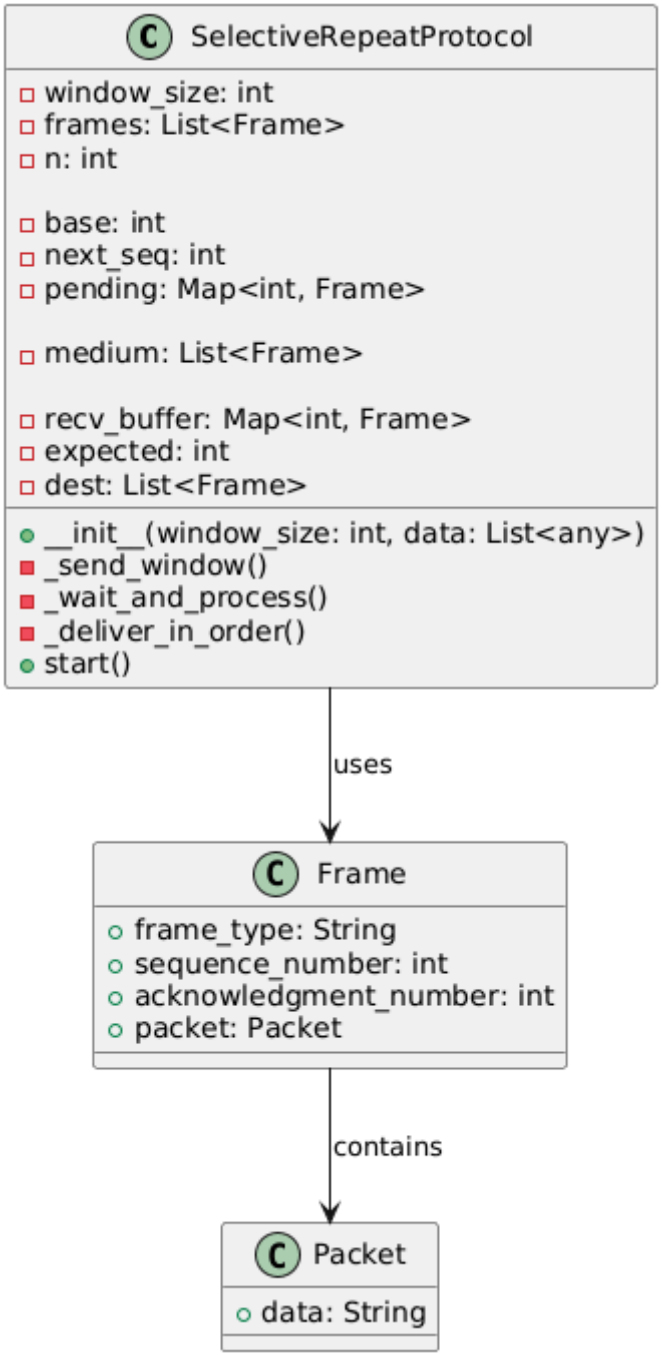
Protocolo Go-Back-N

Aparecen window_size, base y next_seq. Se envía mientras next_seq ≤ base + window_size − 1. Si ocurre error o timeout en cualquier trama de la ventana, se vuelve a base y se reenvía el bloque. El ACK es acumulativo: al confirmarse lo pendiente hasta cierto número, base avanza. Decisión de diseño: turnos alternos A->B y B->A para que el log sea legible en consola. Impacto: buen throughput con baja pérdida; costo alto cuando hay errores porque se repite en bloque.



Protocolo Selective-Repeat

Además de window_size, base y next_seq, el diagrama muestra pending (mapa de tramas en vuelo), recv_buffer (tramas fuera de orden) y expected (siguiente secuencia a entregar). Cada frame se confirma de forma individual; solo se retransmiten los que fallan. La función deliver_in_order recorre recv_buffer desde expected para liberar en orden. Supuestos: búfer suficiente del receptor y números de secuencia con tamaño adecuado al valor de w. Impacto: menor desperdicio que GBN bajo pérdida, con mayor uso de memoria y lógica de control.



Análisis

En Utopía no hay control adicional: se envía desde A al medio y del medio a B. Esta variante funcionó como línea base para validar el formato de la trama y la legibilidad de los mensajes. Stop-and-Wait añade el estado “frame en vuelo”; al llegar la trama, el medio queda libre y se habilita la siguiente. No se modeló ACK explícito; la llegada cuenta como confirmación. Su fortaleza es la simplicidad, a costa de latencia por el ciclo de ida y vuelta. PAR introduce pérdidas y timeouts simulados; ante CKSUM_ERR o TIMEOUT, se reintenta la misma trama con el mismo número de secuencia hasta lograr entrega. En la práctica equivale a Stop-and-Wait con reintento y permite observar cómo crecen las retransmisiones cuando aumenta la pérdida.

Con Sliding Window de 1 bit hay dos sentidos simultáneos. Se gestionan dos colas (A y B) y un pendiente por lado. El ACK viaja por piggyback usando el campo de acuse y se alterna el sequence_number entre 0 y 1. Se aprovecha mejor el canal que en unidireccional, aunque el tamaño de ventana igual a 1 limita el salto en rendimiento. En GBN se mantienen base y next por sentido y se envía mientras next no exceda base + w − 1. Si una trama de la ventana falla, se vuelve a base y se retransmite el bloque. Se usa ACK acumulativo para deslizar la ventana cuando lo previo quedó confirmado. Con baja pérdida, el throughput mejora; cuando la pérdida sube, aparece el costo de retransmisiones en masa.

SR conserva base/next y w, pero el receptor acepta y guarda tramas fuera de orden. Solo se retransmite lo que falló y la entrega al nivel superior se hace en orden usando expected_* y un búfer de recepción. En presencia de errores ofrece menor desperdicio que GBN, a cambio de mayor complejidad de control y memoria.

Los principales intercambios quedan así. En throughput frente a confiabilidad, Utopía y Stop-and-Wait aseguran entrega en este modelo, pero usan poco el canal. GBN aprovecha ACK acumulativos y sube el rendimiento, con el riesgo de repetir bloques enteros si hay errores. SR reduce ese desperdicio con ACK selectivos y búfer en el receptor. Cifras exactas por protocolo: [pendiente]. En complejidad y memoria, Stop-and-Wait y PAR son livianos; Sliding 1-bit añade bidireccionalidad y piggyback; GBN requiere manejo de ventana y bases; SR agrega mapas y búferes de recepción. Tabla comparativa: [pendiente]. En temporización, no hay timers por trama basados en reloj; el TIMEOUT es probabilístico y el paso se regula con sleep_step. Esto simplifica la implementación, pero impide medir RTT o caducidad determinista.

Como límites propios de la simulación, no hay sockets ni CRC real (CKSUM_ERR encapsula la corrupción). No se modelaron duplicados o reordenamientos en todas las variantes. Tampoco hay métricas automáticas de eficiencia o throughput; esos resultados quedan [pendiente]. Para ubicar la lógica en el repositorio: events.py concentra parámetros y eventos, Protocols/protocol_*.py contiene los métodos start de cada variante, y menu.py y gui.py manejan selección, tamaños de ventana, datos A/B y control de error_rate, timeout_prob y step_delay, además de pausa y stop.

Conclusiones

La separación entre medio/eventos y la lógica de cada protocolo permitió reutilizar un canal parametrizable y comparar estrategias de control de errores con salidas consistentes. La interfaz por consola facilitó variar ventana, pérdida y timeout, y pausar o detener sin tocar los estados internos. En GBN y SR se aprecia la diferencia entre ACK acumulativo y ACK selectivo en la cantidad de retransmisiones cuando aumenta la pérdida, aunque las métricas formales siguen.

Las mejoras prioritarias incluyen incorporar timers por trama reales, ACK/NAK explícitos en todas las variantes y un módulo de medición con throughput, eficiencia y tasa de retransmisión. También conviene habilitar una semilla fija para repetir escenarios exactamente y modelar latencia variable o colas limitadas para efectos de congestión. Metas numéricas y escenarios concretos.

Como aprendizajes, el paso de una ventana de 1 a w>1 cambió el rendimiento de forma clara, los ACK selectivos de SR redujeron retransmisiones y la gestión de temporizadores demostró ser clave para el control de errores. Mantener separados eventos y protocolos simplificó el razonamiento y la depuración.

Bibliografía

Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks* (5.ª ed.). Pearson