PAR – Unidad 7

Interconexión de redes: capa de red y capa de transporte:

CAPA DE TRANSPORTE

Capa de transporte

- Se suele establecer una distinción entre el bloque formado por las:
  - capas 1 a 4: prestador del servicio de transporte, independiente de las redes subyacentes (capas 1 a 3): la infraestructura de red
  - capas superiores: usuario del servicio de transporte o de la infraestructura de red
- El servicio ofrecido por esta capa es de extremo a extremo (*end-to-end*) de la comunicación, es decir, los protocolos de transporte se ejecutan en los sistemas finales, mientras los de red en los *routers* (*hop-by-hop*):
- Crea un mecanismo de intercambio de datos independiente de la diversidad de las redes a atravesar, una especie de *capa de enlace virtual entre extremos*, que puede ser:
  - no orientado a la conexión, como UDP
  - orientado a la conexión, como TCP

Capa de transporte - Cuestiones

- Direccionamiento (puertos, en TCP/UDP):
  - por uso o convención: números de puertos bien conocidos, asignados por ICANN (ver */etc/services* o *getent services x*)
  - servicio de nombres o directorio, donde se registran otros servicios y al que los clientes preguntan por estos (*portmap*)
- Multiplexado entre aplicaciones que comparten la misma capa de transporte
- Control de flujo y almacenamiento en búfer (caso del TCP):
  - recordar la capa de enlace de datos (unidad 04-2)
- Establecimiento y fin de conexión (caso del TCP):
  - se pierden, retrasan o duplican paquetes, y llegan con la conexión cerrada => necesita algún tipo de control:
    - acuerdo de tres vías (*3 way handshake*): SYN/SYN_ACK/ACK
    - se define un tiempo de vida máximo
    - liberación de recursos: simétrica o asimétrica

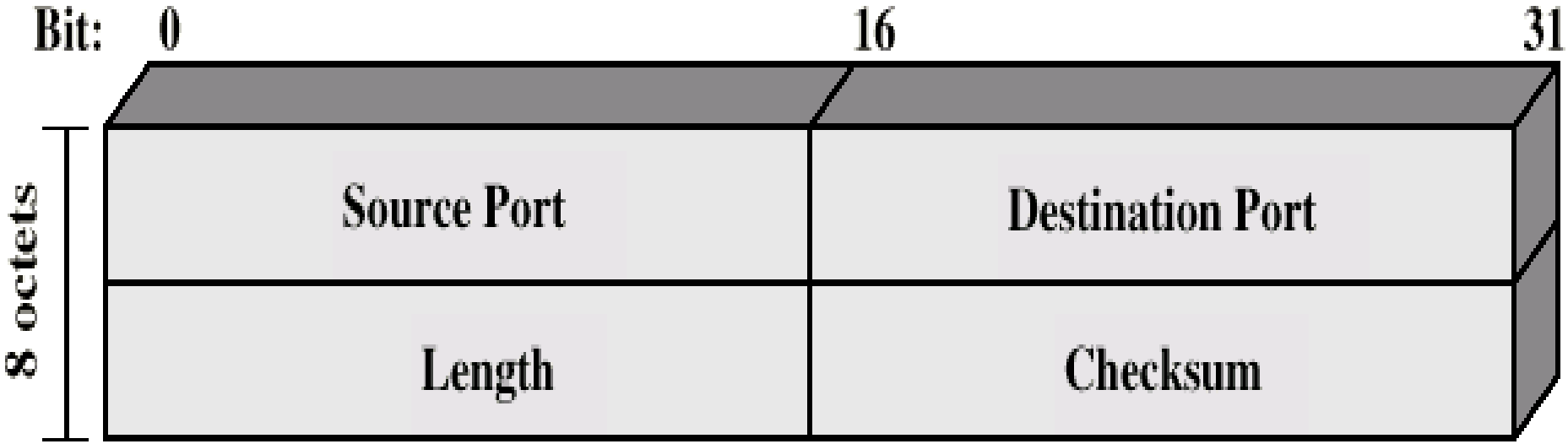
TCP y UDP

- ***Transmission Control Protocol*** [RFC 793, 1122, 1323,...]
  - los paquetes TCP se llaman **segmentos**
  - protocolo fiable, sobre otro no fiable (IP)
  - orientado a conexión, no permite la multidifusión
  - se basa en el intercambio de **flujo de bytes**
- ***User Datagram Protocol*** [RFC 768, 1122]
  - los paquetes UDP se llaman **datagramas**
  - protocolo no fiable, sobre otro no fiable (IP)
  - no orientado a conexión, permite la multidifusión
  - se base en el intercambio de **mensajes individuales**

UDP – Servicios y usos

- Ofrece un servicio no orientado a la conexión y no fiable a la capa de aplicación:
  - basado en mensajes (cada paquete es un mensaje)
  - control de entrega y de duplicados no garantizado
  - no hay establecimiento y cierre de la conexión
  - sobrecarga reducida
- Útil para aplicaciones que:
  - recolectan datos en tiempo real (p.e., SNMP)
  - difunden mensajes a grupos de usuarios (IGMP)
  - transmiten datos en tiempo real (p.e., VozIP)
  - hacen transacciones tipo petición-respuesta (p.e., DNS)
  - necesitan tener control preciso ellas mismas sobre el flujo de paquetes, errores o temporizadores

Cabecera UDP



- **puerto de destino** [16b]
- **puerto de origen** [16b]
- **longitud** del encabezado + datos [16b]
- **suma de verificación** [16b] opcional

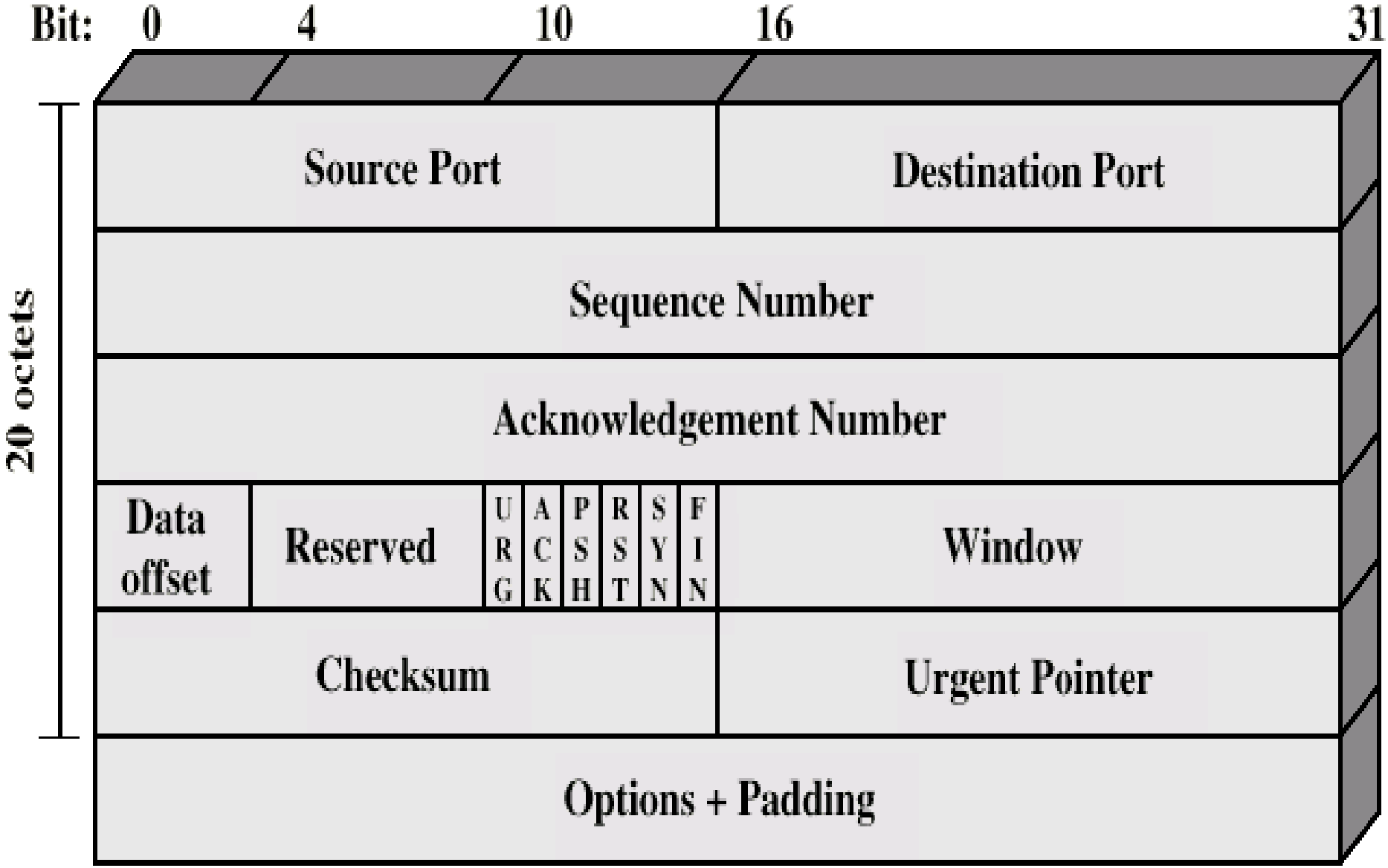
Aplicaciones que usan UDP

- echo: 7
- daytime: 13
- quote: 17
- dns: 53
- bootps/bootpc: 67/68
- tftp: 69
- sunrpc o portmapper: 111
- snmp: 161-2
- netbios-*: 137-9
- ... y muchos más:
  - grep udp /etc/services|less
  - getent services *nombre-del-servicio*

TCP - Servicios

- Ofrece la transmisión fiable de un flujo de bytes entre procesos (usuarios TCP) en nodos extremos a través de una de distintas redes
- Tiene tres modos de funcionamiento:
  - flujo normal de datos
  - flujo de datos forzado (*push*):
    - una aplicación solicita a la pila TCP del otro extremo que envíe los datos a la aplicación receptora de manera inmediata sin almacenarlos en su búfer (**bandera *PSH***)
    - así, p. e., se facilita la comunicación entre aplicaciones en tiempo real o un proceso que ya no tiene más datos que enviar fuerza que el otro extremo procese ya los datos
  - señalización de datos urgentes (*urgent*): *out of band*
    - la aplicación de origen solicita a su servicio TCP que envíe ciertos bytes como urgentes para que la aplicación de destino los trate con prioridad (**bandera *URG***)
    - el receptor decide la manera exacta de manejarlos

Cabecera TCP - Esquema



Cabecera TCP - Campos

- **Puerto de destino** [16b] y **de origen** [16b]
- **Número de secuencia** [32b] del primer octeto de datos
  - si es el primero -SYN-, indica el nº de secuencia inicial (ISN), siendo el primer octeto de datos ISN + 1
- **Número de confirmación** [32b]
  - Contiene el nº de secuencia del siguiente octeto que la entidad TCP espera recibir
- **Posición de datos o longitud de cabecera** [4b], (x32b)
- **Reservado** [3b]
- **Indicadores** [9b]: NS,ECW,ECE|URG,ACK,PSH,RST,SYN,FIN
- **Ventana** [16b] en bytes que se está dispuesto a aceptar
- ***Checksum*** [16b] C^1 ($\sum^{c1}$(datos, cab. TCP y pseudocab. IP))
- **Puntero** [16b] señala al último byte de datos urgentes

Cabecera TCP - Opciones

- Dos formatos de opciones (RFC 793 y 1323):
  - tipo de opción [1B]
  - tipo [1B] + longitud [1B] + valor [(longitud – 2)B]
- Tipos de opciones:
  - 0 : fin de lista | 1 : sin operación (NOP)
  - 2 [4B]: *maximum segment size* (MSS) es el max. nº de bytes de datos que puede recibir en un segmento. Se negocia en SYN/SYN-ACK. Suele estar determinada por el MTU de las capas inferiores (RFC 879)(MSS = MTU – cab.IP – cab.TCP; por defecto, 576 – 20 – 20 B)
  - 3 [3B]: ventanas mayores de 64KB (RFC 1323)
  - 4 [2B]: *Selective*-ACK permitido [en SYN] (RFC 2018)
  - 5 [XB]: SACK de los datos recibidos (RFC 2018)
  - 6/7 [6B]: *echo/reply* [sustituido por el tipo 8]
  - 8 [10B]: *timestamp* (RFC 1323) para calcular el RTT
  - ....

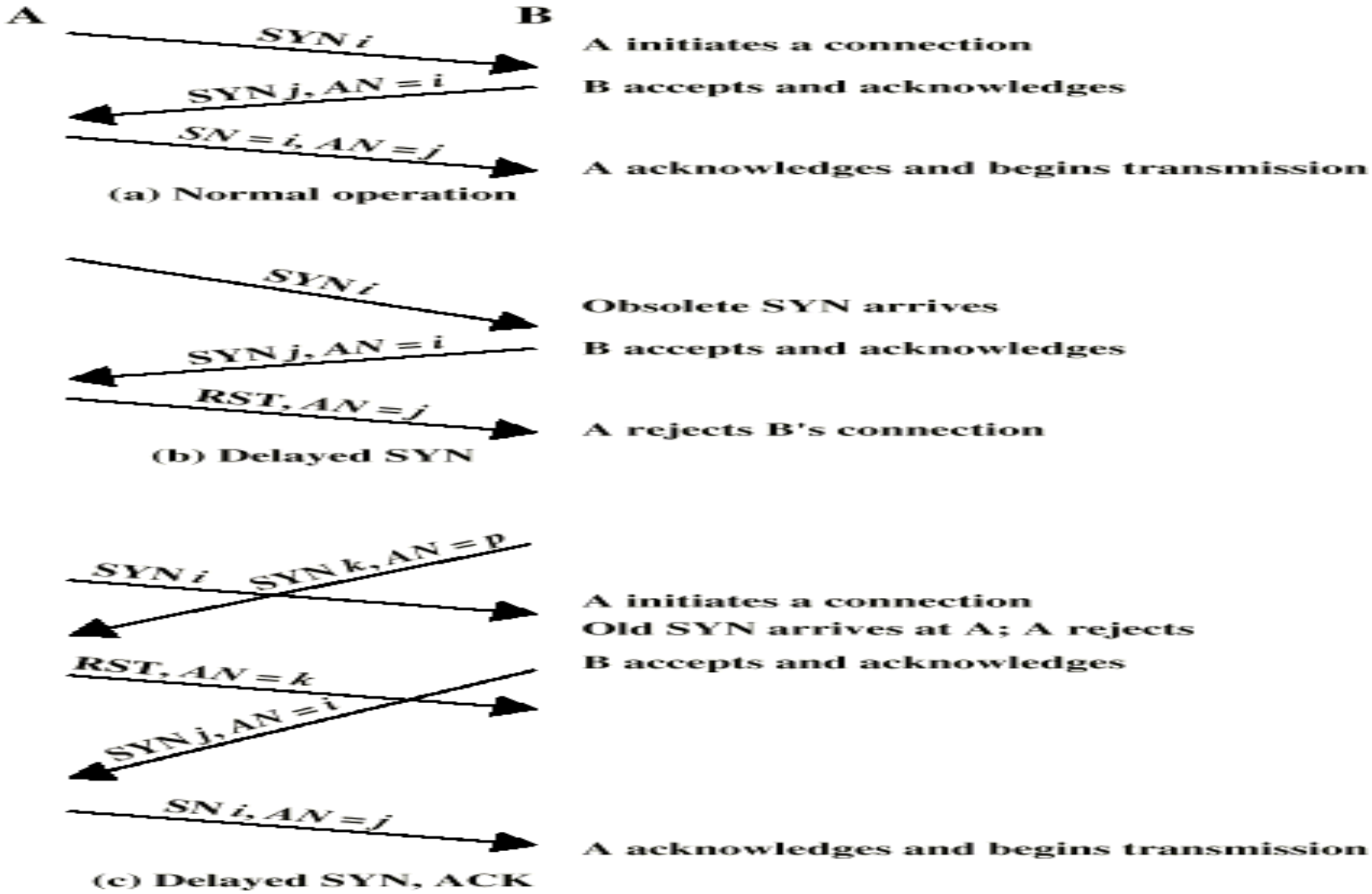
Mecanismo TCP

- **Establecimiento de la conexión** (independiente para cada sentido)
  - típicamente, saludo en tres pasos (SYN/SYN+ACK/ACK)
  - entre un par de ***sockets*** (dirección IP + puerto)
  - la conexión queda identificada por ese par de *sockets*
- **Transferencia de datos**
  - **flujo lógico de bytes** numerados en módulo 2^{32}
  - control de flujo por asignación de crédito de bytes
  - datos se van almacenando (búferes) en ambos extremos
- **Fin de la conexión** (independiente para cada sentido)
  - cierre ordenado: cada una de las partes envía FIN y espera su confirmación
  - cierre abrupto: se transmite un segmento RST, sin esperar confirmación

Establecimiento de la conexión

- Si no hay ningún proceso escuchando en el *socket* de destino, se rechaza el intento de conexión con RST
- Segmentos de conexiones anteriores perdidos o retrasados pueden causar problemas:
  - uso del campo SYN para sincronizar los números de secuencias iniciales (ISN)
  - necesidad de que los ACK incluyan número de secuencia
- ***Three way handshake*** (apretón de manos -o acuerdo- a tres vías):
  - cada flujo de datos en una conexión se controla de forma independiente, a partir de su propio ISN aleatorio (!=0).
  - el nodo que inicia la conexión (activo/cliente) envía un segmento *SYN ISN*
  - el que lo recibe (pasivo/servidor) confirma con *ACK ISN+1* y envía su *SYN ISN*
  - el nodo activo confirma el ISN del pasivo con *ACK ISN+1*

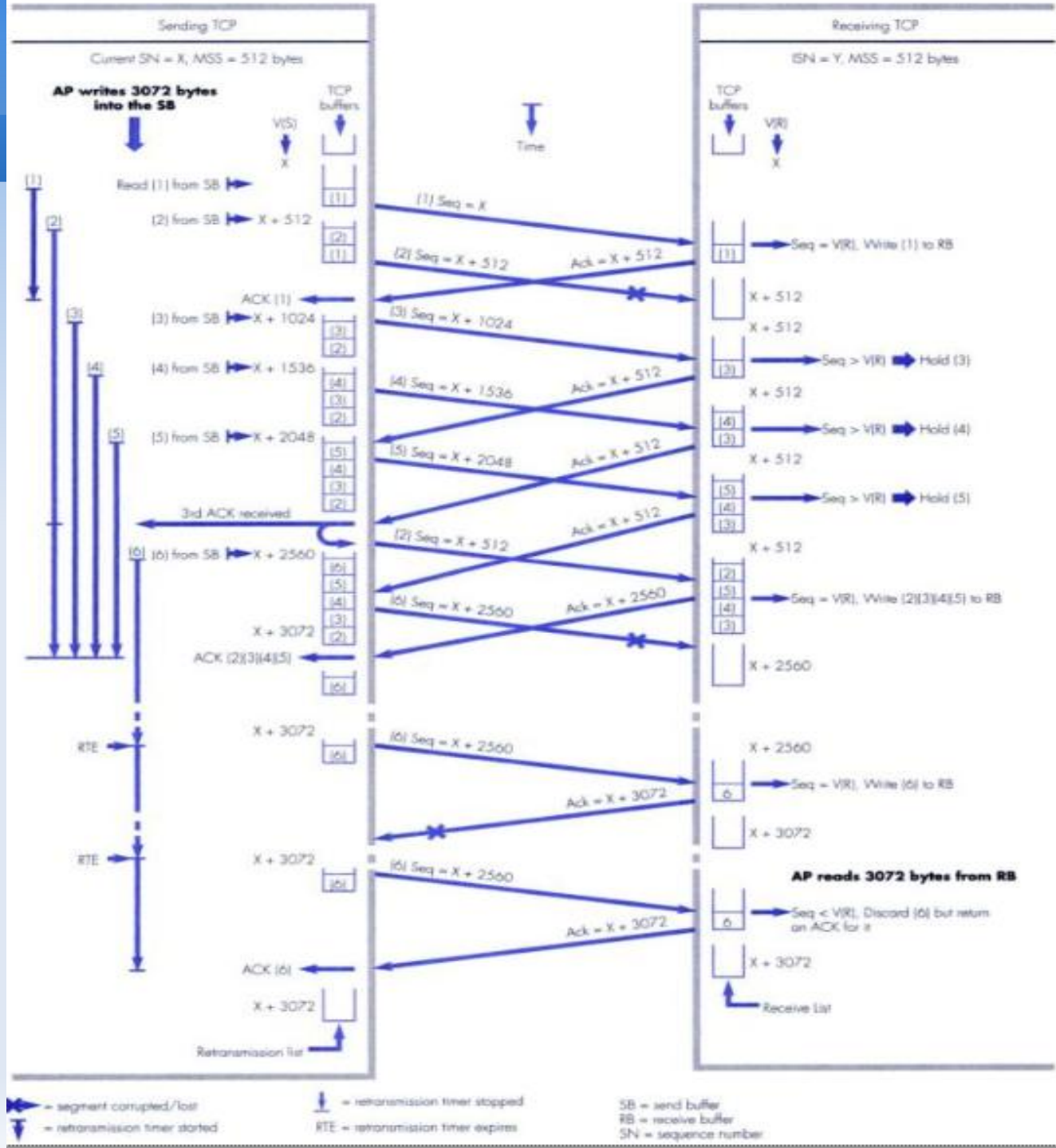
Three way handshake



Control de errores

- El **receptor** almacena los bytes (quizá en desorden) y envía *ACK X*, indicando el **nº de secuencia** del byte que espera a continuación y confirmando todos los bytes anteriores a X
  - si recibe un byte posterior a X sin haber recibido el byte X (desorden), lo almacena y envía un *ACK X* indicando que aún le falta dicho byte (tiene un "hueco")
  - cuando recibe el byte X, como se puede confirmar varios bytes a la vez, envía *ACK X+N* confirmando los N bytes que ha ido recibiendo en desorden (ya no hay "huecos").
- Los segmentos (grupo de bytes) pueden retrasarse, perderse o dañarse, el **emisor** actúa en función de las *ACK* que recibe, usando **temporizadores**:
  - la retransmisión puede no hacerse tras recibir un *ACK X* fuera de secuencia, sino tras varios *ACK X* (retransmisión rápida)
  - por cada segmento que se envía se pone en marcha un temporizador. Si expira sin recibir ACK X, se retransmite X
  - Se suele usar mucho la opción SACK (Selective ACK)

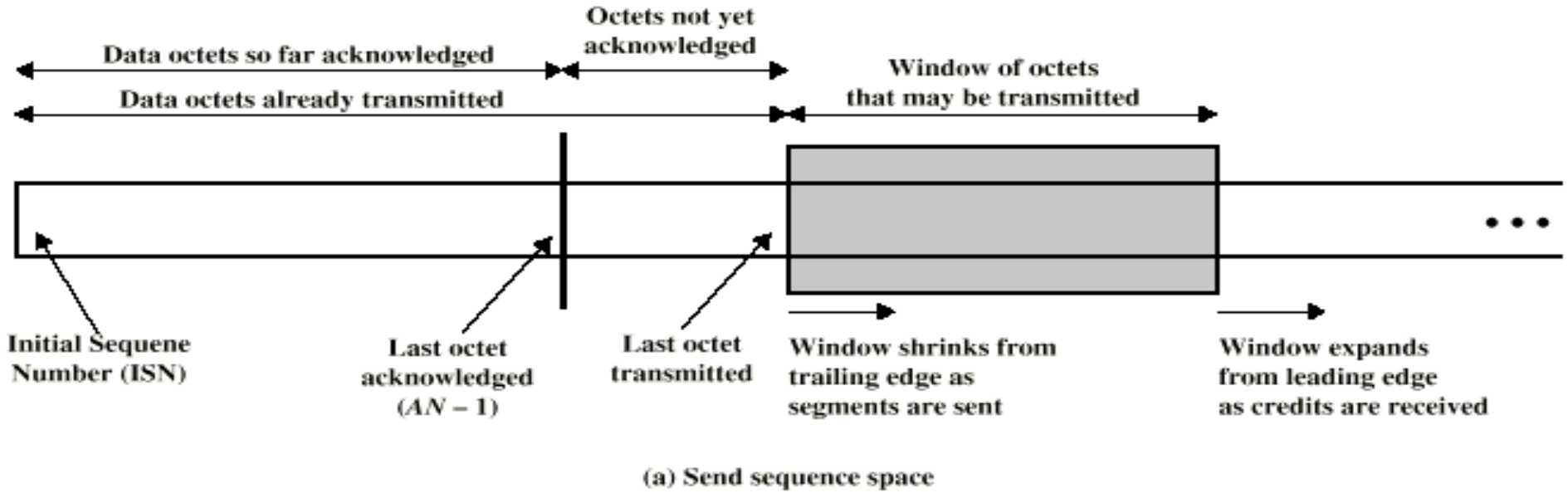
Control de errores



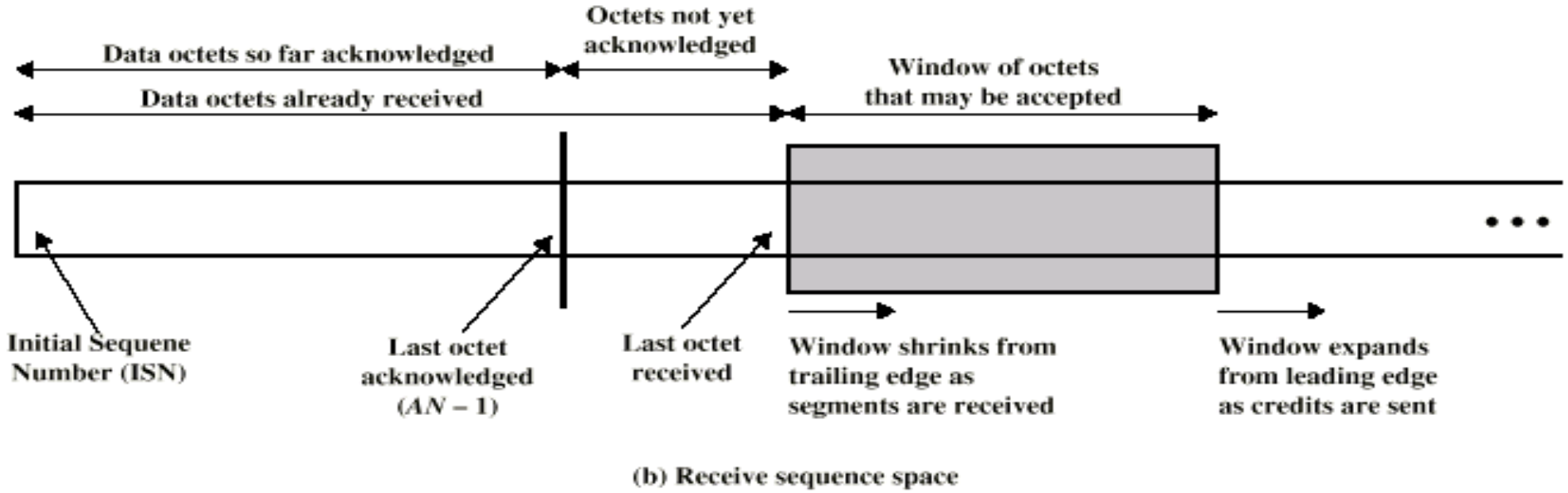
Control de flujo

- Se gestiona mediante el campo **tamaño de ventana**, que indica en cada segmento al otro extremo la cantidad de bytes que le es posible admitir a partir del nº de secuencia de *ACK* (según el espacio libre disponible en su búfer):
  - *ACK X*=i y *W*=j, espera el iº byte y concede un crédito al otro extremo para transmitir j bytes
  - según se van recibiendo datos se va cerrando la ventana
  - según se van confirmando y pasando datos a la capa superior se va abriendo la ventana
  - también depende de los recursos disponibles en el nodo (p. ej., el número de conexiones TCP simultáneas)
- Cuando un extremo envía *ACK X*=i y *W*=0, indica que su ventana está cerrada
  - para reabrirla, envía *ACK X*=i y *W*=j
  - pero por si éste se pierde, se produciría un interbloqueo => hay que usar también un temporizador

Perspectivas de emisor/receptor

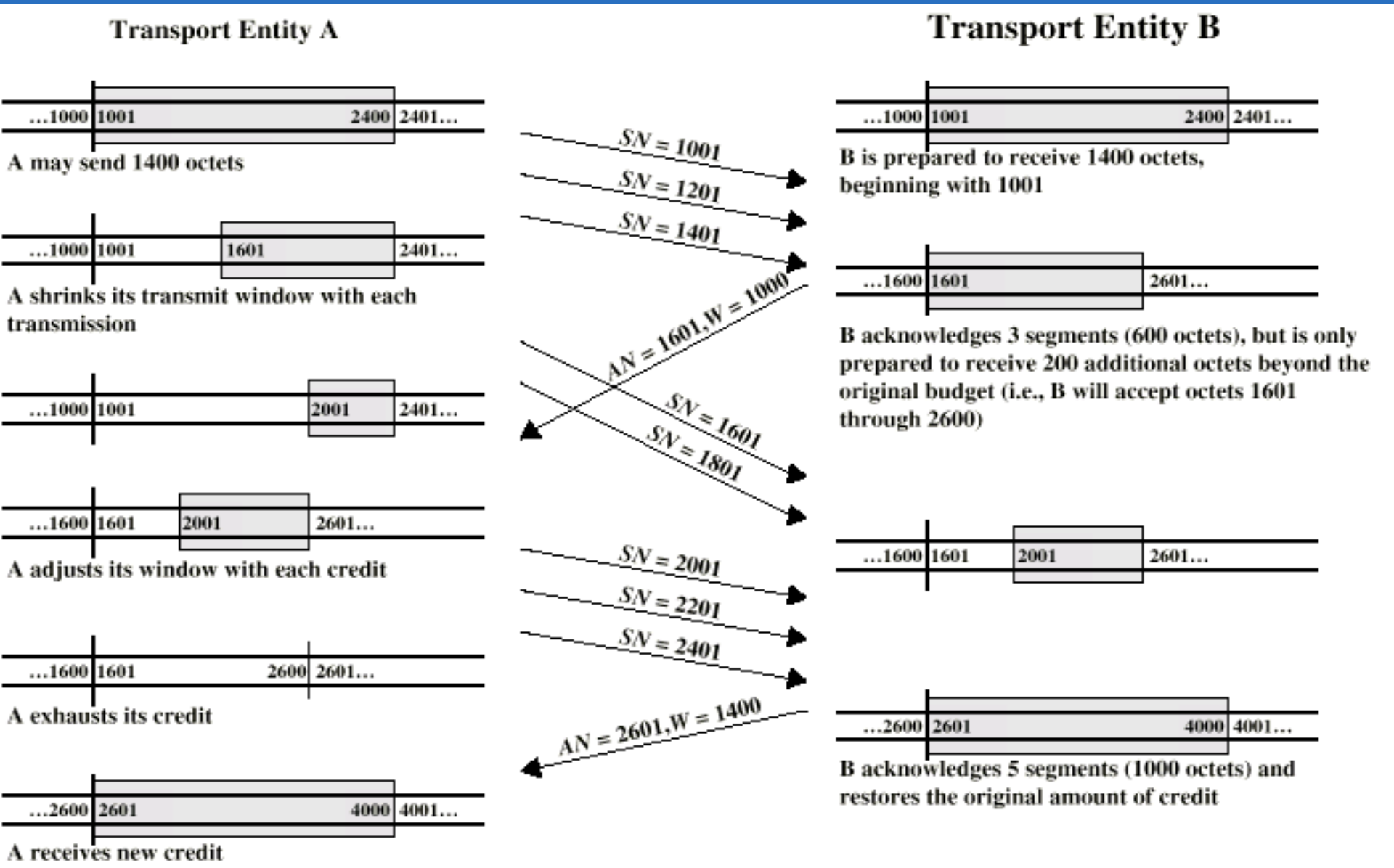


(a) Send sequence space



(b) Receive sequence space

Asignación de crédito



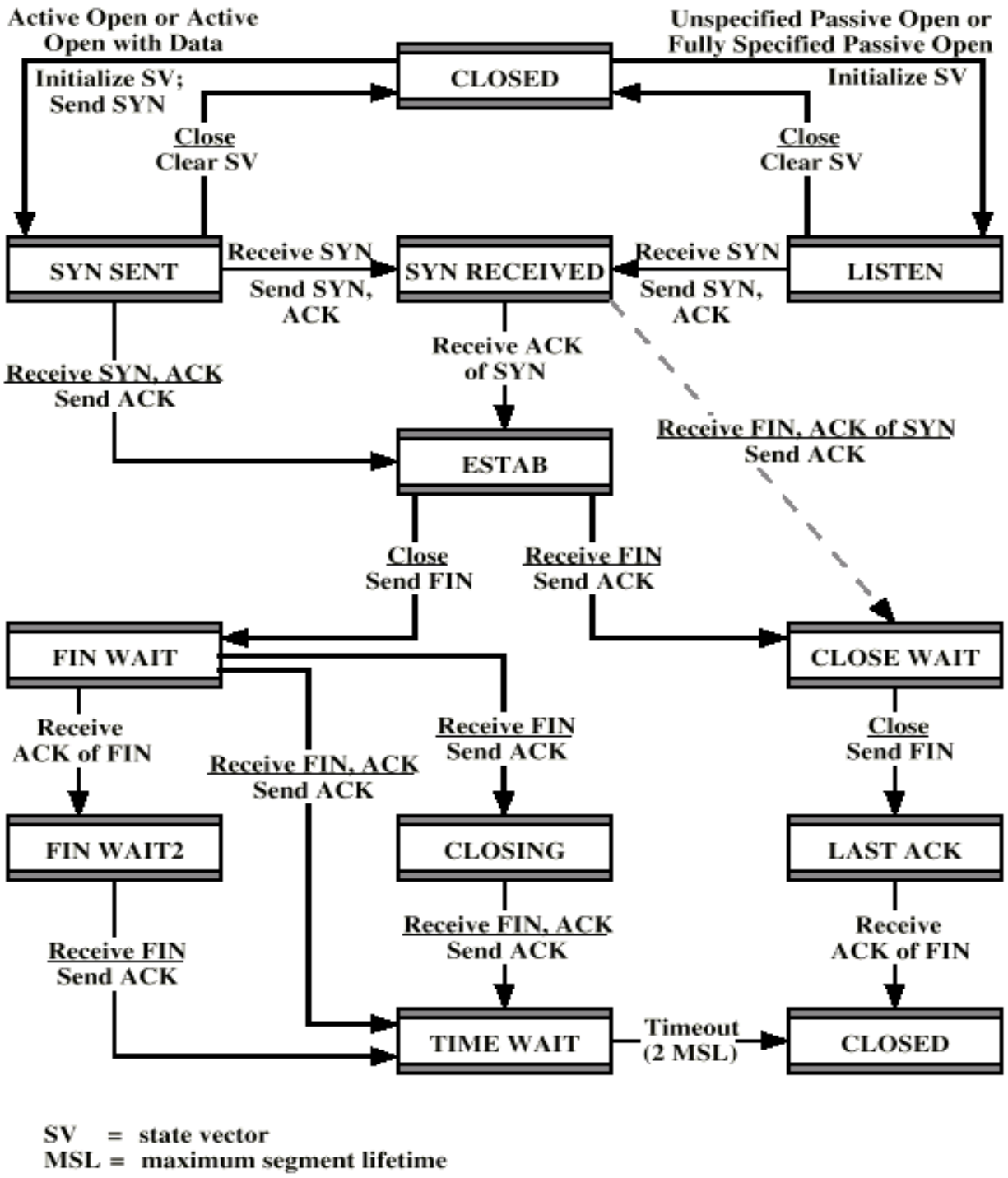
Detección de duplicados

- Si se pierde la ACK de uno o más segmentos, el segmento original se retransmite
  - el receptor tendrá bytes duplicados y puede reconocerlo gracias al número de secuencia
- Si se recibe el duplicado antes de cerrar la conexión
  - el receptor asume ACK perdido y confirma el duplicado
  - el emisor no debe confundirse con múltiples ACKs
  - el espacio de números de secuencia (2^{32}) se supone suficiente para que no se agote dentro del máximo tiempo de vida de un segmento
- Para evitar problemas con los duplicados recibidos después de cerrar la conexión
  - se espera un tiempo antes de comenzar una nueva conexión y se hace con un número de secuencia aleatorio, para evitar coincidencia con las anteriores

Cierre de la conexión

- Para evitar pérdida de bytes retrasados se asocia un número de secuencia a cada FIN
  - el receptor espera a todos los bytes anteriores al número de secuencia de FIN
- La posible pérdida de bytes y presencia de obsoletos obliga a cada extremo a confirmar el FIN del otro
  - se usa un ACK con el nº de secuencia del FIN a confirmar:
    - FIN WAIT ---->[1] FIN i ACK i+1 [2]<------ CLOSE WAIT
    - FIN WAIT2--->[4] ACK j+1 FIN j [3]<------ LAST ACK
    - TIME WAIT..... se espera un intervalo de tiempo igual a dos veces el tiempo máximo de vida de un segmento, durante el que no se pueden utilizar los recursos ocupados (*sockets*)

Diagrama de estado



Recuperación de interrupciones

- Después de un reinicio toda la información de estado se pierde
- La conexión está medio abierta
  - El lado que no falló aún cree que está conectado
- Cierre de conexión usando un temporizador
  - espera un ACK durante (*time-out*) * (máximo de reintentos)
  - cuando expira, cierra la conexión e informa al usuario
- Si la entidad que falla se reinicia rápidamente, puede enviar un RST i por cada segmento i que reciba y provocando un cierre anormal más rápido
- La decisión de reabrir la conexión se deja al usuario

Control de congestión

- El mecanismo de control de flujo basado en créditos se diseñó para evitar la saturación del destino
  - pero, ¿Qué pasa si es la propia red la que está congestionada?
  - aumentan los tiempos de transmisión y los descartes
- Para controlar esto se regulan:
  - los temporizadores de retransmisión
  - la ventana de congestión
- Gestión de **temporizadores de retransmisión:**
  - un **RTO** (*Retransmission TimeOut*) ocurre cuando un ACK no llega a tiempo (o nunca)
  - se estima el retardo de ida y vuelta **RTT** (*Round-Trip Time*) mediante la marcas de tiempo incluidas en los segmentos y se elabora un patrón
  - establece un tiempo algo mayor que el estimado: promedio simple o exponencial (+ reciente + peso) o varianza del RTT (alg. de Jacobson) o decaimiento RTO o algoritmo de Karn

Ventana de congestión

- Concepto que se añade a la ventana de emisión y recepción, de forma que se elige la más pequeña para decidir cuánto se puede transmitir en cada momento
- Gestión de la **ventana de congestión** (W) [RFC 2001]
  - **arranque lento:** empieza con W de tamaño 1 MSS y por cada ACK recibido puede enviar 1 MSS más: exponencialmente (1, 2, 4, 8, 16, 32 ... * MSS), que se detiene si se produce algún error o se llega a un umbral (SST)
  - **evitación de congestión:** a partir de ese umbral crece linealmente hasta alcanzar un 2º umbral a partir del cual se mantiene constante.
  - **recuperación rápida:** si se recibe 3 ACKs duplicados (=> por retraso => congestión leve), el valor de la ventana se divide por la mitad y se vuelve a la evitación de congestión
  - si ocurre RTO (=> por pérdida => congestión grave), se vuelve a una ventana de un segmento y al arranque lento

Ventana de congestión - Gráfica

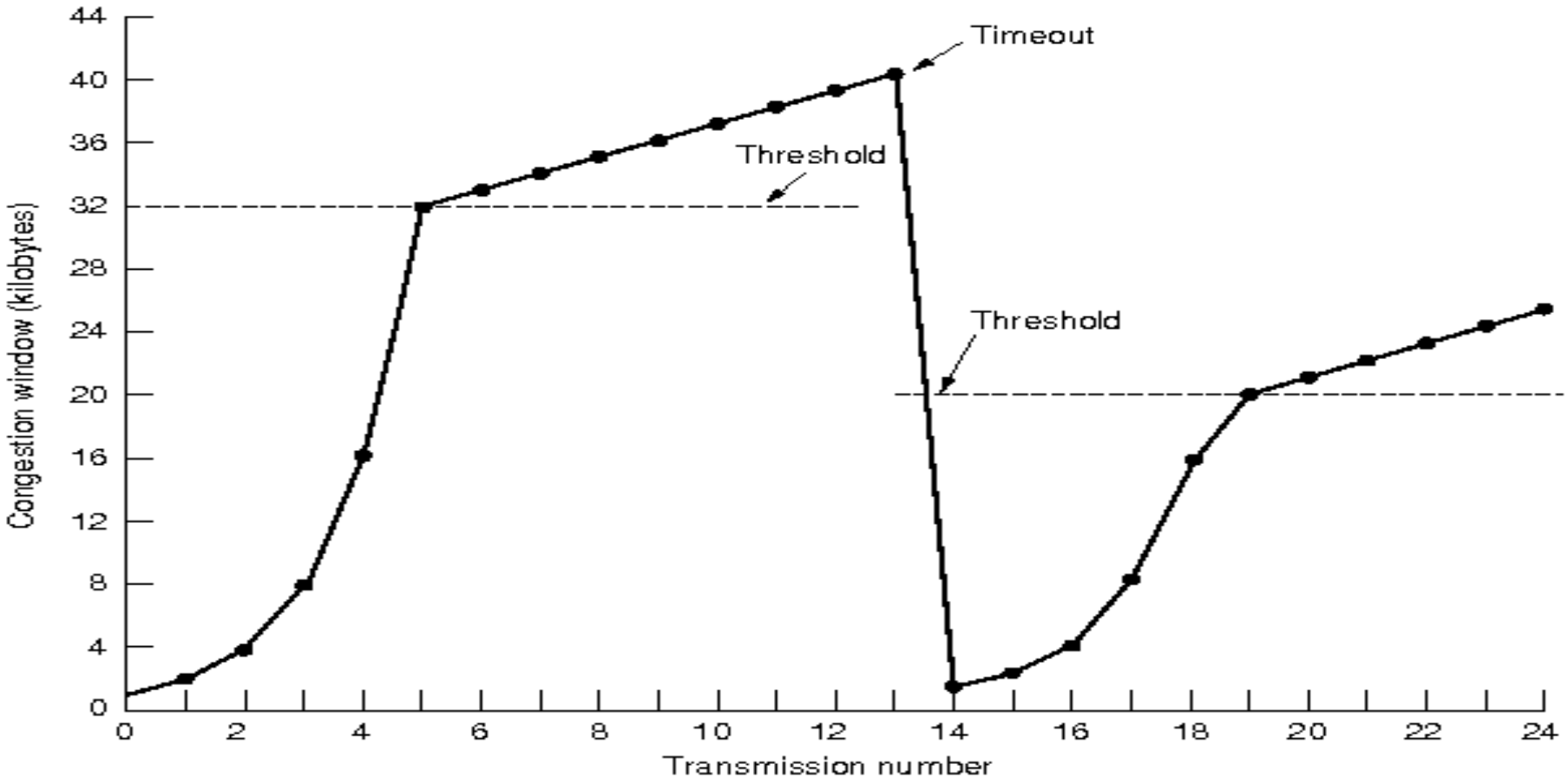


Fig. 6-37. An example of the Internet congestion algorithm.