# capa de transporte

gabriel infante-lopez

Walk on by
Walk on through
Walk 'til you run

"The Unforgettable Fire" - U2

# Elementos de la capa de transporte

- Direccionamiento
- Establecimiento de una conexión
- Liberación de una conexión
- Control de Flujo
- Multiplexión
- Recuperación de caídas.

# Establecimiento de conexión

**Idea Básica:** Para establecer una conexión, enviamos un "connection request". El destino acepta, y envía un ACK

**Problema:** Supongamos que no obtenemos respuesta, por una de las siguientes causas:

- El primer pedido no llego a destino.
- El ACK no llego de vuelta, estamos por establecer una segunda conexión, esto puede ser detectado.
- El primer pedido no llego *todavía*: ahora estamos por hacer una nueva conexión pero nadie sabe si es a propósito.

**Razón:** La red posee capacidad de almacenamiento.

# Atacando Duplicados

Solución: Restringir la vida útil de los TPDUs - si sabemos el tiempo de vida, sabemos que los paquetes viejos serán descartados

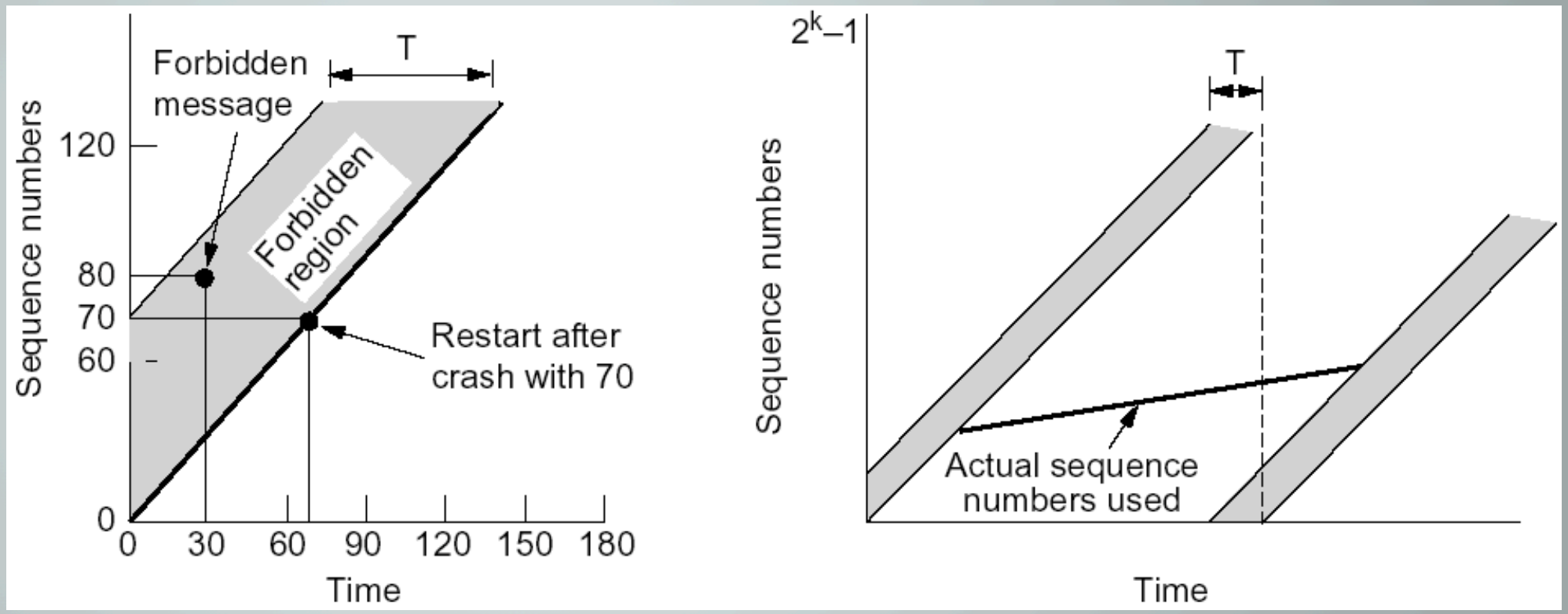
Idea Básica: Asignar números de secuencias a TPDUs de tal manera que no existan dos TPDUs con el mismo número

Problema: Si un host cae, debe empezar su numeración de nuevo? de donde empieza?

- No podemos esperar el tiempo de vida máximo de un paquete para volver a empezar. Este tiempo puede ser demasiado alto.
- Debemos evitar el primer segmento de numero de secuencias asignados. Entonces, solo debemos encontrar el numero inicial indicado.

# Atacando Duplicados

Solución: Asignar números de secuencia utilizando un reloj. Asumir que el reloj sigue funcionando aún durante caídas.




Cuidado: Asignarlos demasiado rápido y demasiado despacio, nos puede hacer entrar en la región prohibida.

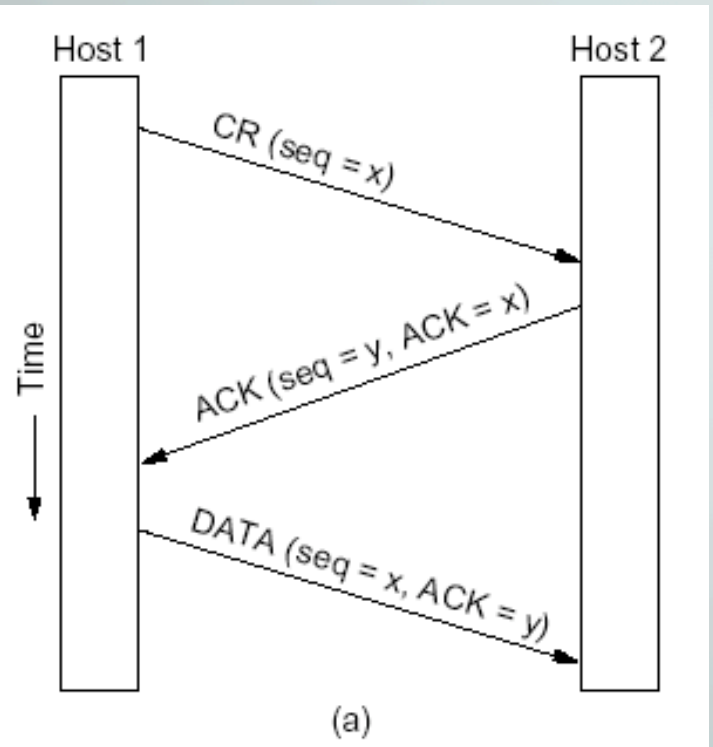
# Estableciendo conexión sin errores

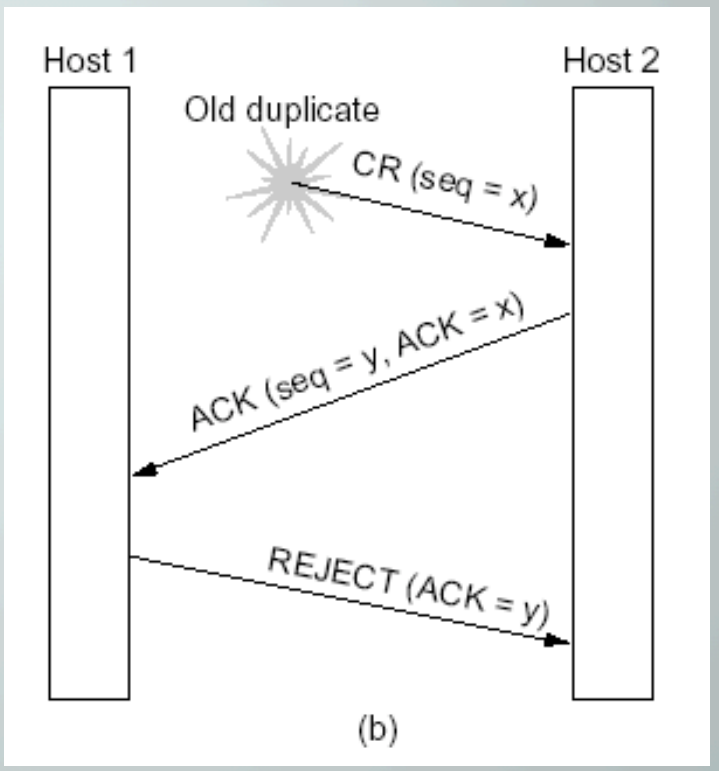
Problema: Bien, tenemos una manera de evitar duplicados, pero como hacemos para obtener una conexión primero?

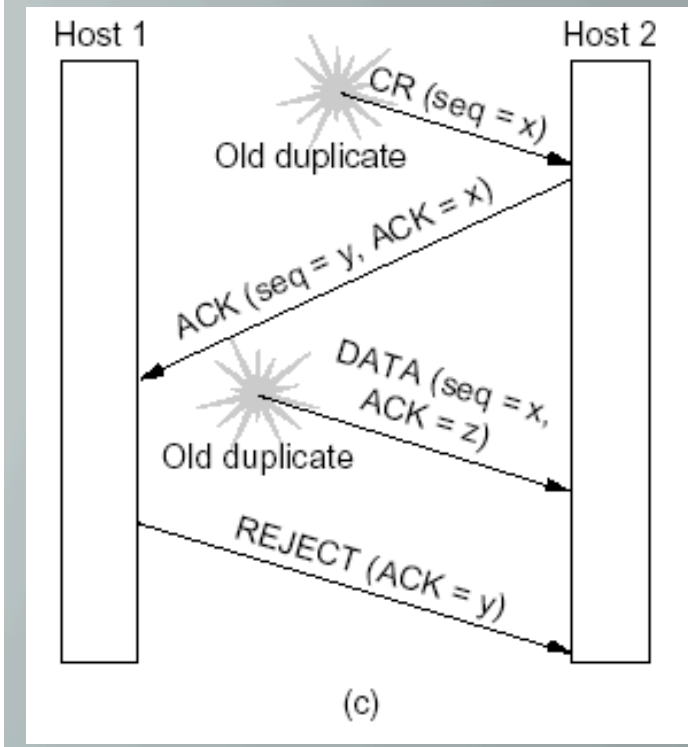
Nota: De una manera u otra tenemos que hacer que el transmisor y el receptor acuerden en los números iniciales. Tenemos que evitar que un número viejo de secuencia aparezca.

# Estableciendo conexión sin errores

**Solución**: Three-way handshake



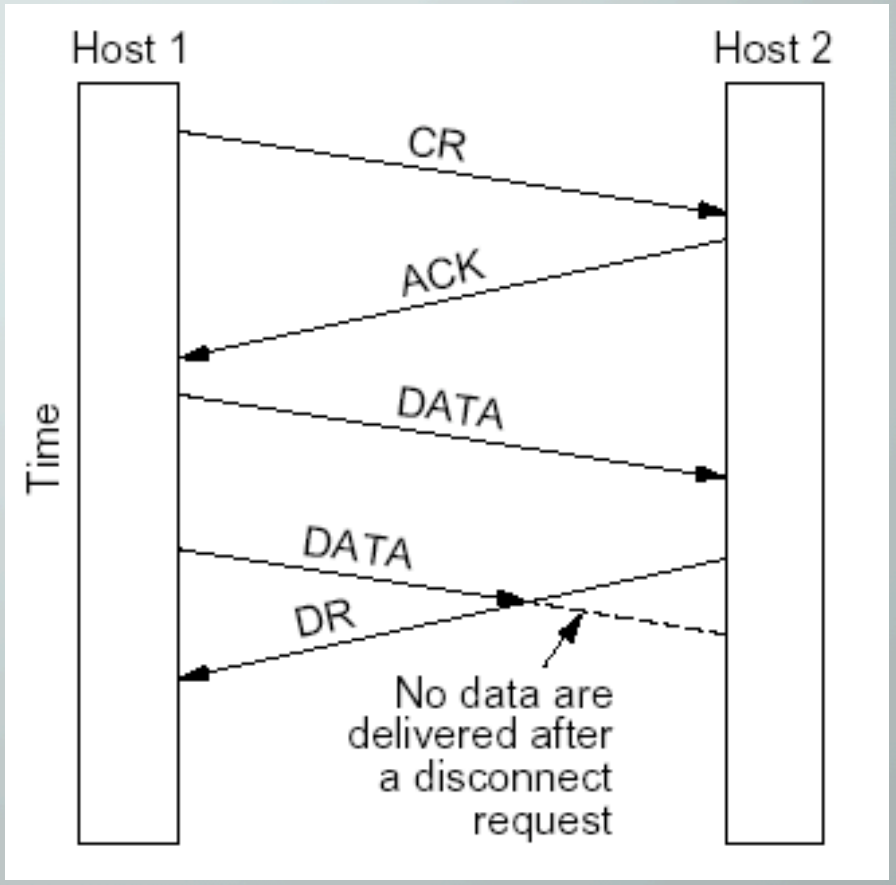
# Liberación de Conexión

## tipos

- Simetrico:
- Asimetrico:

# Liberación de Conexión
## Asimétrico

Una de las partes corta abruptamente la comunicación.
Puede haber perdida de datos.



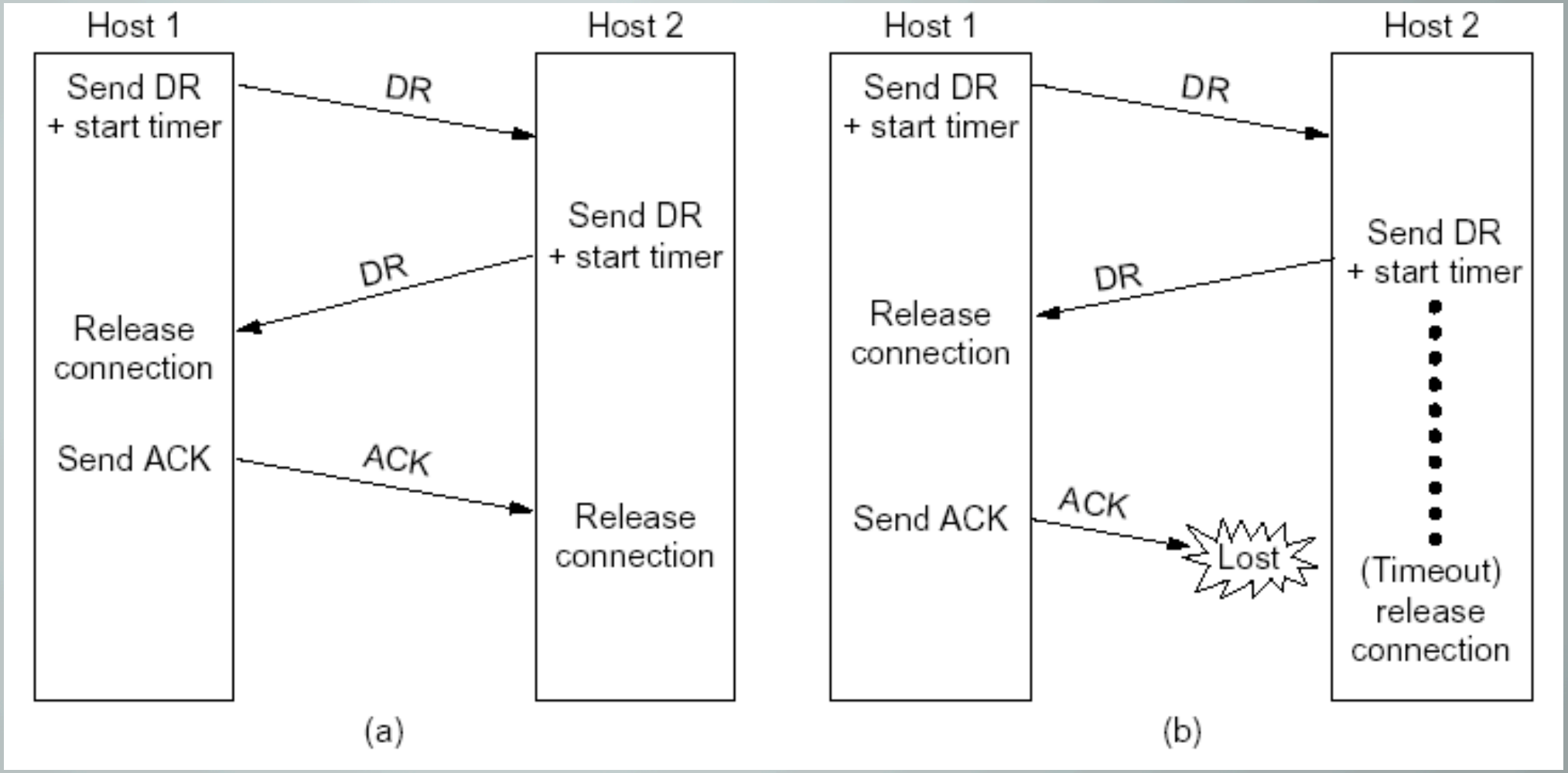
# Liberación de Conexión
## Simétrico

Problema: Podemos diseñar una solución para liberar de tal manera que las dos partes estén de acuerdo? **NO**

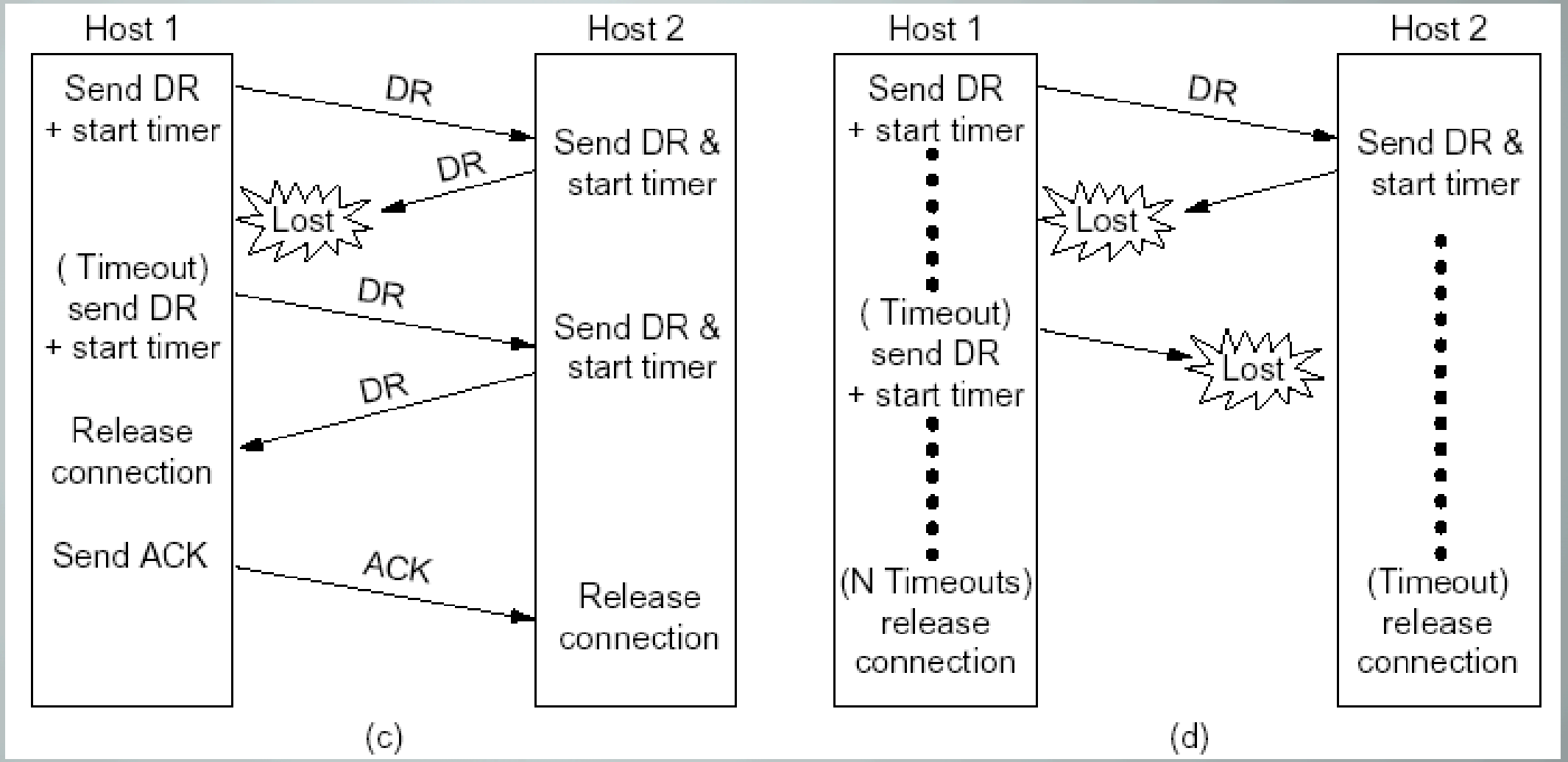
- **Caso Normal:** Host 1 envía un Disconnect Request (DR). Host 2 responde con un DR. 1 responde con ACK y ACK llega a 2.
- **ACK perdido:** Que debe hacer 2? no sabe si su DR llego a 1.
- **DR de 2 perdido**: Que debe hacer 1? Mandar un DR? estamos como al principio.

Solución de aplicación: Usar mecanismos de timeout. Esto atrapa la mayoría de casos pero no es una solución final: algún DR puede perderse, resultando en operaciones half-open

# Liberación de Conexión
## Simétrico



# Liberación de Conexión
## Simétrico



# Control de flujo

Problema: Hosts pueden tener tantas conexiones que hacen imposible asignar una cantidad fija de buffers por conexión para implementar un algoritmos de "sliding windows" i.e., necesitamos un esquema de asignación dinámica.

- Con una red "unreliable", i.e., un servicio unreliable dado por la capa de red, el transmisor deberá guardar TPDUs hasta que sean ACKs
- El receptor podrá tirar TPDUs entrantes si no tiene espacio en los buffers.
- Con una red reliable, el transmisor deberá guardar TPDUs hasta que sean ACKS: la capa de red no puede ayudar! Por que?

En general: El emisor y el receptor necesitan negociar de antemano el número de TPDUs que pueden ser transmitidos en secuencia, tan solo por que el espacio no viene gratis.

# Reservando Buffer

Idea Básica: El emisor pide un número de buffers en lado del receptor cuando abre una conexión. El receptor responde con un "credit grant". Después de esto el receptor asigna mas crédito cuando hay mas buffers disponibles.

| | A | Message | B | Comments |
|---|---|---|---|---|
| 1 | → | < request 8 buffers> | → | A wants 8 buffers |
| 2 | ← | <ack = 15, buf = 4> | ← | B grants messages 0-3 only |
| 3 | → | <seq = 0, data = m0> | → | A has 3 buffers left now |
| 4 | → | <seq = 1, data = m1> | → | A has 2 buffers left now |
| 5 | → | <seq = 2, data = m2> | • • • | Message lost but A thinks it has 1 left |
| 6 | ← | <ack = 1, buf = 3> | ← | B acknowledges 0 and 1, permits 2-4 |
| 7 | → | <seq = 3, data = m3> | → | A has 1 buffer left |
| 8 | → | <seq = 4, data = m4> | → | A has 0 buffers left, and must stop |
| 9 | → | <seq = 2, data = m2> | → | A times out and retransmits |
| 10 | ← | <ack = 4, buf = 0> | ← | Everything acknowledged, but A still blocked |
| 11 | ← | <ack = 4, buf = 1> | ← | A may now send 5 |
| 12 | ← | <ack = 4, buf = 2> | ← | B found a new buffer somewhere |
| 13 | → | <seq = 5, data = m5> | → | A has 1 buffer left |
| 14 | → | <seq = 6, data = m6> | → | A is now blocked again |
| 15 | ← | <ack = 6, buf = 0> | ← | A is still blocked |
| 16 | • • • | <ack = 6, buf = 4> | ← | Potential deadlock |

Pregunta: Que podemos hacer acerca del DEADLOCK potencial

# Control de flujo

**Problema**: Ahora que hemos ajustado la tasa de transferencia entre el emisor y el receptor, consideremos también la capacidad de la red, esta puede no ser suficiente para lo que el emisor y el receptor quieren hacer.

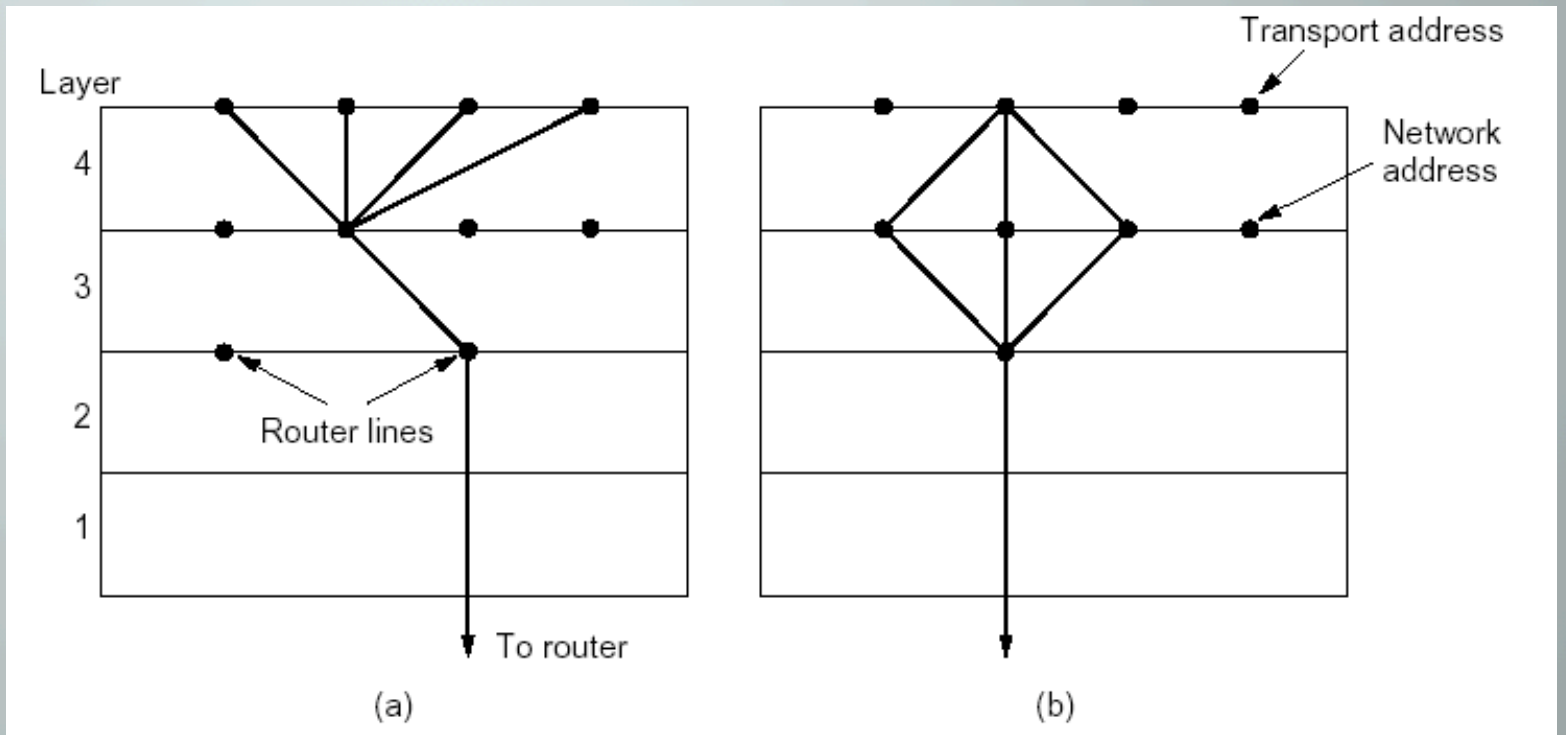
**nota**: si la red maneja *c* TPDUs por segundo, y toma un total de *r* segundos para transmitir, propagar, encolar, procesar un TPDU, y para mandar un ACK el emisor solo necesita mantener *c x r* buffers

**Solución**: Dejemos el emisor estime c y r (COMO?) y que ajuste sus buffers

# Multiplexión

Idea Básica l: Suponiendo que la red ofrece solo una cantidad finita de circuitos virtuales o que el usuario no quiere pagar mucho, entonces pomdemos usar un solo circuito para varias conexiones. (upward multiplexing)

Idea Básica 2: Si un usuario requiere mucho ancho de banda que no puede ser dada por un solo circuito virtual, podemos usar varios circuitos para la misma conexión. (downward multiplexing)



# Recuperación de caídas

**Problema**: En una situación típica, un host responde por cada TPDU que recibe con un ACK. Como debe responder el emisor cuando el receptor cae antes, durante, o después de esta respuesta?

# Recuperación de caídas

**Situación**: Asuma que el emisor es informado de que el receptor se ha recuperado de una caída. Debería el emisor retransmitir el TPDU enviado recientemente, o no? Distinguir entre:

- S0: el emisor no tiene TPDUs sin ACK
- S1: el emisor tiene un TPDU sin ACK

| | Strategy used by receiving host | | | | | |
|---|---|---|---|---|---|---|
| | First ACK, then write | | | First write, then ACK | | |
| Strategy used by sending host | AC(W) | AWC | C(AW) | C(WA) | W AC | WC(A) |
| Always retransmit | OK | DUP | OK | OK | DUP | DUP |
| Never retransmit | LOST | OK | LOST | LOST | OK | OK |
| Retransmit in S0 | OK | DUP | LOST | LOST | DUP | OK |
| Retransmit in S1 | LOST | OK | OK | OK | OK | DUP |

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message