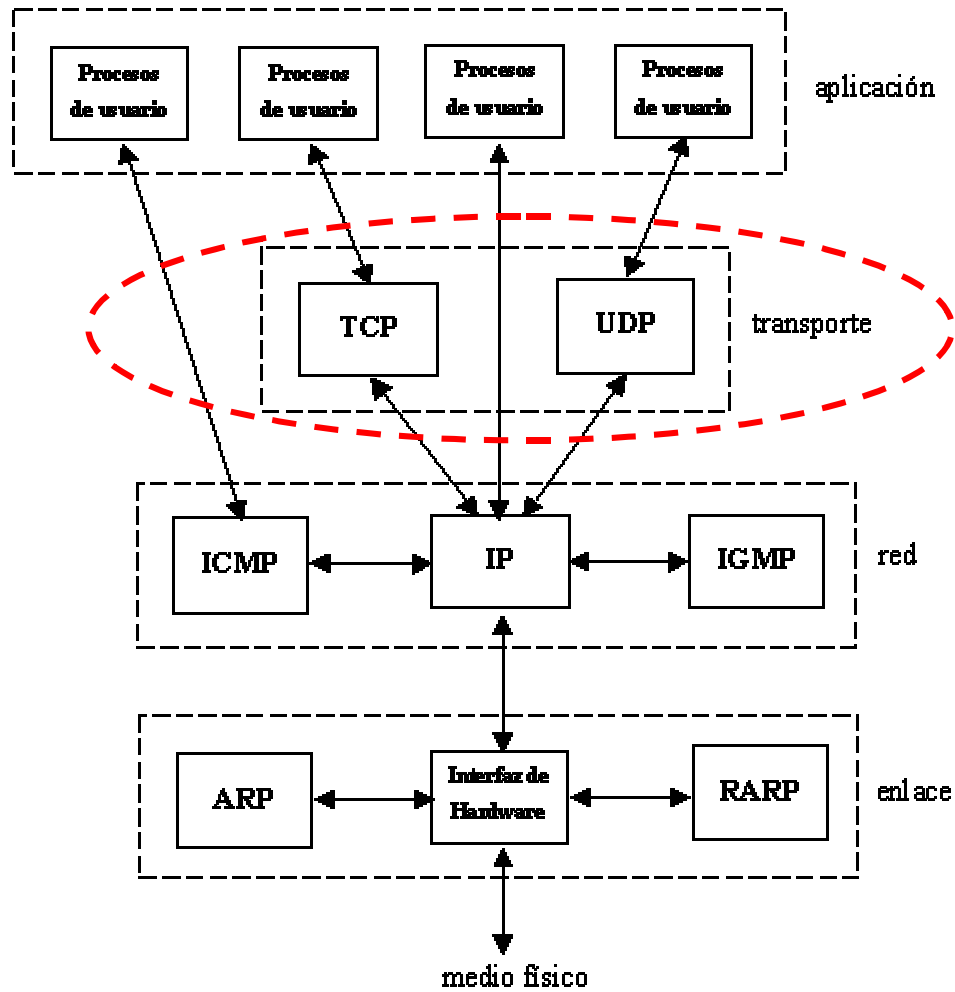


# Protocolos de Extremo a Extremo (Transmission Control Protocol, TCP)

## Contenidos

- Establecimiento y término de conexión
- Revisión a Ventana Deslizante
- Control de Flujo
- Temporizadores Adaptivos

# Contexto

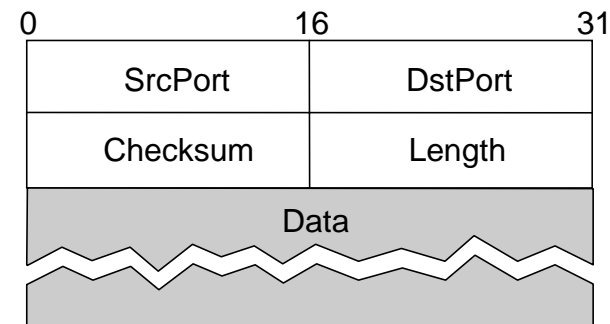


# Protocolos End-to-End

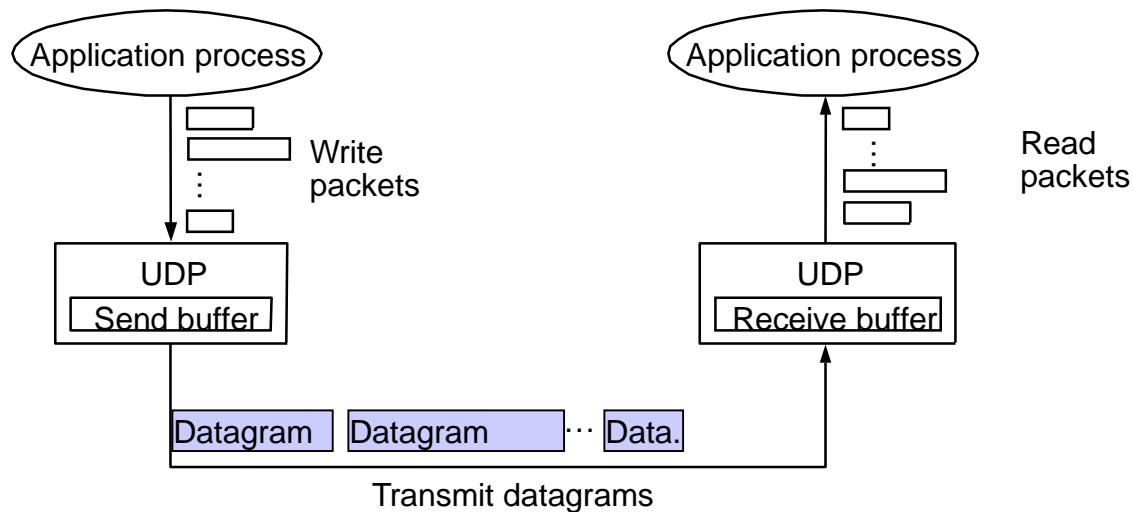
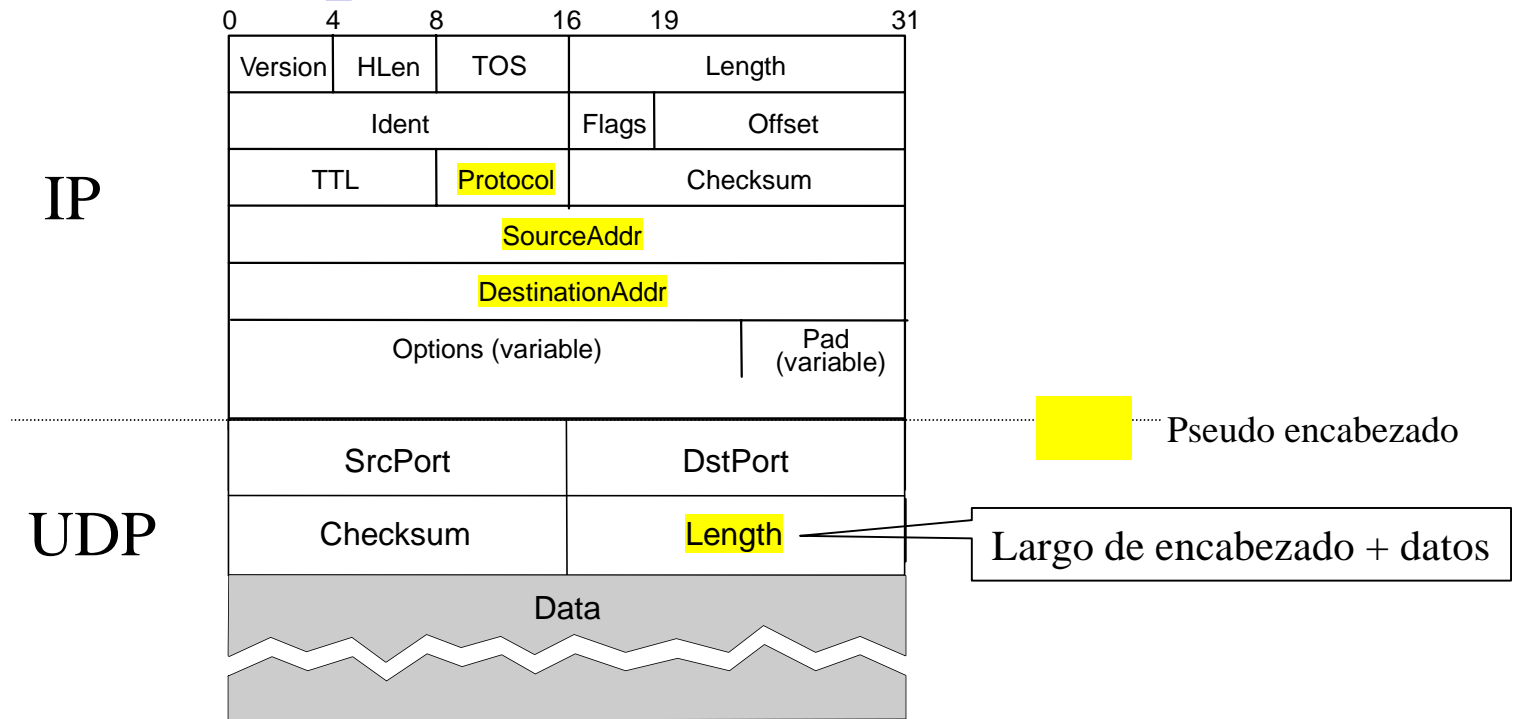
- Se apoyan en la capa Red, la cual es de mejor esfuerzo (best-effort)
  - descarta mensajes
  - re-ordena mensajes
  - puede entregar múltiples copias de un mensaje dado
  - limita los mensajes a algún tamaño finito
  - entrega mensajes después de un tiempo arbitrariamente largo
- Servicios comunes ofrecidos/deseados end-to-end
  - garantía de entrega de mensajes
  - entrega de mensajes en el mismo orden que son enviados
  - entrega de a lo más una copia de cada mensaje
  - soporte para mensajes arbitrariamente largos
  - soporte de sincronización
  - permitir al receptor controlar el flujo de datos del transmisor
  - soportar múltiples procesos de nivel aplicación en cada máquina

# Demultiplexor Simple (UDP:User Datagram Protocol)

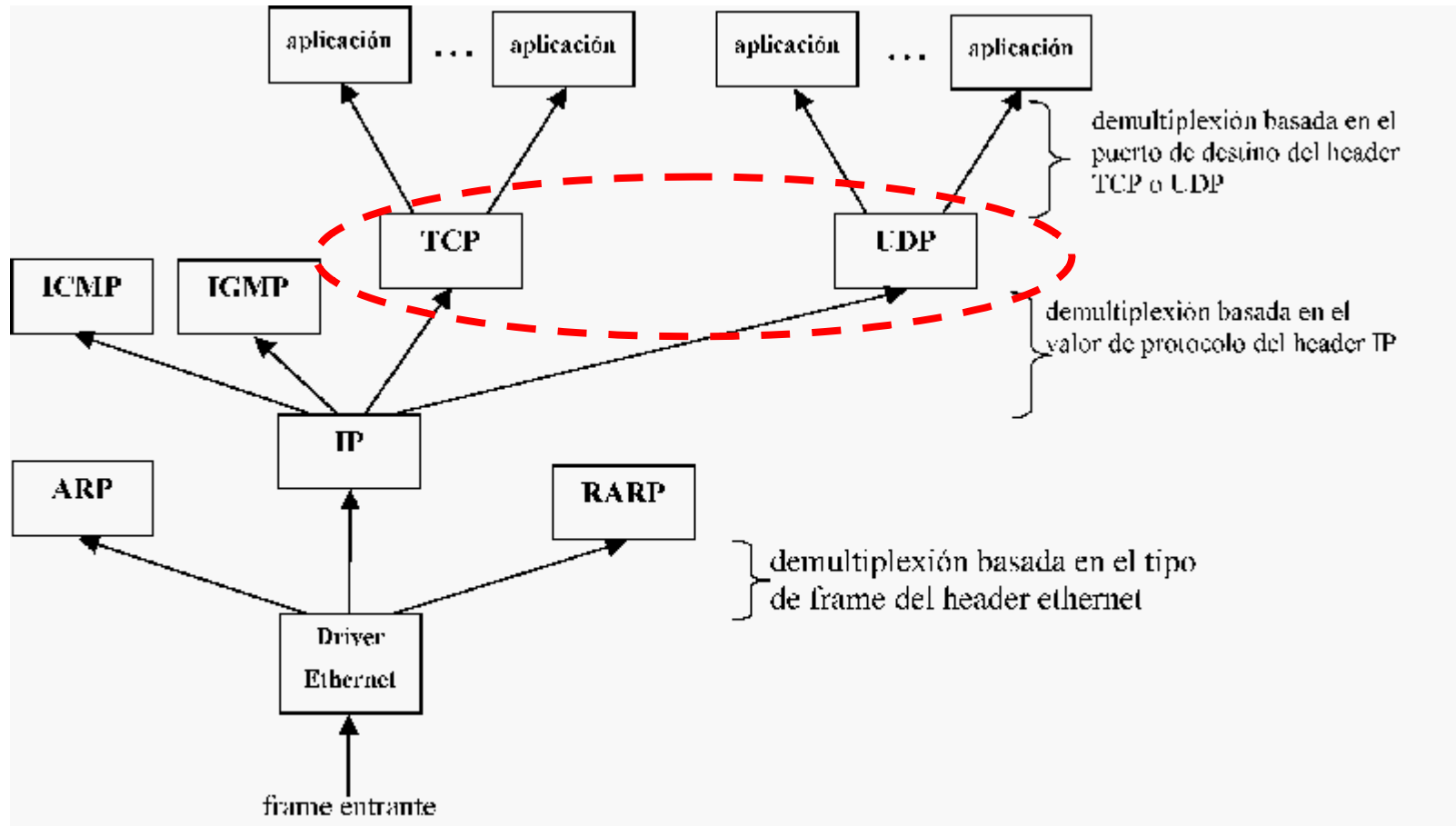
- Servicio de entrega no confiable y no ordenado de datagramas
- Agrega multiplexión
- No hay control de flujo
- Hay puertos definidos en cada extremo
  - servidor posee un puerto *bien conocido*
  - ver [/etc/services](#) en Unix (o Linux)
- Formato de encabezado
- Chequeo se suma opcional
  - pseudo header(IP) + UDP header + data



# Contexto para encabezado UDP

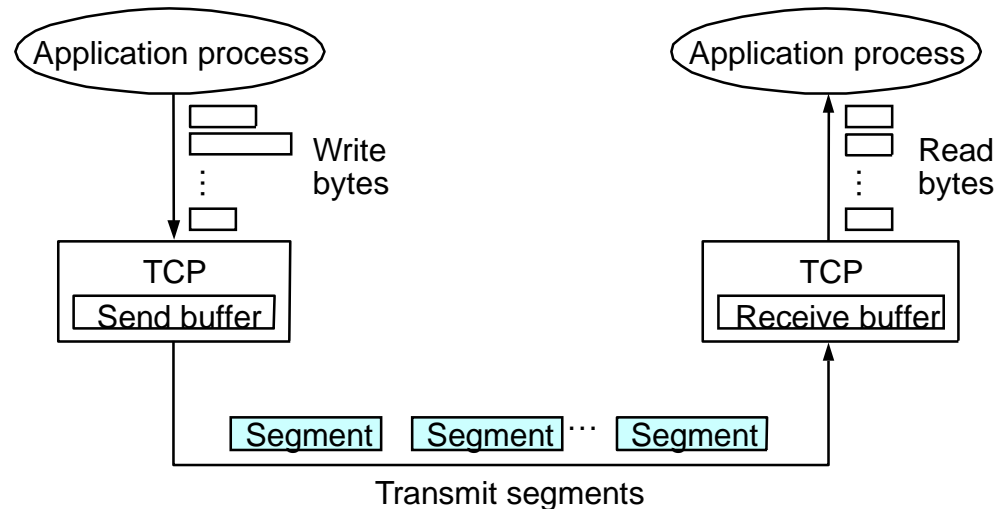


# Niveles de Demultiplexión



# TCP Generalidades

- Orientado a conexión
- flujo de byte
  - app escriben bytes
  - TCP envía *segmentos*
  - app lee bytes
- Full duplex
- Control de flujo: evita que el Tx rebalse al receptor
- Control de congestión: evita que el Tx sobrecargue la red

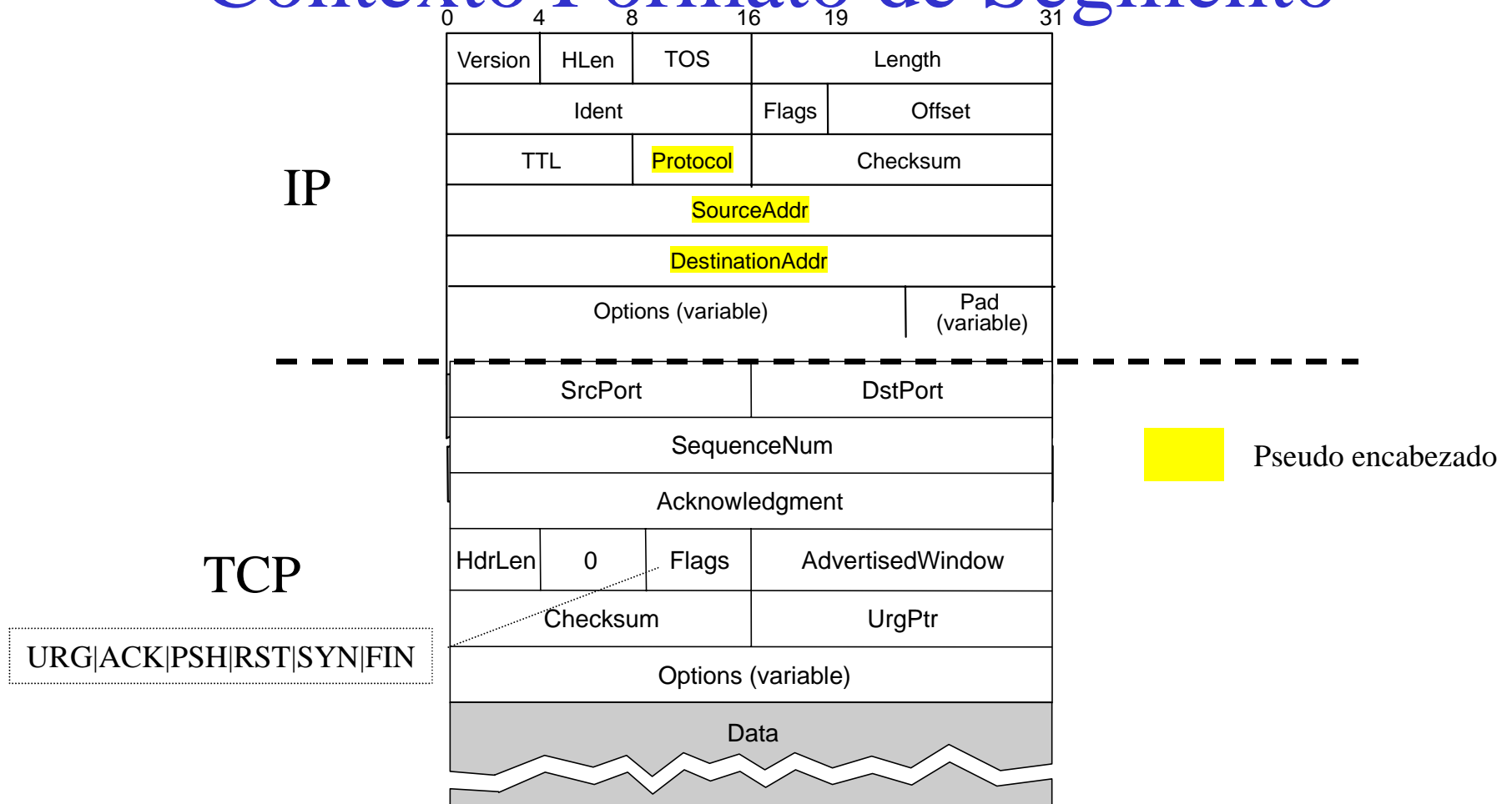


# Capa Enlace de Datos versus Transporte

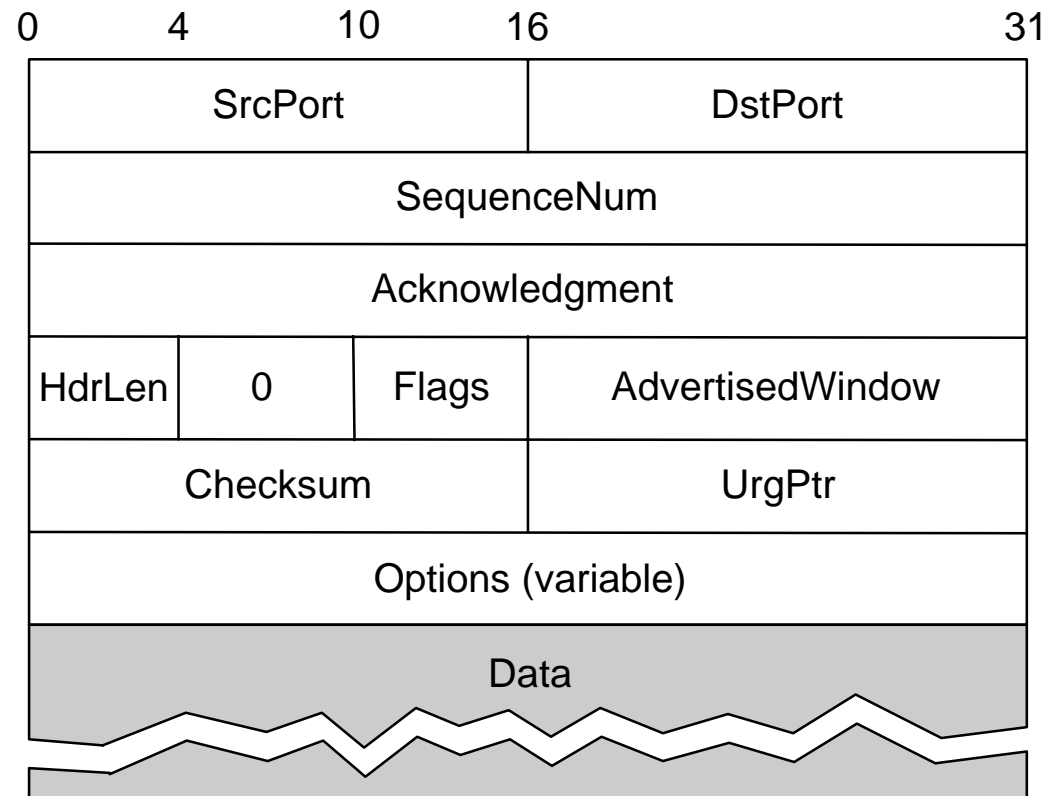
- Potencialmente conecta muchas máquinas diferentes
  - requiere de establecimiento y término de conexión explícitos
- Potencialmente diferentes RTT
  - requiere mecanismos adaptivos para timeout
- Potencialmente largos retardos en la red
  - requiere estar preparado par el arribo de paquetes muy antiguos
- Potencialmente diferente capacidad en destino
  - requiere acomodar diferentes capacidades de nodos
- Potencialmente diferente capacidad de red
  - requiere estar preparado para congestión en la red



# Contexto Formato de Segmento

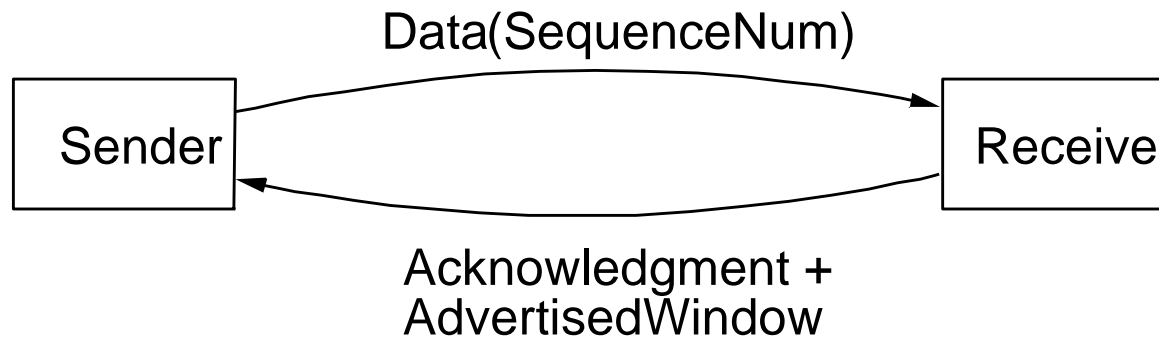


# Formato de Segmento



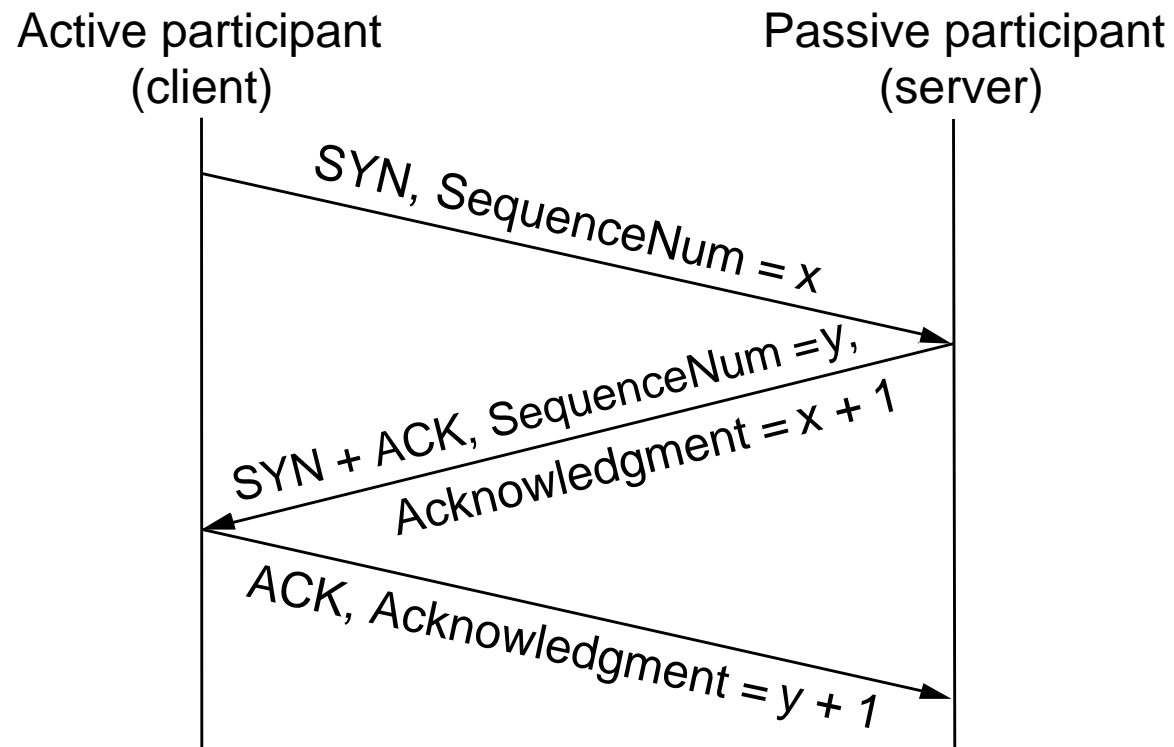
# Formato de Segmento (cont)

- Cada conexión es identificada por la 4-tupla:
  - (SrcPort, SrcIPAddr, DsrPort, DstIPAddr)
- Ventana deslizante + control de flujo
  - acknowledgment, SequenceNum, AdvertisedWindow

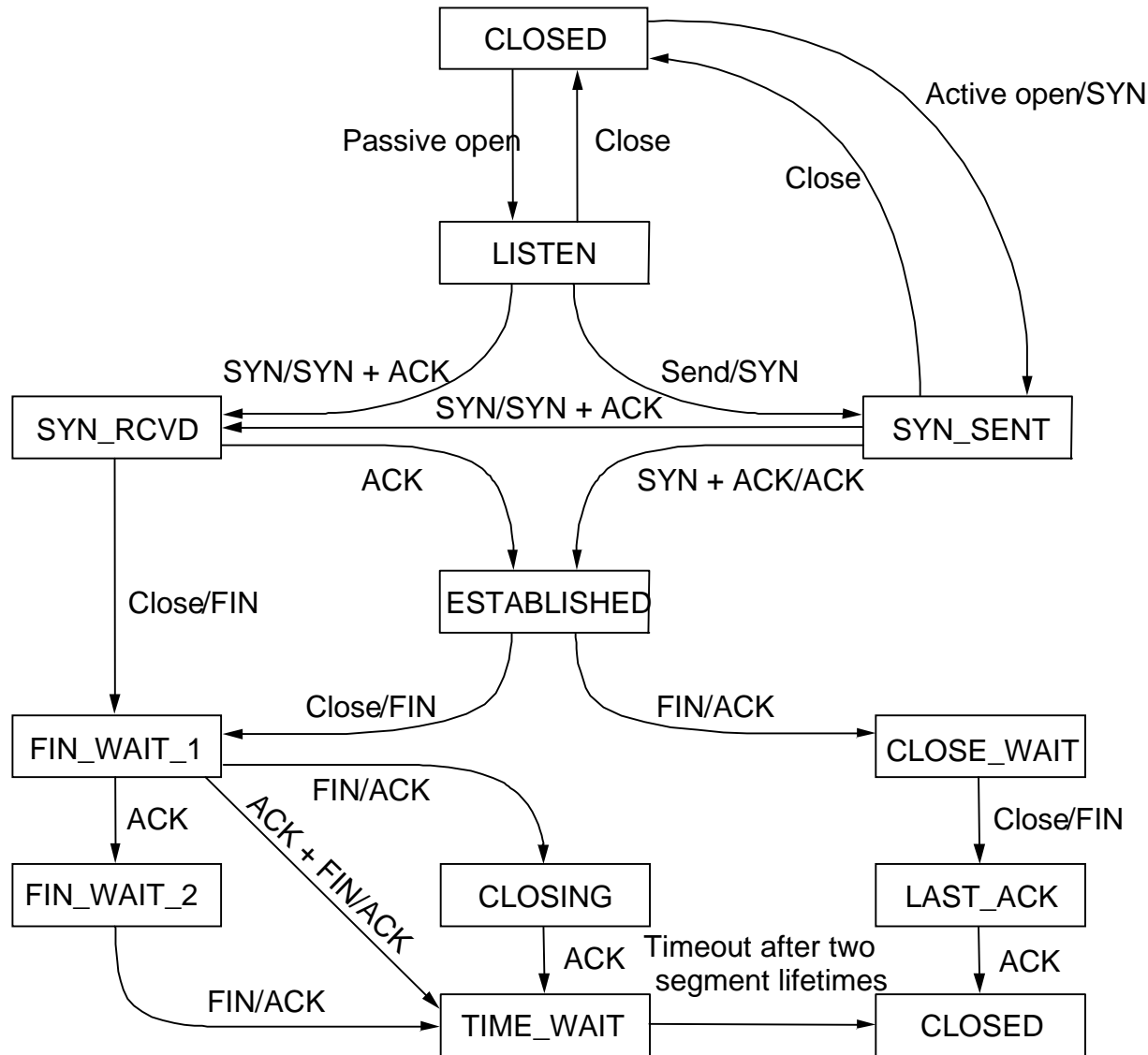


- Flags
  - SYN, FIN, RESET, PUSH, URG, ACK
- Checksum
  - pseudo header(IP) + TCP header + data

# Establecimiento y Término de Conexión

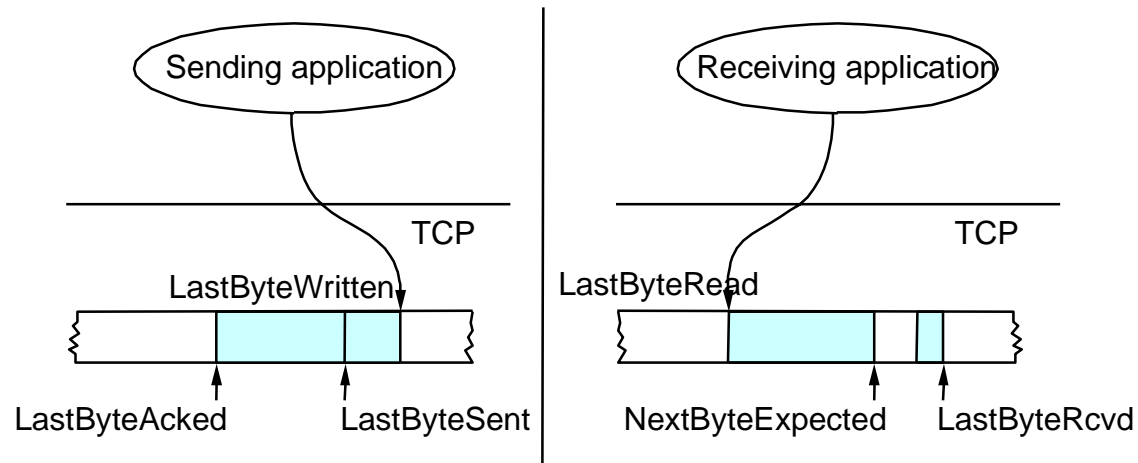


# Diagrama de Estado de Transmisión



ELO-309

# Revisión de Ventana Deslizante



- Lado Transmisor

$$\text{LastByteAacked} \leq \text{LastByteSent}$$

$$\text{LastByteSent} \leq \text{LastByteWritten}$$

Se tiene en buffer los bytes entre  
**LastByteAacked** y  
**LastByteWritten**

- Lado Receptor

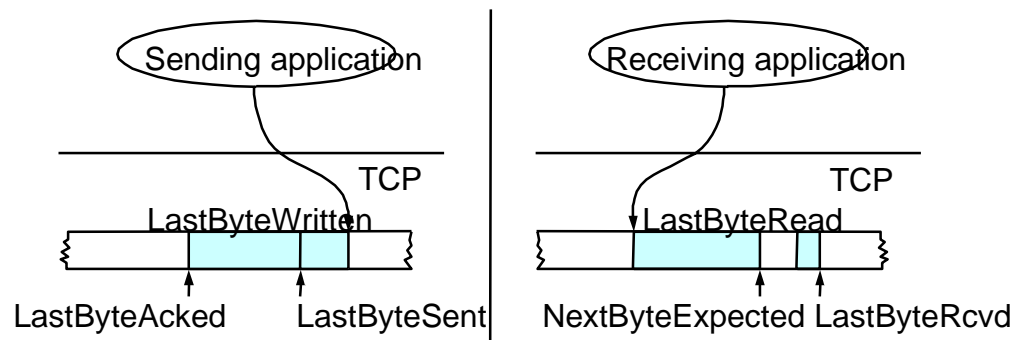
$$\text{LastByteRead} < \text{NextByteExpected}$$

$$\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$$

Se tiene en buffer los bytes entre  
**NextByteRead** y  
**LastByteRcvd**

LastByteRead+1

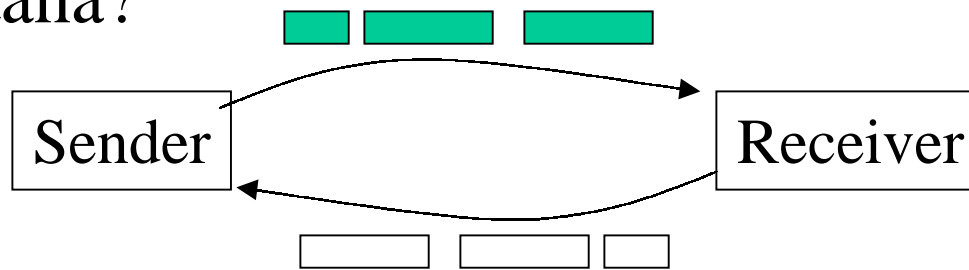
# Control de Flujo



- Tamaño del buffer de envío: **MaxSendBuffer**
- Tamaño del buffer de recepción: **MaxRcvBuffer**
- Lado receptor
  - $LastByteRcvd - LastByteRead \leq MaxRcvBuffer$
  - $AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - NextByteRead)$
- Lado Transmisor
  - $LastByteSent - LastByteAcked \leq AdvertisedWindow$
  - $EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)$
  - $LastByteWritten - LastByteAcked \leq MaxSendBuffer$
  - Bloquear Tx si  $(LastByteWritten - LastByteAcked) + y > MaxSenderBuffer$ , y bytes que la aplicación desea escribir.
- Siempre enviar ACK en respuesta a la llegada de segmentos de datos
- Tx persiste enviando 1 byte cuando **AdvertisedWindow = 0**

# Síndrome de Ventana estúpida (Silly)

- ¿Qué tan agresivamente el Tx explota la apertura de ventana?



- Soluciones en lado Receptor
  - Retardar los acuses de recibo



# Algoritmo de Nagle

- ¿Qué tanto tiempo el Tx retarda la transmisión de datos para así acumular más y llenar un segmento?
  - Demasiado largo: afecta aplicaciones interactivas
  - Demasiado corto: Utilización de la red es pobre
  - Estrategias: Basadas en timers v/s auto relojes
- Cuando la aplicación genera datos adicionales:
  - Si se llena un segmento (y la ventana está abierta): enviar
  - Sino
    - Si hay datos sin ack en Tx: dejar en buffer hasta llegada de ack
    - sino: enviar datos

# Protección contra reapariciones de igual número de secuencia

- **SequenceNum de 32 bits**

Bandwidth	Tiempo hasta tener problema
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

# Mantenimiento de la tubería llena

- **AdvertisedWindow de 16 bits**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

64 KB

Asumiendo RTT de 100 ms

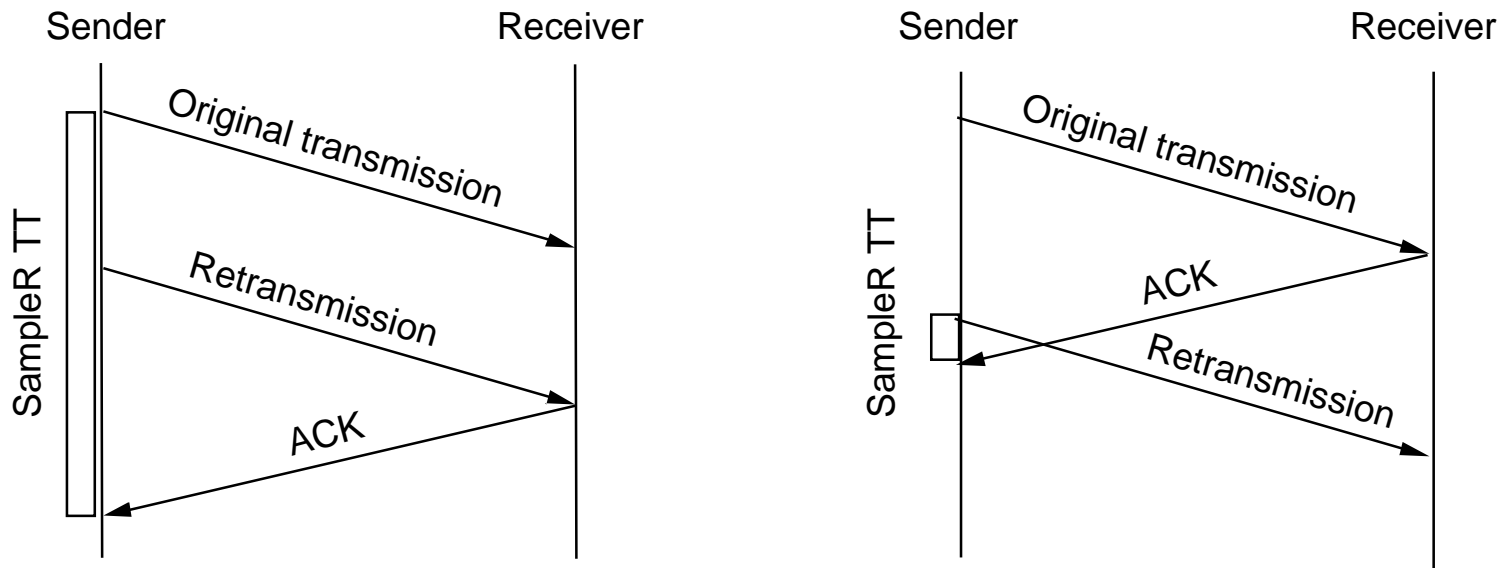
# Extensiones de TCP

- Son implementadas como opciones del encabezado
- Almacenar marcas de tiempo en segmentos de salida
- Extender espacio de secuencia con marca de tiempo de 32-bit
- Desplazar (escalar) ventana avisada. La idea es medir la ventana en unidades de 2, 4, 8 bytes.

# Retransmisión Adaptiva (Algoritmo Original)

- Mide **sampleRTT** para cada par segmento/ ACK
- Calcula el promedio ponderado de RTT
  - **EstimatedRTT** =  $\alpha \times \text{EstimatedRTT} + \beta \times \text{sampleRTT}$
  - donde  $\alpha + \beta = 1$
  - $0.8 \leq \alpha \leq 0.9$
  - $0.1 \leq \beta \leq 0.2$
- Fijar timeout basado en **EstimatedRTT**
  - **TimeOut** =  $2 \times \text{EstimatedRTT}$

# Algoritmo de Karn/Partridge



- No considerar RTT cuando se retransmite
- Duplicar timeout luego de cada retransmisión

# Algoritmo de Jacobson/ Karels

- Nueva forma de calcular el promedio de RTT
- **Diff** = **sampleRTT** - **EstRTT**
- **EstRTT** = **EstRTT** + (  $\delta$  x **Diff** )
- **Dev** = **Dev** +  $\delta$  ( |**Diff**| - **Dev** )
  - donde  $\delta$  es un factor entre 0 y 1 (Por ejemplo 1/8)
- Considerar varianza cuando fijamos el timeout
- **TimeOut** =  $\mu$  x **EstRTT** +  $\phi$  x **Dev**
  - donde  $\mu = 1$  y  $\phi = 4$
- Notas
  - Los algoritmos son tan buenos/malos como la granularidad del reloj.
  - Un preciso mecanismo de timeout es importante para controlar la congestión (más adelante)
  - Además de controlar congestión, la idea es no retransmitir cuando no es necesario.