3.1. INTERCONEXIÓN DE REDES (I)

☞ En la práctica existe más de una red y cada máquina usando en cada capa protocolos probablemente distintos.

☞ Existe realmente mucha heterogeneidad: más de 20000 SNAs, 2000 DECNETs e incontables tipos de LAN.

☞ Es difícil que las redes particulares desaparezcan para integrarse todas en OSI.

☞ Razones para la heterogeneidad:

- Ya existían antes de aparecer OSI múltiples redes.
- IBM aún vende redes SNA.
- Las estaciones UNIX suelen llevar TCP/IP.
- Las LANs son raramente OSI.

☞ Aparece *hardware* muy barato que contribuye a la diversidad y a decisiones muy locales de compañías (no generales).

☞ Ejemplos de la necesidad de la Interconexión de Redes:

- **LAN-LAN** Transmisión de un fichero entre dos departamentos.
- **LAN-WAN** Transmisión remota por *email* de un dpto. a otro.
- **WAN-WAN** Intercambio remoto de ficheros.
- **LAN-WAN-LAN** Dos departamentos remotos comunicándose.

RELAYs

- La unión entre dos redes tiene lugar a través de la inserción de cajas negras para manejar la conversión de paquetes que tiene lugar al pasar de una a otra. A estos dispositivos se les llama RELAY.

- *Tipos*:

  - **Bilaterales**: Conectan directamente dos redes entre sí.
  - **Multilaterales**: Conectan directamente varias redes entre sí.

3.1. OSI E INTERCONEXIÓN DE REDES (II)

☞ El modelo OSI contempla la Interconexión de Redes a Nivel de Red. No está bien definido (como ocurre con la seguridad).

☞ No se tomaron en cuenta las opiniones de ARPANET. CCITT también sufrió una falta de predicción asignando 4 dígitos decimales (10000 redes) para direccionamiento. OSI suponía una o dos redes públicas por país. Las redes privadas se agruparían en redes públicas mayores. Esto no ocurrirá ni aunque todas fuesen OSI.

☞ El modelo OSI de la capa de red distingue 3 subcapas:

a. SUBCAPA DE ACCESO A LA SUBRED

- ❖ Maneja el protocolo específico de la subred física usada.
- ❖ Genera y recibe paquetes de datos y control.
- ❖ El software está diseñado para acceder a la subred real. No hay garantías de que funcione en otras subredes.

b. SUBCAPA DE MEJORA DE LA SUBRED

- ❖ Armoniza el funcionamiento en subredes que ofrezcan servicios distintos.
- ❖ Presenta a la subcapa interred un modelo uniforme de servicios.

c. SUBCAPA INTERRED

- ❖ Conecta de forma lógica las dos subredes.
- ❖ Utiliza NSAP de manera uniforme para que los procesos de transporte que usan sus servicios se conecten. Luego se direcciona realmente dichos procesos, y no a máquinas sólamente.
- ❖ Su principal función es el encaminamiento extremo-a-extremo. Decide el próximo enlace y cuál de entre las subredes disponibles para cada paquete.
- ❖ El encaminamiento es similar al presentado para una única subred. El encaminamiento *Jeráquico* está muy indicado para grandes redes.
- ❖ No confundir *INTERRED* (*internet*) con la red muy conocida 'Internet'.

3.2. RELAYs

✟ Son dispositivos para la conexión de dos subredes de comunicación.

✟ El sistema *relay* puede establecerse a cualquier nivel del modelo OSI.

✟ Existen 4 **tipos** muy comunes de *relays*:

- ❖ **REPETIDORES (y HUBS)**
  - ⌦ Amplifican señales eléctricas cuando se usan cables largos (no sólo cables, sino también otros medios de transmisión física).
  - ⌦ Sólo copian los bits que reciben por su entrada en su salida.
    - ❑ Por ejemplo la temporización en 802.3 del protocolo MAC permite cables de hasta 2,5 Km, pero los transceivers sólo dan potencia para 500 m. La solución es usar repetidores para aumentar la distancia.
- ❖ **PUENTES (y SWITCHES)**
  - ⌦ También se les llama BRIDGES.
  - ⌦ Tienen implementado hasta el nivel de Enlace de Datos.
  - ⌦ Son dispositivos de Almacén y Envío. Aceptan una trama, la pasan a DLL para verificar *checksums* y se devuelve a la capa física para envío a la otra subred.
  - ⌦ Pueden hacer algunos cambios en las cabeceras de las tramas, pero no conocen ni usan las cabeceras de nivel de red para nada.
- ❖ **PASARELAS**
  - ⌦ Trabajan a Nivel de Red.
  - ⌦ Algunos autores llaman Gateway a un *relay* que trabaja a cualquier nivel OSI y ROUTER a un Gateway a nivel de Red. Para nosotros Gateway y Router es lo mismo.
  - ⌦ Las subredes conectadas por un *gateway* pueden ser mucho más distintas entre sí que usando un *bridge*.
  - ⌦ Una decisiva ventaja sobre el *bridge* es que un *gateway* puede conectar redes con formatos de direccionamiento distintos. Por ejemplo una LAN 802 con direcciones binarias de 48 bits y una red X.25 con direcciones X.121 decimales de 14 dígitos.
- ❖ **CONVERSORES DE PROTOCOLO**
  - ⌦ Trabajan a Nivel de Transporte o Aplicación (normalmente).
  - ⌦ Su trabajo es mucho más complejo que el de un Gateway: convertir entre dos protocolos sin perder mucho significado.
    - ❑ Ejemplos son la conversión de TP4 (OSI) a TCP (Internet) a nivel de transporte, o la conversión de SMTP (Internet) a SMS (telefonía móvil).

✟ Las conversiones de un *relay* son a veces imposibles de llevar a cabo. Su dificultad depende de la importancia de las diferencias entre las dos subredes.

3.3. PUNTOS CONFLICTIVOS EN EL DISEÑO DE PUENTES (I)

- Cada LAN usa su propio formato de trama. No existe ninguna razón técnica para dicha **incompatibilidad**, sólo que las grandes compañías promotoras de los 3 estándares (Xerox, General Motors e IBM) no cooperaron entre sí. Esto supone que los puentes consumirán CPU y memoria en los cambios de formatos, *checksums*, etc... y pueden aparecer errores sutiles en sus memorias.

- Un problema más serio es el **ancho de banda** diferente de las diferentes redes.

  802.3 1 a (**10**) 20 100 ... Mbps 802.4 a 1 a **10** Mbps 802.5 a 1/**4** Mbps

  - Esto supone que si se conecta una 802.3 o 802.4 a una 802.5 el puente deberá retener los datos en *buffer*s porque no puede entregarlos tan rápido. Algo similar pasa al conectar una 802.4 a una 802.3 estándar, ya que la primera tiene 10 Mbps reales, mientras que la segunda malgasta parte de su tasa en las colisiones.

- Un problema sutil son los **temporizadores**. Si una 802.4 envía un mensaje largo a una 802.5 y espera confirmación el retraso en el puente por la menor velocidad de la 802.5 puede hacer saltar la temporización y retransmitir. Tras múltiples intentos abortará la transmisión e informará a la capa de transporte de *destino muerto*. Es posible también en los Gateways.

- Los tres principales tipos de LAN 802 tienen una máxima longitud de trama diferente:

  - **802.3** 1518 bytes (estándar a 10 Mbps)
  - **802.4** 8191 bytes
  - **802.5** 5000 bytes (si MTHT es 10 ms)

  - Si la trama de una red es demasiado larga para la destino **no se especifica** en 802 si se puede o cómo fragmentar en tramas más pequeñas. En aras de la transparencia se debería descartar este tipo de transmisiones interred demasiado largas (aunque el problema sí que tiene solución). Esto es así porque todos los protocolos asumen que una trama llega o no llega (no se contempla una llegada parcial: fragmentos que sí o que no).

- **Puentes Remotos**: variante para conectar LANs dispersas geográficamente. Se usa un protocolo punto a punto como PPP (túnel ideal si las redes son iguales entre sí) o bien uno traductor (elimina/genera cabecera/cola).

3.3. PUENTES IEEE 802.x a 802.y (II)

802.3 a **802.3**

☆ El puente puede **agotar su memoria** si la red destino está muy saturada. Esto supone que descartarán tramas. Siempre puede ocurrir cuando se envía a una 802.3 (no lo volveremos a mencionar). Con 802.4 y 802.5 las estaciones y el puente tienen asegurada la obtención del token y una retención de este tipo no es un problema.

802.4 a **802.3**

☆ Las tramas 802.4 contienen **bits de prioridad** que no existen en las 802.3. Esto supone que si dos 802.4 se comunican a través de una 802.3 estos bits (la prioridad) pierden su sentido.

☆ El uso de **tokens temporales** en 802.4 (trama con bit de token a 1 que viaja al destino para permitirle confirmar) hace que un puente no sepa qué hacer: si se lo confirma al origen es mentira porque el destino podría estar muerto y si no se lo confirma probablemente el origen informará a la capa superior de que el destino está muerto. No parece haber solución.

802.5 a **802.3**

☆ Las tramas 802.5 tienen los bits **A** y **C** en el byte de *Frame Status* que el destino debe rellenar para indicar al origen si se direccionó bien la trama y si el destino la copió. El puente podría confirmar ambos pero si el destino está realmente muerto habría un gran problema: en todo caso el puente cambia la semántica.

802.3 a **802.4**

☆ El problema es qué valores usar para los bits de **prioridad**. Una solución es que el puente retransmita a la mayor prioridad porque las tramas que le llegan ya habrán sido retrasadas considerablemente.

802.4 a **802.4**

☆ El problema es el **préstamo temporal** del token, pero si el **B** retransmite a la mayor prioridad posible puede que el destino confirme antes del *timeout*.

802.5 a **802.4**

☆ Los problemas son los bits **A** y **C** y además que las definiciones de **prioridad** son distintas (al menos existen bits de prioridad en ambas). El puente puede copiarlos unos en otros y “rezar” para que funcione el cambio.

802.3 a **802.5**

☆ El puente debe inventarse los **bits de prioridad** para la 802.5.

802.4 a **802.5**

☆ Los problemas son potenciales respecto a la transmisión de **tramas muy largas**. Además está el problema del **préstamo temporal** de token.

802.5 a **802.5**

☆ El problema es el del manejo de los bits **A** y **C**.

3.3. TIPOS DE PUENTES (III)

- Cuando el comité IEEE 802 se sentó a diseñar un estándar LAN realmente produjo 3 estándares incompatibles.

- Al reunirse para diseñar un estándar para puentes mejoró notablemente y produjeron sólo 2 estándares incompatibles.

-

3.3. PUENTES TRANSPARENTES (IV)

- Operan en modo *promiscuo*: aceptan cualquier trama desde cualquier LAN a la que estén conectados. Para cada paquete debe decidirse si descartarla o enviarla y si es así por qué LAN hacerlo. Usan una **tabla hash** para asociar a cada nodo destino una de sus LANs por donde alcanzarlo.

- Cuando llega una trama con destino desconocido (por ejemplo recién conectado el puente) utiliza *Inundación* para encaminarla (la repite por el resto de salidas). Con el tiempo aprende dónde están los destinos y deja de usar inundación.

- **Algoritmo de Aprendizaje Hacia Atrás** (Baran)
  - Ya que operan en modo promiscuo pueden consultar la dirección de origen de cada nuevo mensaje que aparezca en una red, anotando en su tabla en qué red lo detectó para saber que es por dicha red por donde se alcanza al nodo que envió ese paquete.
  - Tras asociar un nodo a una LAN en su tabla cualquier paquete para dicho nodo en otra LAN lo pasa a ésta, mientras que si se trata de un paquete en la misma LAN entonces lo descarta (lo ignora).
  - Además de la red se anota en la tabla el *tiempo* en que se vió por última vez un paquete de cada nodo que envíe. Periódicamente el puente purga su tabla de las entradas más antiguas (varios minutos) y de esta forma se adapta a topologías cambiantes. Luego si una máquina está silenciosa mucho tiempo al recibir después nuevo tráfico se usará inundación hasta que ella misma envíe alguna trama a la red.

- Por seguridad se pueden usar *puentes paralelos*. Esto provoca problemas de bucles. Para evitarlo todos los puentes se organizan en *árbol* abarcando todas las LANs. De esta manera se evitan ciclos al existir un único camino origen-destino.

- Para construir el árbol se elige un nodo raíz (por ejemplo el puente cuyo número de serie sea menor) y cada puente difunde su identidad y los puentes que conoce conectados también a sus LANs. El árbol se construye haciendo que los paquetes elijan el camino más corto a la raíz.

- No todos los puentes estarán en el árbol necesariamente. El algoritmo se ejecuta con periodicidad para detectar cambios de topología.

- El algoritmo para la construcción del árbol presenta problemas de escala en grandes redes (p.ej. telefónicas). Si dos grupos de LANs se comunican por puentes conectados por una WAN el algoritmo necesita que se optimice su ejecución para minimizar los paquetes necesarios a transmitir sobre la WAN.

3.3. PUENTES POR ENCAMINAMIENTO FUENTE (V)

⇨ Mejoran el uso del ancho de banda frente a los transparentes que usan **un árbol**.

⇨ Se asume que cada emisor sabe si el destino está o no en su LAN. Cuando envía una trama a una LAN distinta activa el bit más significativo de la dirección destino e incluye en la *cabecera* la **ruta exacta** que la trama debe seguir.

⇨ El camino se construye como una sucesión *puente-LAN-puente-LAN...* donde cada LAN tiene un identificador único de **12** bits y cada puente otro de **4** bits pero único sólo en su LAN (otro puente en otra LAN puede tener el mismo id.).

⇨ Un puente sólo se interesa por las tramas cuyo *MSB* de la dirección destino está a **1**. Entonces busca en la ruta de la trama el número de LAN por el que llegó. Si tras éste le sigue su propio número de puente entonces comprueba cuál es la siguiente LAN y la pasa a ésta. Si no se trata de su número de puente la ignora.

⇨ **Implementaciones Posibles**

a) *SOFTWARE*: Modo promiscuo. Copia todas las tramas a su memoria para comprobar su MSB. Si no está activo entonces termina con ella.

b) *HIBRIDO*: La interfaz a la red inspecciona el MSB y entrega la trama al puente sólo si está a 1. Una interfaz por *hardware* reduce el Nº de insp.

c) *HARDWARE*: La interfaz además de comprobar el MSB también busca en la ruta para ver si su puente asociado está en ella. Sólo le entrega la trama si este puente la debe retransmitir. Ahorra mucho tiempo de CPU.

La implementación **Sw** requiere poco **Hw** pero la CPU debe ser rápida, mientras que la implementación Hw funciona con una CPU más lenta pero necesita circuitos adicionales (además podría manejar más LANs).

⇨ Para conocer cualquier camino que necesite un nodo debe difundir **una Trama de Descubrimiento** que los demás nodos difunden a su vez. Cuando la respuesta desde el destino vuelve, cada puente anota su identidad. El origen recibirá muchos caminos distintos y elegirá el mejor de entre ellos según su criterio.

⇨ Aunque se asegura encontrar la mejor ruta (¡se exploran todas!) la **explosión** en el número de tramas difundidas provoca congestión en el sistema. Algo parecido ocurre en los puentes transparentes pero su explosión es *lineal* y no *exponencial* respecto al número de nodos en la red, como ocurre aquí.

⇨ Las rutas hacia los distintos destinos se almacenan en cachés en los *hosts* para limitar esta explosión, pero en todo caso esto supone consciencia en los *hosts* de la existencia de los puentes, problemas de administración, etc. y por supuesto esto dista mucho de ser transparente.

3.3. COMPARACION ENTRE PUENTES IEEE 802 (VI)

Concepto	Transparente	Encaminamiento Fuente
Orientación	Sin Conexión	Orientado a la Conexión
Transparencia	Totalmente	No Transparente
Configuración	Automática	Manual
Encaminamiento	Sub-óptimo	Óptimo
Localización	Aprendizaje Hacia Atrás	Tramas de Descubrimiento
Fallos	Manejados por el puente	Manejados por los *hosts*
Complejidad	En los puentes	En los *hosts*

TRANSPARENTES	ENCAMINAMIENTO FUENTE
Invisibles a los *hosts* y Compatibles 802	Ni invisibles ni compatibles
Se adaptan a la topología	Difíciles de instalar y administrar
Usan sólo un árbol entre los puentes	Encaminam. óptimo y Distrib. de carga
Problema: Localización inicial de nodo	Problema: Explosión exponencial de D.
Manejan fallos sin que los *hosts* lo sepan	Si cae un puente: detección lenta y cara

⌦ Los puentes por encaminamiento fuente sin uso de *hardware* inteligente tienen un alto tiempo de proceso por paquete. Además complican a los *hosts* en el almacenamiento, descubrimiento y manejo de rutas, consumiendo sus tiempos de CPU. Ya que típicamente existe un número de *hosts* varios órdenes de magnitud mayor que el número de puentes parece mejor poner la complejidad y el coste extra en los puentes y no en todos los *hosts*.

3.4. PASARELAS -GATEWAYs- (I)

- Los *Gateways* operan a nivel de **red** mientras que los puentes operan a nivel de **enlace**.

- Los Gateways son más flexibles que los puentes. Pueden realizar conversiones entre **formatos de dirección** de redes distintas.

- Son más **lentos** que los puentes debido a que necesitan realizar un mayor volumen de cálculos.

- Se suelen usar en **WANs** donde nadie espera que manejen 10000 paquetes por segundo ni velocidades similares y el *retraso* es admisible.

- Existen *dos tipos* de *gateways* según el tipo de redes para el que estén pensados:

  ⇨ **Gateways Orientados a la Conexión**

  ⇨ **Gateways Sin Conexión**

- En TCP/IP *Gateway=Router* y en entornos de redes locales *un Gateway se supone un conversor de protocolos* para conectar redes heterogéneas.

3.4. GATEWAYS ORIENTADOS A LA CONEXIÓN (II)

☞ Pensados para el modelo OSI de *Internetworking*: Concatenación Orientada a la Conexión de redes que usan Circuito Virtual.

☞ Los CVs son a nivel de red (en puentes es a nivel de enlace). Un puente puede manejarse fácilmente por parte del propietario de la LAN, pero un Gateway que conecte a dos WANs de distintos países provoca problemas de gestión. La solución es dividirlo en dos Medios-Gateways separados por un cable. El único conflicto es acordar el protocolo a usar en dicho cable. Cada organización maneja su mitad como más le convenga.

☞ Un protocolo típico sobre cabe entre medios-gateways es X.75 (CCITT) que es casi idéntico a X.25, basado en la idea de CVs concatenados interredes.

☞ El establecimiento de CVs interred es similar al normal intrarred. Los semi-Gateways eligen al próximo semi-gateway al que enviar el paquete de setup del CV, así hasta alcanzar el *host* remoto. Los gateways al reenviar un paquete cambian apropiadamente el número de CV y realiza los cambios de formatos necesarios.

☞ Todos los paquetes de datos atravesarán la misma secuencia de gateways a menos que los CVs se implementen internamente usando datagramas. En la práctica las redes que usan CVs entre redes también los usan internamente.

☞ El proceso de uso es como el visto para CVs intrarred pero sólo es fija la secuencia de gateways y no necesariamente la secuencia de IMPs.

☞ Cuando se conectan dos redes con X.25 el host conecta con la subred en X.25. El protocolo interno IMP-IMP no se especifica en CCITT y probablemente sea distinto para cada subred. La misma libertad tienen los gateways para comunicar con el resto de la subred pero las opciones son claras: usar el mismo protocolo que IMP-IMP (gateway como IMP) o usar X.25 (gateway como *host*).

☞ Aunque el protocolo X.75 sólo se usa en líneas G-G dicta la arquitectura de CVs concatenados, al requerir que todos los paquetes de una conexión pasen por el mismo CV entre G y G.

3.4. GATEWAYS SIN CONEXIÓN (III)

☞ Usados en el modelo CCITT de redes conectadas por *datagramas* (servicio S.C.).

☞ En el viaje de un mensaje de transporte la capa de transporte podría subdividirlo en datagramas (si es demasiado largo) su homónima en el destino los reensambla. Para pasar de una red a otra (de G a G) cada gateway enmarca la trama con la cabecera (*header*) y cola (*trailer*) a nivel de *Enlace de Datos* (DLL) apropiados para la red en que lo deposita, tras despojarla de los correspondientes a la red de la que la recibe.

☞ Cada red le impone un tamaño máximo a los paquetes. Algunas causas son:

- *Hardware* : Anchura del slot de transmisión TDM.
- *Sistema Operativo* : Todos los *buffer*s son de 512 bytes de tamaño.
- *Protocolos* : N. de bits dedicados a la longitud del paquete.
- *Cumplir con un Estándar* : Nacional o Internacional.
- *Deseo de Reducir el Error* : Inducir retransmisiones a algún nivel.
- *Evitar que un paquete ocupe el canal Demasiado Tiempo.*

☞ Los diseñadores no son libres de elegir su propio tamaño máximo. Ejemplos:

- **HDLC** : ∞
- **802.4** : 65.528 bits
- **X.25** : 32.768 bits
- **ARPA** (radio) : 2.032 bits
- **ARPANET** : 1.008 bits
- **ALOHANET** : 640 bits

☞ Cuando un paquete grande debe viajar por una red **cuyo tamaño máximo por unidad de transferencia** (MTU) es demasiado pequeño surgen nuevos problemas que se deben solucionar. Los tipos de soluciones propuestas son:

❋ Evitar que se presente la situación. La interred podría usar un algoritmo de encaminamiento que permitiese evitar pasar paquetes grandes por redes que no puedan manejarlos. Pero si la red destino del paquete es una de este tipo de redes es imposible evitarla. Se puede minimizar el problema pero no evitarlo totalmente.

❋ La solución real práctica casi única es que los gateways puedan partir los paquetes en fragmentos y enviarlos como paquetes separados. Sin embargo fragmentar es considerablemente más simple que desfragmentar. Dos técnicas: *Transparente* y *No Transparente*.

3.4. COMPARACION ENTRE GATEWAYS OC y SC (IV)

☺ VENTAJAS

Circuitos Virtuales Concatenados	DATAGRAMAS
Se puede prevenir la congestión por preasignación de *buffer*s en los gateways	Adaptación fácil ante problemas por congestión
Se garantiza el secuenciamiento	Puede usarse en redes que no tienen C.V.s
Se pueden usar cabeceras cortas	Robustos frente a fallos en los gateways
Se pueden evitar problemas causados por paquetes duplicados retrasados	Es posible usar una gran variedad de algoritmos de encaminamiento adaptativo

☹ INCONVENIENTES

Circuitos Virtuales Concatenados	DATAGRAMAS
Espacio reservado en las tablas G para cada conexión abierta aunque no se usen	Mayores encabezamientos
No hay rutas alternativas para hacer CdC	Mayor probabilidad de congestión
Vulnerabilidad ante fallos en los gateways	
Difícil de implementar si hay una red datagrama	

3.5. TIPOS DE FRAGMENTACION (I)

⌦ TRANSPARENTE

- La fragmentación en una red es transparente a las demás redes que deba atravesar. **El gateway que fragmenta envía todos los fragmentos a un mismo gateway destino y éste se encarga de componerlos de nuevo**. Las demás redes no perciben la fragmentación realizada.

☠ Es simple aunque problemática. El gateway destino debe saber cuándo tiene todos los fragmentos (usar contador o bit de fragmento final). Además todos siguen la misma ruta hacia el **G** destino y eso puede ser *ineficiente* y es posible que el gateway destino presente *bloqueo* por reensamblado. Un último problema es la *sobrecarga* de sucesivas fragmentaciones y composiciones al pasar por varias redes de pequeña MTU.

⌦ NO TRANSPARENTE

- **Sólo reensambla el *host* destino**. Los fragmentos se manejan como si fuesen paquetes originales. Una ventaja es que podrían utilizarse rutas a diferentes gateways de salida de la subred, aunque si el modelo usado es el de CVs concatenados esta mejora no puede utilizarse.

☠ Los problemas asociados son que se exige *que cualquier host pueda reensamblar* y además cada fragmento aumenta la *sobrecarga* por la aparición de nuevas cabeceras que permanecen todo el viaje del paquete.

3.5. ALGORITMOS DE FRAGMENTACION (II)

Shoch (1979)

- Cada paquete lleva un bit indicando si el destino es capaz o no de reensamblado. Si es así cada gateway puede elegir si usar fragmentación transparente o no. Si el destino no puede reensamblar cada gateway debe hacer que el siguiente gateway reensamble los fragmentos.

☞ La numeración de los fragmentos debe permitir su reensamblado. Soluciones:

Numeración en Árbol

- ⌑ El paquete X se fragmenta en trozos numerados X.0, X.1, etc... Si alguno se vuelve a fragmentar se le numera X.0.0, X.0.1, etc... Si hay espacio en la cabecera para el peor caso y no se generan duplicados esta solución asegura el reensamblado correcto (no importa orden de llegada).
- ☠ El problema es que si alguna red pierde o descarta paquetes la retransmisión del paquete grande original provocará efectos indeseados, sobre todo si llegaron sólo algunos fragmentos y la nueva fragmentación tiene lugar en una red diferente de aquella por la que viajó el paquete original (el nuevo X.0 puede no ser el anterior X.0 -!-).

MTU Elemental

- ⌑ Se elige un tamaño básico de MTU que pase por cualquiera de las interredes.
- ⌑ Al fragmentar todos los paquetes son de dicho tamaño menos el último que puede ser menor. Un paquete podría contener varios de estos fragmentos básicos por eficiencia de envío.
- ⌑ La cabecera podría contener el número del paquete original así como el número del primer fragmento contenido en el paquete. Debe existir un bit en el paquete indicando que el último fragmento elemental contenido en la trama es (o no) el último de los fragmentos originales.
- ⌑ Se requieren dos números de secuencia en la cabecera de los paquetes interred. En el límite un fragmento elemental es un bit o byte. El primer campo indica el número de datagrama original y el segundo el *offset* en él.
- ⌑ Algunos protocolos interred llevan este método más lejos y consideran toda la transmisión por un C.V. como un paquete gigante y cada fragmento contiene el número absoluto de byte del primer byte en el fragmento.

3.6. *SOFTWARE* PARA PUENTES Y PASARELAS

⌦ Muy distinto del usado en los *hosts* ordinarios. La eficiencia es imprescindible. Los B y G están limitados por CPU. Las interrupciones son rápidas pero dan lugar a *software mal estructurado*.

⌦ Lo ideal es planificar **procesos** que estarán *dormidos* esperando sobre colas de trabajos a realizar y cuyos resultados encolan en sus salidas, que serán a su vez de entrada a otros procesos. Cada proceso *tiene su propio espacio de direcciones*. Los procesos que atienden a las entradas deben tener la mayor prioridad posible para evitar perder paquetes de entrada cuando éstos llegan.

⌦ Los procesos de *salida* tienen prioridad intermedia y los de *encaminamiento* la menor prioridad. Estos últimos cogen paquetes y determinan su ruta poniéndolos en la cola de algún proceso de entre los de salida. *También filtra*: si el paquete es de **control** o bien su destino es la misma red que su origen no lo encola: lo ignora.

⌦ La orientación a procesos está bien estructurada pero es **lenta** por los cambios de contexto entre diferentes procesos. Un compromiso es que los procesos compartan el mismo espacio de direccionamiento al correr todos en modo kernel (núcleo). Para un SO de propósito general es imposible esto pero en un gateway se los supone bien definidos y vale la pena perder seguridad frente a las ganancias en velocidad y eficiencia.

⌦ El **algoritmo de planificación** en un gateway es crítico. Mejor que *round-robin* es dejar que los procesos terminen para evitar tener que salvar el contexto. El sistema sólo necesitaría una pila global (no una por proceso).

⌦ La ejecución de procesos por prioridad también puede ser cara y podría considerarse que al terminar cada proceso hiciese un ***polling*** de las interfaces de red y marcase los procesos que podrían ejecutarse a partir de ese momento. El planificador eligiría el de mayor prioridad para ejecutar en cada momento.

⌦ La **comunicación** entre procesos es también crítica. **Copiar mensajes** de una cola a otra es inaceptable. Deben pasarse *apuntadores*. El problema es que el proceso que los pasa no sabe cuándo liberar el *buffer* y es el receptor del mensaje el que debe liberarlos o reutilizarlos. Este proceso se agrava por el hecho de que los paquetes pueden cambiar de tamaño y además deben pasar por varias capas. Una solución es leer el paquete no desde el principio del *buffer*, sino a una distancia igual a la de la cabecera más larga posible.

⌦ *Problemas adicionales*: Temporizadores y cambios de orden en los bytes.