Programación Sobre Redes

T12: Conceptos básicos de redes

Nicolás Mastropasqua

October 19, 2020

Instituto Industrial Luis A. Huergo

Contenidos

1. Previamente
2. Comunicación entre procesos: Repaso
3. Introducción a redes
4. Arquiterctura de redes

Previamente

¿Dónde estamos parados?

- Mundo de la concurrencia y no-determinismo
  - Procesos
  - Threads
  - Race conditions
  - Sincronización
    - Algoritmos de sincronización
    - Semáforos
    - Objetos atómicos
    - Esquemas de sincronización (mutex, barreras, etc)

Comunicación entre procesos: Repaso

Memoria compartida vs Envío de mensajes

- El modelo de mensajes es interesante a la hora de intercambiar poca información, ya que no incurre en conflictos de coherencia y puede ser más fácil de implementar.
- El modelo de memoria compartida permite mayor velocidad, ya que no tiene el overhead provocado por el manejo de mensajes y se puede trabajar, en general, a la velocidad que lo permita la memoria (si son procesos locales). Sin embargo, es necesario una complejidad adicional para sincronizar los procesos y proteger los datos. Cómo se logra esto?

Introducción a redes

Conectividad

La conectividad en las redes se da en distintos niveles. En su nivel mas bajo, y abstracto, tenemos.

Links y Nodos

Consideremos dos o más computadoras conectados directamente por algún medio físico, como un cable coaxil, UTP o fibra óptica. Llamamos **link** a dicho medio y **nodos** a las computadoras que conecta.

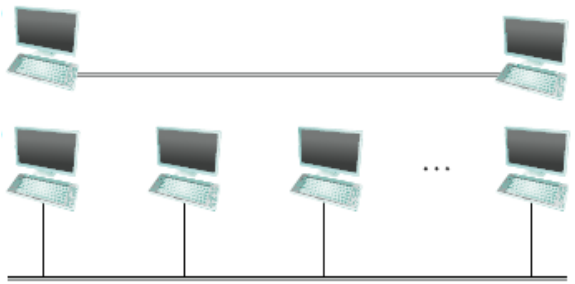

Figure 1: Point to Point y Multiple access

Pregunta: Es escalable este diseño? No! Necesitamos algún tipo de conexión indirecta entre nodos...

Switched networks

Circuit-Switched networks

Se establece un circuito dedicado entre ambos nodos. Luego, el emisor puede enviar stream de bits por dicho circuito hacia el nodo destino.
Se garantiza el total del **ancho de banda** del link.

Packet-Switched networks

Los nodos mandan bloques de datos discretos llamados paquetes o mensajes. Utilizan una estrategia llamada **store-forward**. Cada nodo que recibe un paquete por medio de un link, lo guarda en su memoria intera y lo forwardea al próximo nodo.

Packet-Switched network

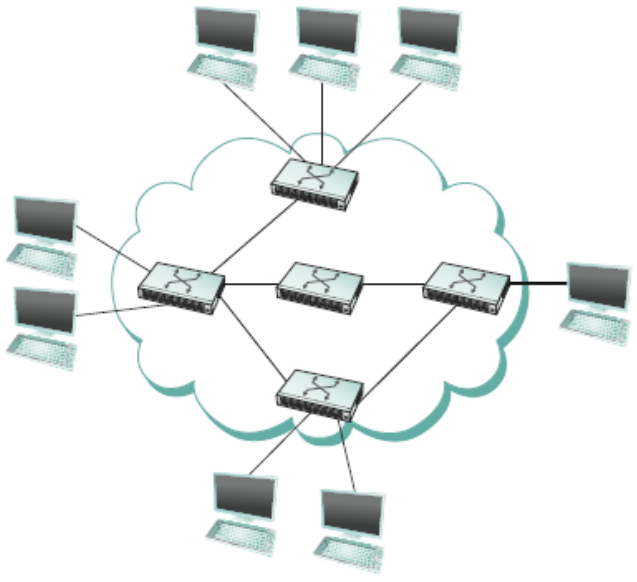

Figure 2: Los nodos fuera de la nube, **hosts**, ”usan” la nube, mientras que los de adentro, **switches**, hacen store-forward únicamente

Red de redes

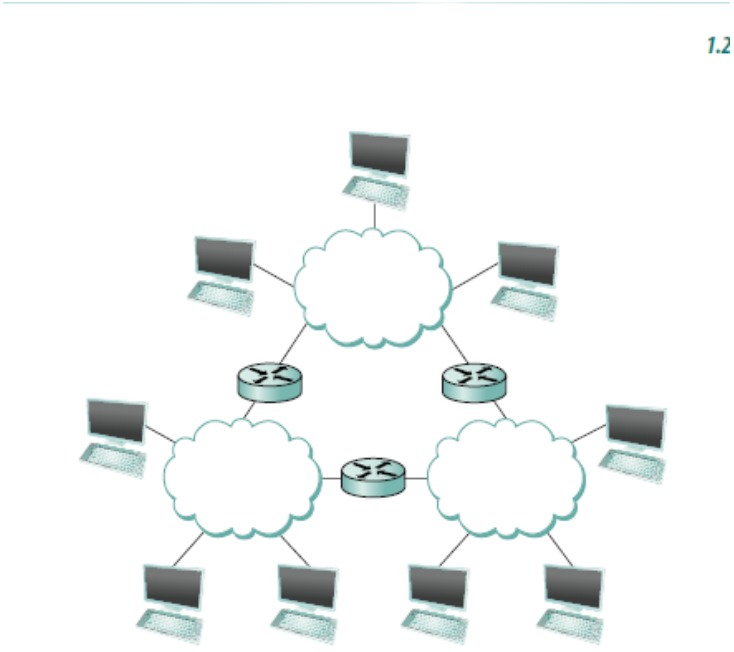

Figure 3: Los nodos que es´tan conectados a una o mas redes se los denomina router o **gateway** y actúan de forma similar a un switch

A su vez, cada red de redes es una red. Por lo tanto puede conectarse a otra red. El proceso puede continuar recursivamente.

Resumen

Red

Podemos definirla recursivamente.
Dos o mas nodos conectados por un link (o switches intermedios) forman una red. El resultado de conectar dos redes por una o más nodos, también es una red.

Importante:

- Para tener conectividad host to host, necesitamos que cada nodo tenga una dirección que lo identifique en la red Si no están conectados directamente, los switches y los routers usan esta dirección para decidir como forwardear el mensaje para que llegue a destino. A esto le decimos routing.

Packet switching

La intención de la red es permitir que todos los nodos alcancen a cualquier otro a través del envío de mensajes.
Pregunta: Cómo hacen los nodos para comunicarse al mismo tiempo?
Más aún, ¿Como hacen para compartir el mismo link si todos quieren usarlo al mismo tiempo?

Multiplexing

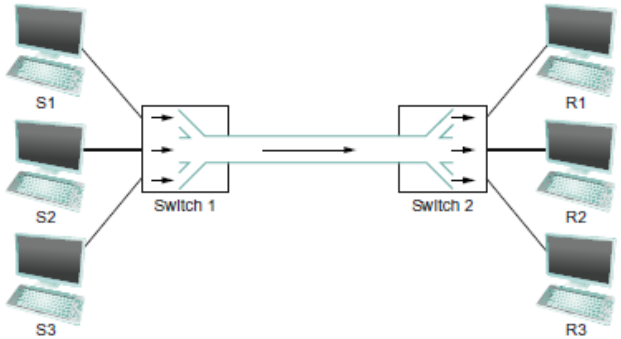

Figure 4: Multiplexando varias comunicaciones por un único link

STDM (Synchronous time-division multiplexing

Se usan quantums con Round Robin

FDM(Frequency-division multiplexing)

Cada host utiliza una frecuencia distinta

Statistical multiplexing

Si alguno de los host está idle durante su quantum, estamos desperdiciando ancho de banda. En los anteriores, el número máximo de comunicaciones está fijo de antemano.

Statstical multiplexing

Similar a STDM: El link se comparte a lo largo del timepo

Distinto a STDM: Los datos se transmiten on-demand en lugar de utilizar un quantum

Pregunta: Cómo nos aseguramos que todos los host puedan tomar su turno y que ninguno envía un mensaje arbitrariamente largo?

Paquetes

Cada host tiene permitido enviar una cantidad máxima de datos. Este bloque finito se denomina **paquete**, y se distingue de un **mensaje**, arbitrariamente largo, que puede ser enviado por una aplicación

Multiplexing packets

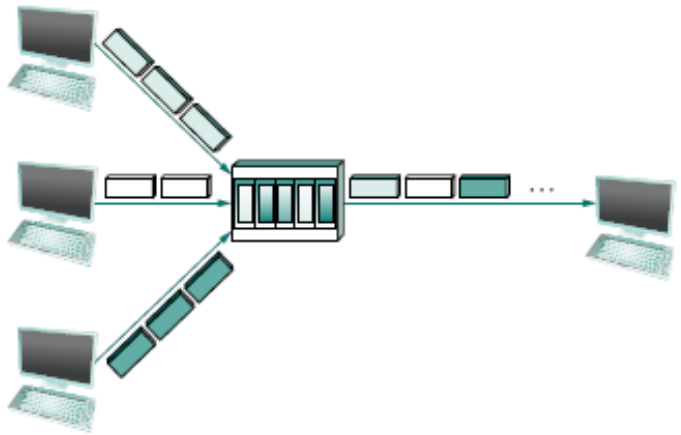

Figure 5: Un switch multiplexando paquetes de multiples hosts a un único link compartido

Pregunta:¿Qué pasa si hay diferencia de velocidades entre lo que entra y lo que sale? Se dropean paquetes. En este caso, el switch se dice congestionado

Abstracción:Canales

- Hasta ahora, hablamos de comunicación entre hosts.
- A más alto nivel, nos interesa que dos programas, corriendo en host distintos, puedan comunicarse.
- Por suerte, no hay que preocuparse de toda la complejidad subyacente y podemos abstraernos utilizando servicios provistos por la red
- Podemos ver la red como una conjunto de canales que proveen servicios utilizados por las aplicaciones para comunicarse

Abstracción: Canales

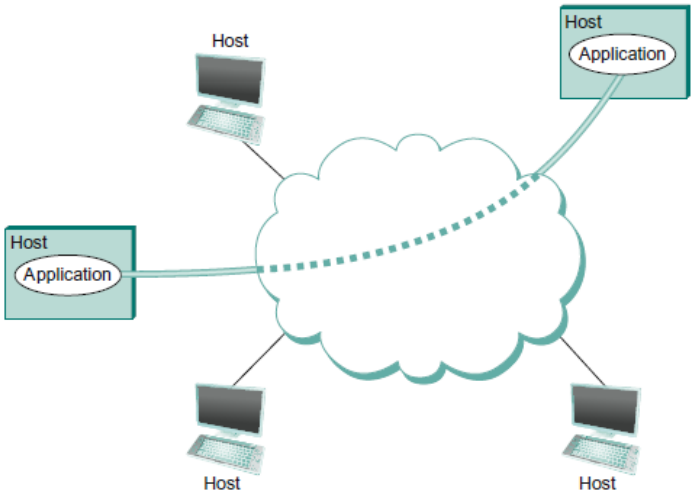

Figure 6: Comunicación de procesos a través de canales abstractos

Para que la abstracción tenga sentido, hay que poder determinar que funcionalidades provee un canal a una aplicación.

- Hay garantías de que el mensaje siempre se envía?
- Es aceptable que se pierdan algunos paquetes?
- Deben los mensajes llegar en orden?
- Es importante que nadie pueda ”escuchar” sobre el canal?

Patrones de comunicación

Figure 7: Distintos requerimientos de comunicación

Request-Reply Channel

Garantía de entrega de mensajes

Solo una copia del mensaje se entrega

Proteger la privacidad e integridad del mensaje

Message-Stream Channel

No garantía de entrega de todos los menajes

Mensajes se reciben en el orden que se enviaron

Proteger la privacidad e integridad del mensaje

Multicast

Arquiterctura de redes

Capas

Arquitectura por capas

- Descompone el problema original en componentes más manejables
- Moduluarización
- Abstracción

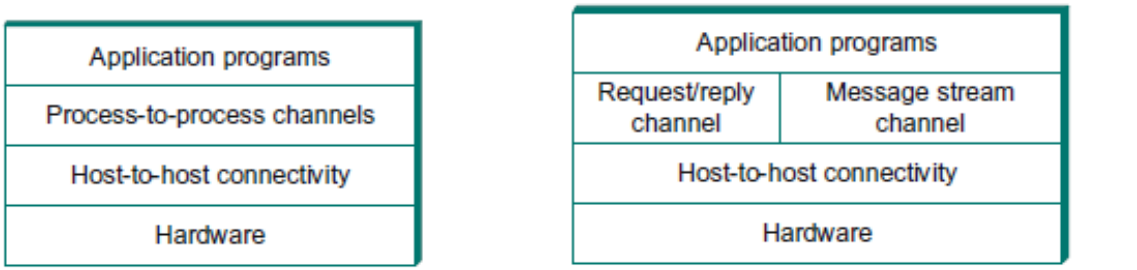

Figure 8: Ejemplo de red por capas

Protocolo

- Un protocolo provee los servicios de comunicación que los objetos de más alto nivel, como las aplicaciones o, incluso, protocolos de capas superiores), utilizas para enviar mensajes.
- Por ejemplo, podemos definir una red que soporte un protocolo request/reply o message stream.
- Cada protocolo define dos interfaces: Service interface y Peer interface
- Los objetos de orden superior utilizaran al protocolo invocando funciones de la Service Interface
- El protocolo se comunica con su contraparte utilizando su Peer interface

Interface

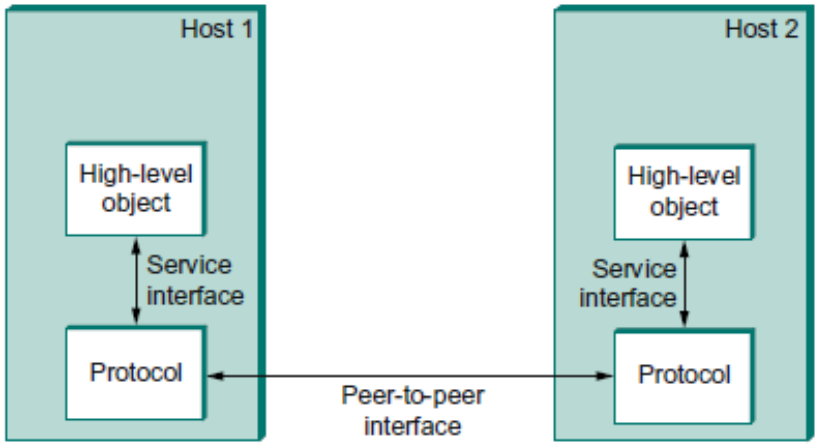

Figure 9: Service and peer Interfaces

Grafos de protocolos

No necesariamente hay un único protocolo por capa!

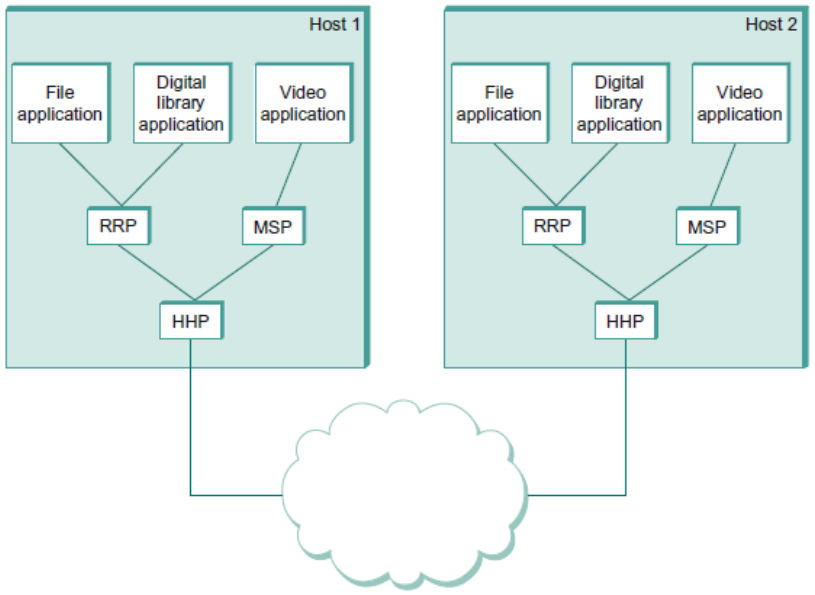

Figure 10: Service and peer Interfaces

Ojo! Distinguimos la especificación de un protocolo de la implementación.

Caso interesante, donde la especificación se probo ”segura” pero luego apareció un ataque:

Encapsulamiento: Payloads, Headers y Protocol Stack

Si bien un protocolo se comunica con su crontraparte utilizando servicios de la peer interface, esto último tiene que pasar por el resto de las capas.

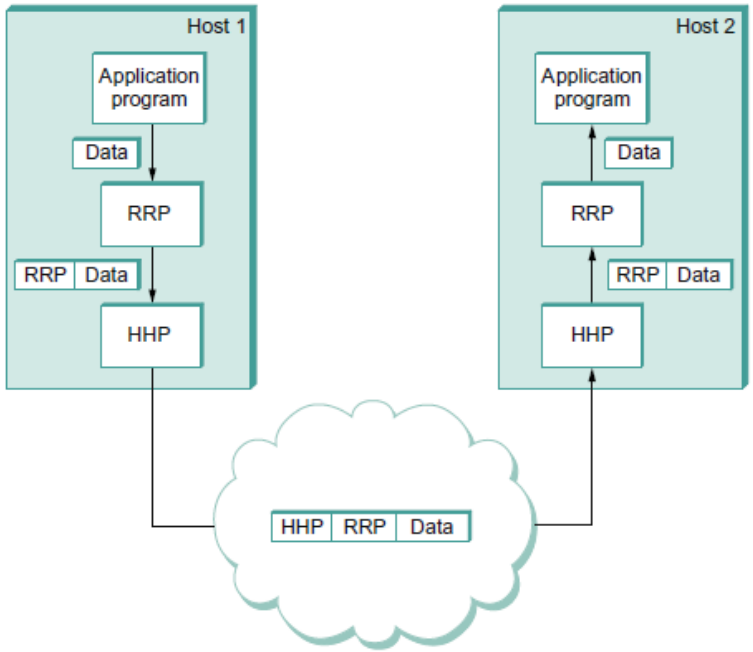

Figure 11: Encapsulamiento de mensajes de alto nivel a bajo nivel

Demultiplexación de paquetes

Dado un programa *p* que quiere comunicarse con otro programa q en un host diferente a través de un canal. ¿Cómo determina el host *q* a que aplicación debe entregar el paquete?

Demux key

El origen agrega al header una demux key.

El destino revisa el header y utilizando la demux key, demultiplexa el mensaje a la aplicación correcta

Arquitectura Internet

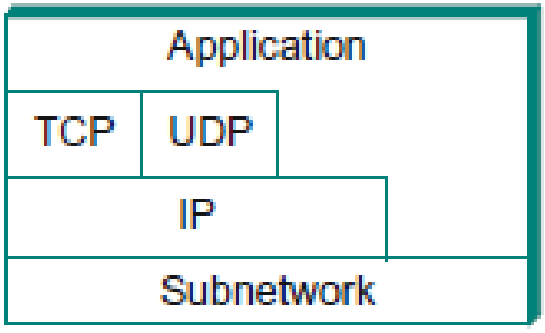

Figure 12: Arquitectura internet en capas

OSI

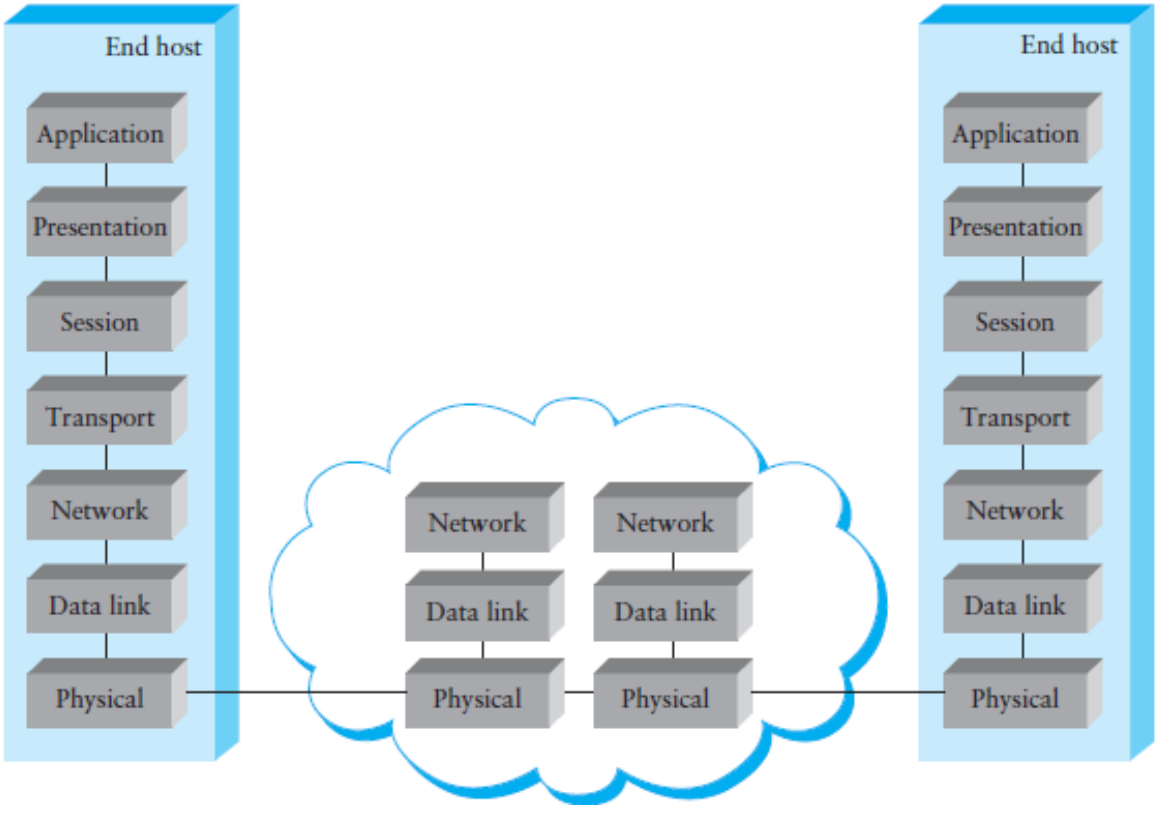

Figure 13: Arquitectura OSI