Sistemas Distribuidos
Comunicación

Prof. Lic. R. Alejandro Mansilla
Prof. Ing. Sebastián Cardello

Licenciatura en Ciencias de la Computación
Facultad de Ingeniería
Universidad Nacional de Cuyo

Fundamentos

La comunicación entre procesos es la base de funcionamiento de todo sistema distribuido

Modelo de capas: Modelo de referencia OSI

- Diseñado para permitir la comunicación de sistemas abiertos
- cada capa maneja un aspecto específico de la comunicación
- cada capa le presta servicios a la superior y consume servicios de la inferior
- Cada capa posee un encabezado y un espacio de carga útil o payload que permite el encapsulamiento

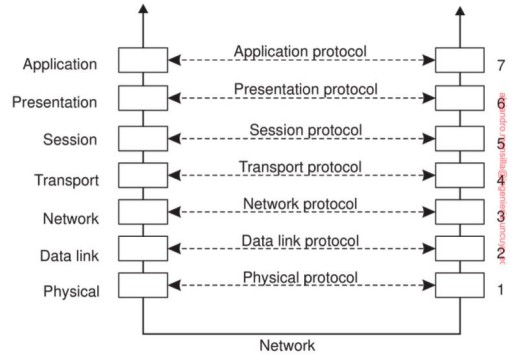


Figure 4.1: Layers, interfaces, and protocols in the OSI model.

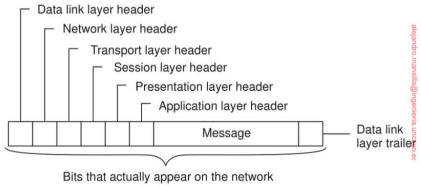


Figure 4.2: A typical message as it appears on the network.

Una versión adaptada del modelo OSI

- Protocolos de aplicación: Ej. HTTP
- Protocolos de middleware: son de uso común y generalizado para las viabilizar aplicaciones. Ej. DNS
- Protocolos a nivel de host: facilitan la comunicación entre máquinas identificables, no necesariamente en el mismo espacio físico. Ej. TCP/IP
- Protocolos a nivel de HW: Ej. Ethernet

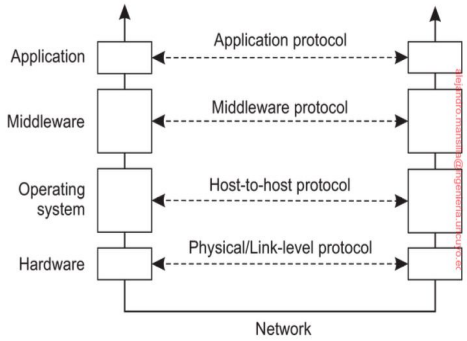


Figure 4.3: An adapted reference model for networked communication.

Tipos de comunicación

- Comunicación persistente: el mensaje se persiste hasta ser recibido. Ej. email / discord
- Comunicación transitoria: el mensaje no se persiste y si descarta si no puede ser entregado. Ej. HTTP.

  además

  - Comunicación asíncrona: el remitente no se bloquea y sigue con la ejecución (Ej. AJAX)
  - Comunicación síncrona: el remitente es bloqueado hasta que la petición fue aceptada (Ej. RPC)

  adicionalmente

  - Discreta: cada mensaje forma una unidad completa de información. Ej. HTTP..
  - por flujos: involucra el envío de varios mensajes, uno después de otro, y los mensajes se relacionan entre sí por el orden en que se envían, o porque existe una relación temporal.. Ej. WebSocket.

RPC *(remote procedure call)*

Cuando un proceso de la máquina A llama a un procedimiento de la máquina B, el proceso que llama desde A se suspende, y la ejecución del procedimiento llamado ocurre en B.

La información puede transportarse en los parámetros desde quien llama hasta el que es llamado, y puede regresar en el procedimiento resultante.

Ningún mensaje de paso es visible para el programador. Este método se conoce como llamada a procedimiento remoto, o simplemente RPC.

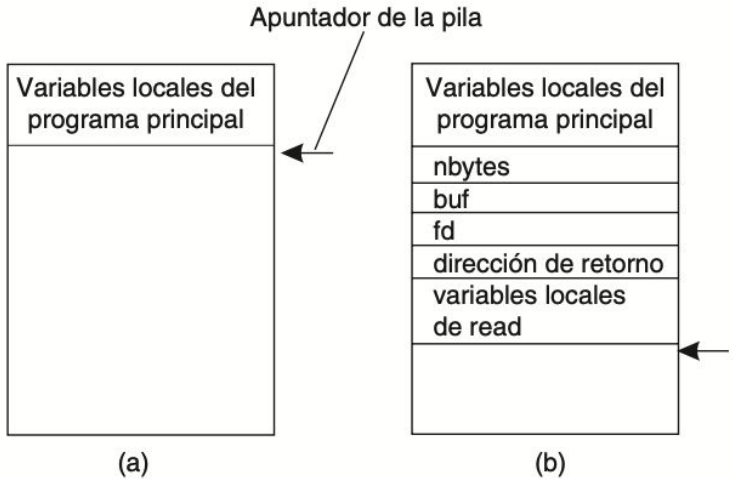
L(ocal)PC: Operación básica

Una llamada a un procedimiento local:

```
count = read(fd, buf, nbytes);
```

Luego de que read termina, coloca el resultado en un registro y elimina la dirección de retorno

Si los parámetros se pasan por valor, se copian a la pila en cambio si se pasa por referencia, solo se coloca en la pila un puntero.

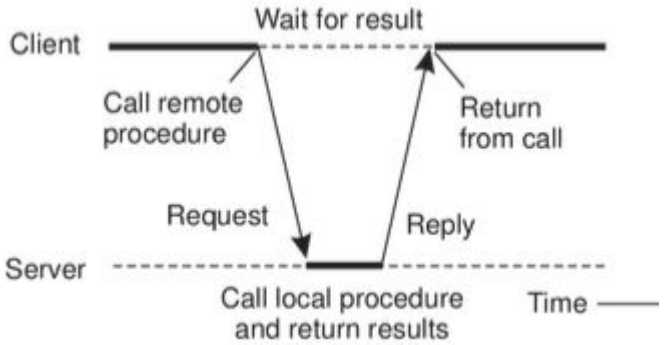


RPC: Operación básica

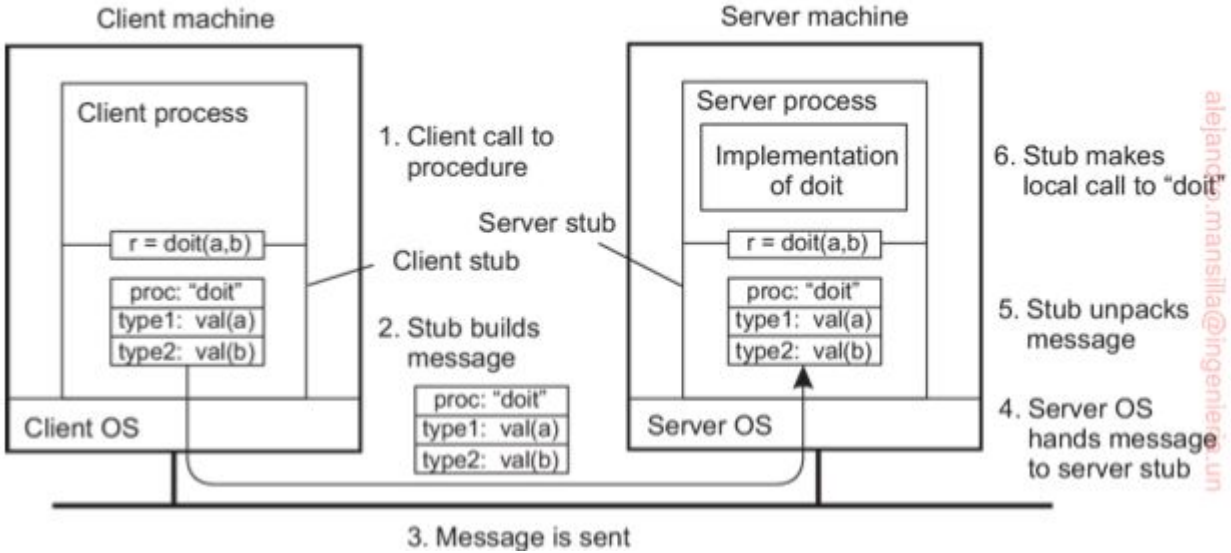
Una llamada RPC deber **parecer local**, y por lo tanto **transparente**: el llamado no debe enterarse que el llamado está siendo ejecutado en otra computadora

En rpc aparece el concepto de **Stub**, que es una copia ofrecida al cliente del procedimiento a ejecutar en forma remota. Que NO hace lo que debería, sino que empaqueta los parámetros y los envía al servidor. Cuando el mensaje llega al servidor éste llama a un Server Stub, que es la contraparte del client stub del lado del servidor.

El server stub desempaqueta el mensaje y ejecuta localmente (al servidor). Cuando recibe los resultados, los empaqueta nuevamente y los envía al client stub quien desbloquea el proceso.

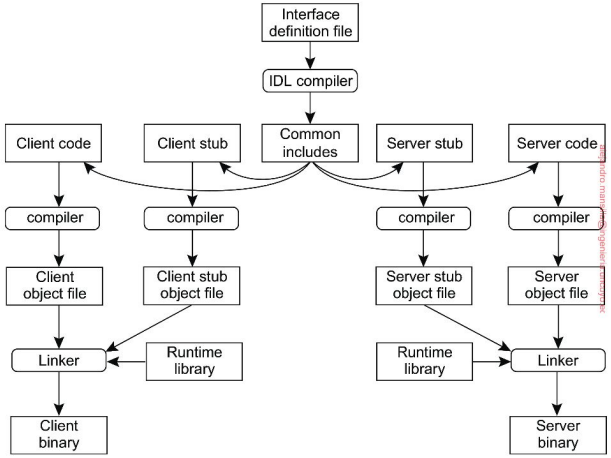


RPC: en resumen



Interface Definition Languages (IDL)

- Son lenguajes utilizados para describir interfaces
- Permiten la generación automática de los stubs, inclusive entre diferentes lenguajes (cross-platform)
- Facilitan el versionamiento y evolución de dichas definiciones.
- Ej. Protocol Buffer y gRPC




```
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

A proto definition.

```
// Java code
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

Using a generated class to persist data.

```
// C++ code
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

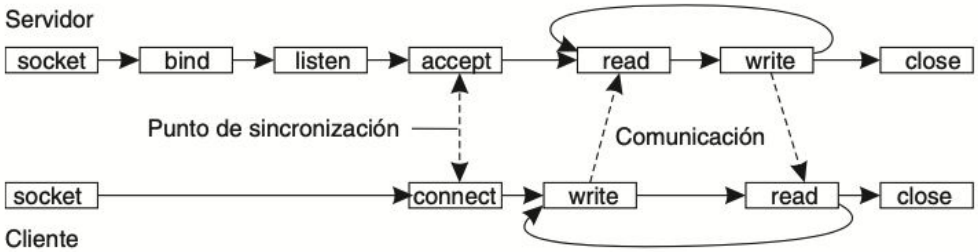
Using a generated class to parse persisted data.

Comunicación transitoria orientada a mensajes

Sockets Berkeley

"Es un punto final de comunicación en el que una aplicación puede escribir información destinada a enviarse fuera de la red subyacente, y desde el cual puede leerse información entrante"

Primitiva	Significado
Socket	Crea un nuevo punto final de comunicación
Bind	Asocia una dirección local a un socket
Listen	Anuncia la conveniencia de aceptar conexiones
Accept	Bloquea a quien llama hasta que llega una petición de conexión
Connect	Activa el intento de establecer una conexión
Send	Envía algunos datos a través de la conexión
Receive	Recibe algunos datos a través de la conexión
Close	Libera la conexión



Comunicación transitoria orientada a mensajes

MPI *(Message passing Interface)*

- Evolución natural de sockets, más específico e independiente del hardware y la plataforma
- Diseñada para aplicaciones paralelas y comunicaciones transitorias
- Utiliza directamente la red subyacente y no intenta recuperar automáticamente las fallas fatales
- Las comunicaciones suceden dentro de un grupo conocido de procesos
- La dupla **GroupID, ProccessID** identifica la fuente o destino de un mensaje y reemplaza las direcciones a nivel de transporte

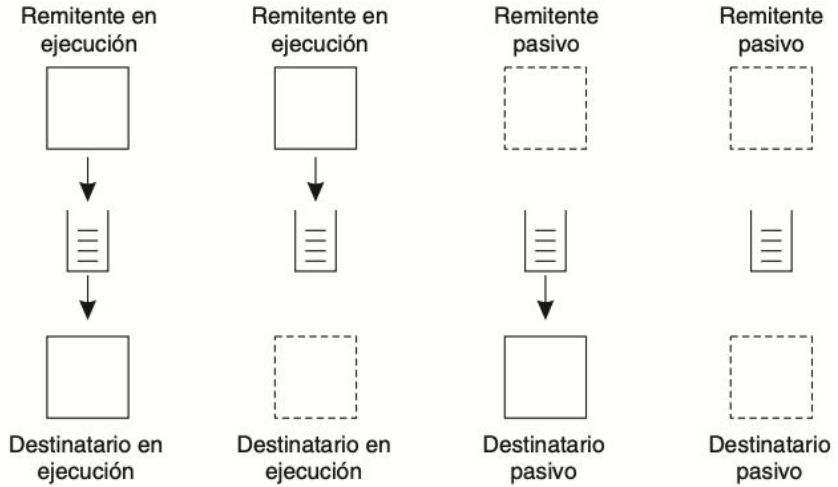
Comunicación persistente orientada a mensajes

Middleware Orientado a Mensajes o **MOM**

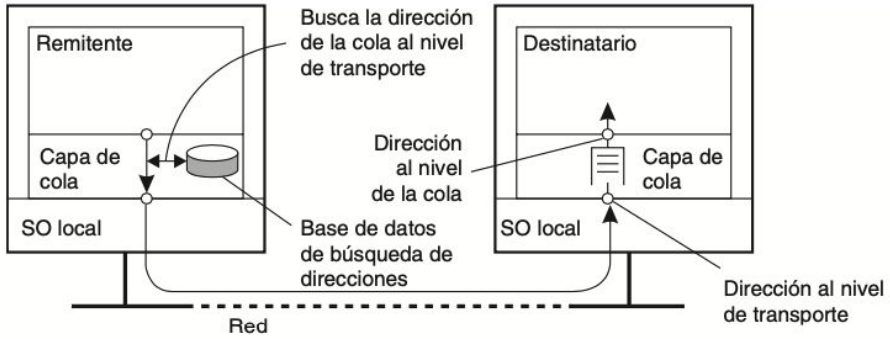
- También conocidos como sistemas de colas de mensajes
- Proporcionan amplio soporte para comunicaciones asíncronas persistentes
- Ofrecen almacenamiento de término medio para mensajes
- Las transferencias de mensajes pueden tomar minutos en lugar de segundos o milisengudos como en Sockets o MPI

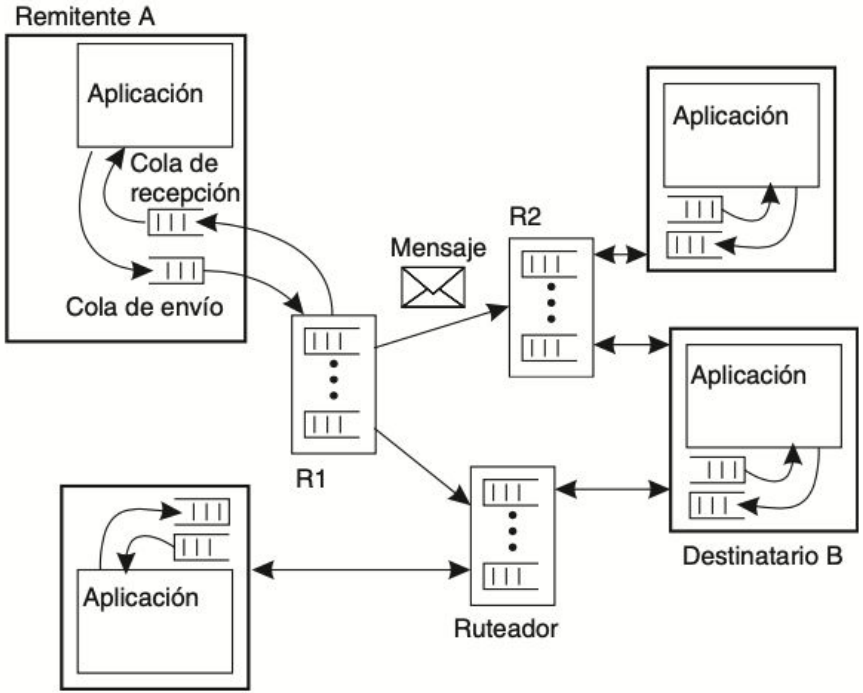
Modelo de colas de mensajes

- Las aplicaciones se comunican insertando mensajes en colas locales específicas
- Cada aplicación tiene su cola de mensajes privada aunque también pueden ser compartidas
- Estos sistemas por lo general no dan garantía de entrega ni lectura de los mensajes, solo que fue agregado a la cola correspondiente.
- Poco acoplamiento en la comunicación
- Los mensajes pueden contener cualquier información
- Se puede entender el email como este modelo*



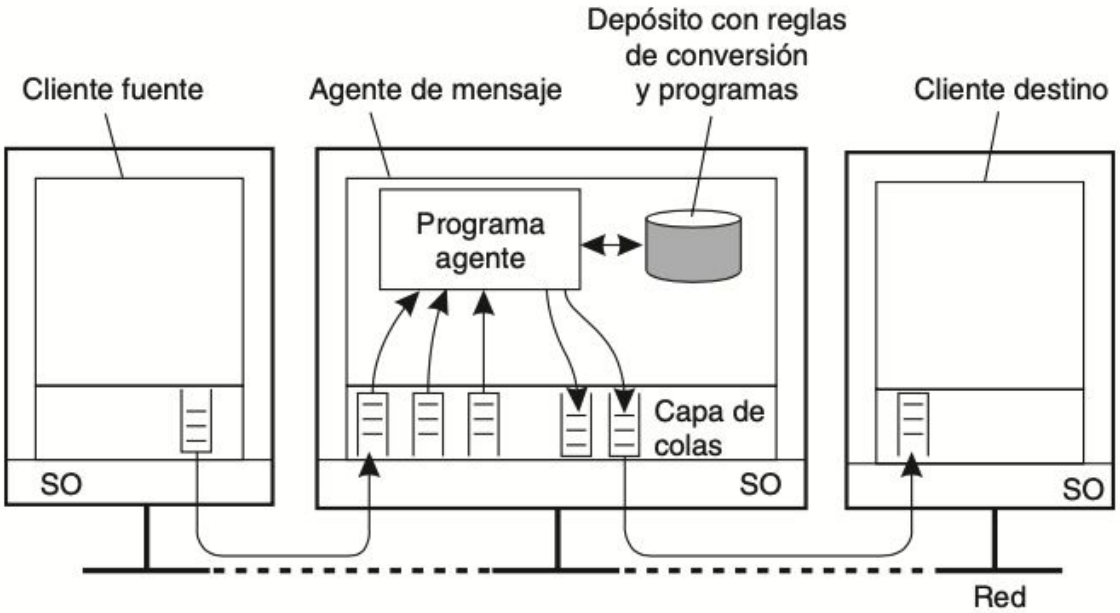
Arquitectura general de un sistema de colas





¿Cómo se mantienen las tablas de direccionamiento cuando se agregan o modifican los nodos?
¿Que pasa si los nodos deben escalar de forma elástica?

Agentes *(brokers)* de mensajes



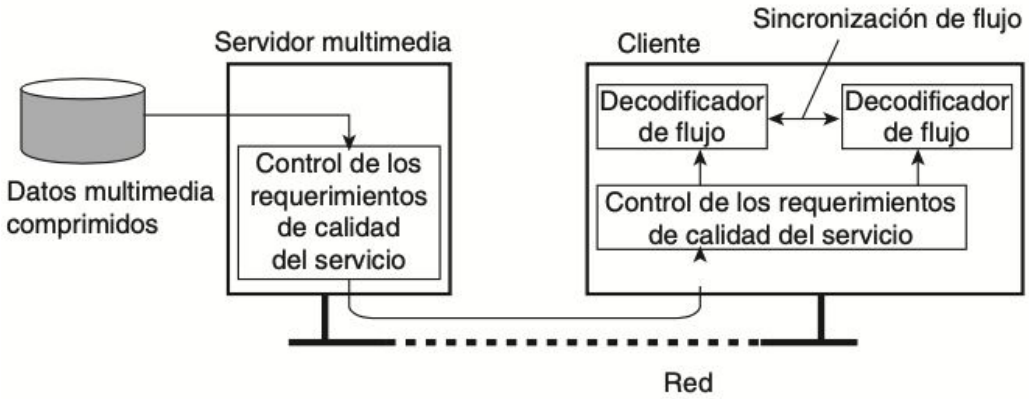
Algunos ejemplos:

- RabbitMQ
- Kafka
- Redis
- ActiveMQ
- Kestrel
- Mosquitto (Mqtt)

Patrones de comunicación orientados a mensajes

- **Request-Reply Pattern**: sigue el modelo de cliente-servidor. Un cliente emite un mensaje y espera una respuesta (síncrona o asíncrona) del servidor de dicha petición. En esencia RPC.
- **Publish-Subscribe Pattern**: Un servidor publica mensajes los cuales son solo recibidos por aquellos clientes que se suscriben.
- **Pipeline Pattern**: Un servidor "expulsa" (*push out*) un mensaje esperando que *cualquier* cliente pueda "jalarlo" (*pull in*). La diferencia con el patrón pub-sub es que el mensaje se procesa una vez independientemente de cliente realice el trabajo.

Comunicación Orientada a flujos



(Estudiar del Libro de Tanenbaum en español, página 157 en adelante)

Comunicación por multitransmisión (multicasting)

- Brinda soporte a enviar datos a varios destinatarios
- Son ampliamente utilizados dentro del dominio de los protocolos de red (OSI 2 y 3)
- A nivel de aplicación,
  - son modelos de comunicación orientados a sistemas distribuidos uniformemente (peer-to-peer).
  - hay diferentes modelos para diseminar la información que difieren en la eficiencia y la topología de las redes subyacente.
    - Por jerarquía. Una jerarquía predefinida (ej de árbol) permite diseminar información de forma eficiente.
    - Por Inundación. Todos los nodos comparten con todos los datos deteniéndose solamente cuando el destinatario ya conoce el mismo
    - Por muestro. (gossiping/epidemic protocol). Se distribuye información siguiendo un patrón de diseminación estadística.