Tema 6: Servicios de Mensajería Interactiva

Ingeniería de Servicios

2023-2024

Tipos de servicios interactivos

- Mensajes de texto "en tiempo real":
  - IRC (*Internet Relay Chat*)
  - Los diferentes *messenger* (~~Microsoft Messenger~~, Yahoo Messenger, AOL AIM, Jabber)
  - Servicios de chat basados en web (Facebook chat, MSN, ~~Google Talk~~)
  - Para dispositivos móviles (Whatsapp, Google Chat, iMessage, Telegram)
  - Muchos de ellos permiten chat en grupo, canales de difusión, etc.
- Interacciones multimedia:
  - Voz sobre IP y videoconferencia (Microsoft Teams, Google Meet, Zoom, etc., ~~Skype~~, ~~Google Talk~~, ~~Microsoft Lync~~)

Qué tienen en común

- Detección de presencia (¿está el usuario disponible?)
- Inicio de sesión (conectar con el otro interlocutor y establecer parámetros de la comunicación)
- Transmisión de los datos.
  - Texto: protocolos basados en texto
  - Audio o video: *streaming* sobre RTP

Protocolos y estándares

- `SIP`: *Session Initiation Protocol* define mecanismos para iniciar y finalizar sesiones interactivas multimedia.
  - Protocolo ASCII (tipo HTTP o SMTP)
  - Usa SDP para definir los parámetros de los *streams* y sus *endpoints*.
  - Asume el uso de RTP para la transmisión de los flujos multimedia
  - Puede ir sobre TCP ó UDP y admite TLS
- `SIMPLE`: *Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions*
  - Extiende SIP para mensajería instantánea (de texto)
  - Y mecanismos SUBSCRIBE/NOTIFY para la presencia

Protocolos y estándares

- `XMPP`: *eXtensible Messaging and Presence Protocol*
- Incorpora sub-protocolos para
  - Presencia
  - Inicio de sesión
  - Mensajería

El resto de este tema se centra en XMPP

XMPP

Introducción

- Finales de los 90: muchos sistemas de mensajería incompatibles entre sí.
- 1999: Grupo de trabajo IMPP (*Instant Messaging and Presence Protocol*) intenta un estándar (recomendaciones)
- 1999: Jeremie Miller programa Jabber, (código abierto, descentralizado)
- 2004: Jabber, rebautizado como XMPP, es aprobado como estándar (compatible con IMPP)
- 2005: Google Talk adopta XMPP
- 2010: Facebook chat, usa XMPP (pero incompleto)
- Más servicios basados en XMPP pero no del todo compatibles: Skype, Whatsapp, Microsoft Messenger
- 2013: Google Hangouts, abandona XMPP
- Actualidad: de nuevo muchos sistemas de mensajería incompatibles entre sí, pero influidos por XMPP
- XMPP sigue existiendo y en desarrollo (derivando hacia IoT)

Introducción: X de eXtensible

- XMPP es un protocolo "modular"
- Presenta una funcionalidad "*core*" para mensajería instantánea entre dos personas
- Y un gran número de extensiones (XEPs) para otros usos:
  - Notificación de actividad en chat ("Escribiendo...")
  - Salas de chat en grupo (tipo IRC)
  - Mecanismos SUSCRIBE/NOTIFY genéricos
  - Voz sobre IP (esta extensión se denomina *Jingle*)
  - Comunicación sobre HTTP lo que permite su uso en páginas web

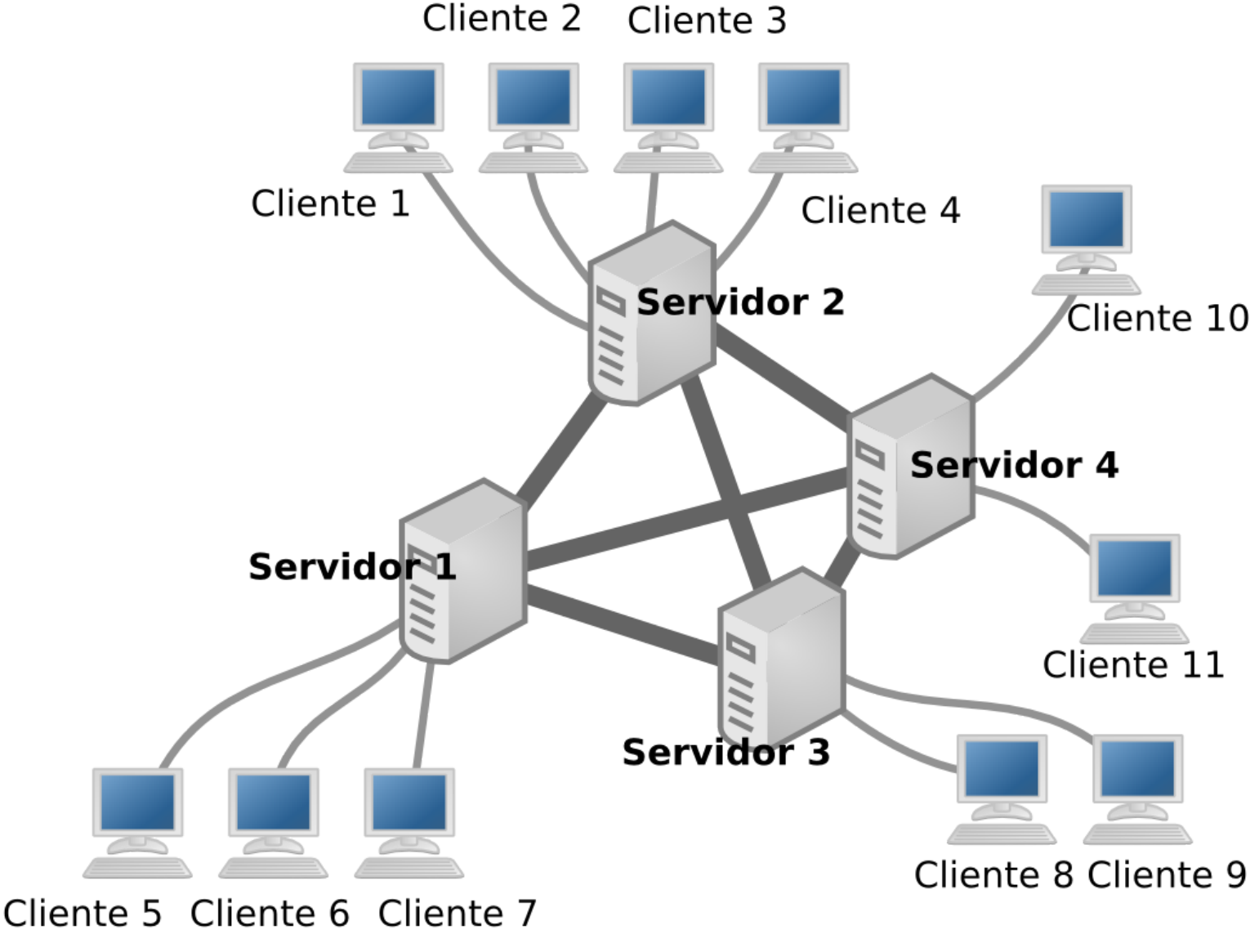
Introducción: XMPP es asíncrono

Comparación:

- HTTP
  - Las conexiones TCP tiene corta vida
  - El cliente siempre inicia las comunicaciones
  - Mensajes basados en "cabeceras/cuerpo"
  - El servidor no mantiene estado
  - Si el cliente quiere saber cuándo hay cambios, debe preguntar de nuevo (*polling*)
- XMPP
  - Mantiene una conexión TCP permanente cliente-servidor
  - Cliente o servidor pueden enviar mensajes por esa conexión en cualquier momento
  - Mensajes en XML
  - El servidor mantiene estado
  - El servidor notifica al cliente cuándo hay cambios (*push*)

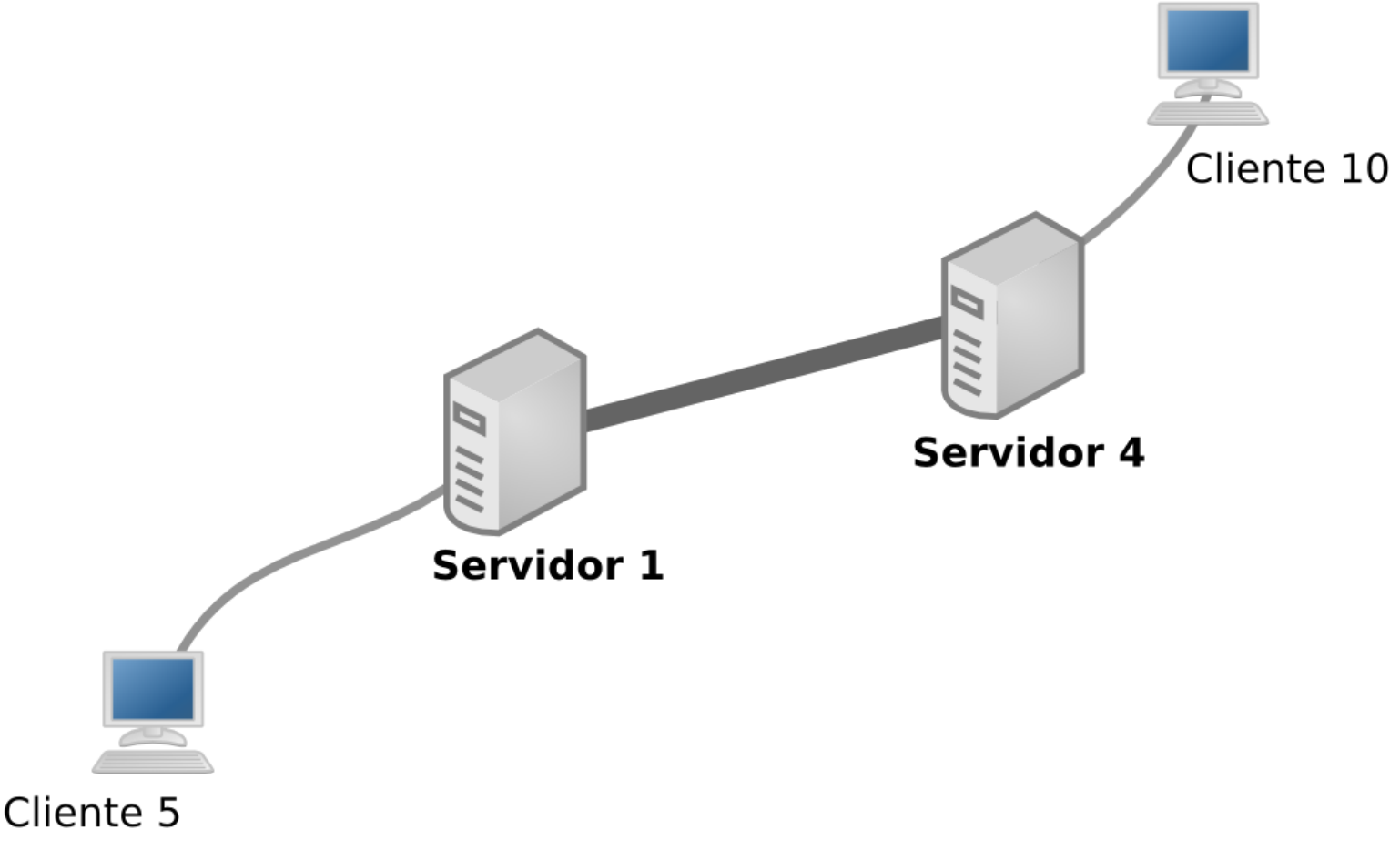
Introducción: Arquitectura

La red XMPP es *federada*: varios servidores conectados entre sí, y a los que se conectan los clientes.



Introducción: Arquitectura

Pero la comunicación es *single-hop*, esto es, el mensaje entre dos clientes sólo atraviesa dos servidores como máximo (a diferencia por ejemplo de SMTP).



Introducción: Jabber ID (JID)

Cada usuario tiene un identificador único con esta estructura:

```
usuario@dominio.servidor/recurso
```

Ej: `manolo@jabber.org/tablet`

Cuando no se especifica el recurso, se denomina *bare JID* (Ej: `manolo@jabber.org`), cuando se especifica se llama *full JID*.

Un mensaje puede ir dirigido a:

- Un *full JID* -> Se intentará entregar en el recurso correspondiente
- Un *bare JID* -> Se entregará en alguno de sus recursos asociados

Introducción: El *roster*

Es como XMPP denomina a la "lista de contactos".

- Mantiene una lista de JIDs de los que somos "amigos"
- El cliente XMPP generalmente nos muestra cuáles de ellos están *online* (más sobre esto después)
- Está alojada en el servidor
- Una parte del protocolo se dedica a recuperar el *roster* o actualizar sus datos.

Protocolo

Estructura de los mensajes

La comunicación es un *stream* de XML que pueden llegar en cualquier momento.

> Esto implica programación **orientada a eventos**

Cada "fragmento" XML que se recibe se denomina una *stanza* (estrofa).

Hay sólo tres tipos de *stanzas*:

- `presence` (señala si el usuario está "en línea" y disponible)
- `message` (transporta un mensaje a un JID)
- `iq` (solicita información o transporta la respuesta a otro `iq`)

Protocolo

Ejemplo de un "diálogo" entre cliente y servidor

```
C: <stream:stream>
C:   <presence/>
C:   <iq type="get">
       <query xmlns="jabber:iq:roster"/>
     </iq>
S:   <iq type="result">
       <query xmlns="jabber:iq:roster">
         <item jid="alice@wonderland.lit"/>
         <item jid="madhatter@wonderland.lit"/>
         <item jid="whiterabbit@wonderland.lit"/>
       </query>
     </iq>
C:   <message from="queen@wonderland.lit"
              to="madhatter@wonderland.lit">
       <body>Off with his head!</body>
     </message>
S:   <message from="king@wonderland.lit"
              to="party@conference.wonderland.lit">
       <body>You are all pardoned.</body>
     </message>
C:   <presence type="unavailable"/>
C: </stream:stream>
```

Protocolo

Presencia

Las notificaciones de presencia siguen un modelo pub/sub.

- **A** desea conocer cuándo **B** está disponible.
- A se "suscribe" a la presencia de B
  - Su cliente envía

```
    <presence from="A" to="B" type="subscribe"/>
```

- B debe autorizarlo
  - Cuando B lo autoriza, A recibe

```
    <presence from="B" to="A" type="subscribed"/>
```

- El estado de la suscripción queda almacenado en el servidor

Habitualmente también B se suscribe a la presencia de B, pero no es necesario.

Protocolo

Presencia: Notificación

Cuando el usuario B se conecta, su cliente emite la *stanza* más simple posible:

```
<presence/>
```

Que indica "estoy online". Todos los usuarios suscritos a la presencia de este usuario, recibirán una *stanza* como esta:

```
<presence from="B" to="A">
```

Donde `"B"` sería realmente el *full JID* del usuario B, lo que incluye el nombre del recurso desde el que se ha conectado.

Protocolo

Presencia: Disponibilidad

La *stanza* de presencia puede contener información más detallada sobre el estado de un usuario, pues puede contener los tags `<show>` y `<status>`, por ejemplo:

```
<presence>
  <show>away</show>
  <status>Tomando un café</status>
</presence>
```

Si bien en `status` puede ir cualquier mensaje que el usuario haya preparado, en `show` en cambio sólo puede ir uno de los siguientes:

- `chat` Disponible para charlar
- `away` Ausente (no interactúa con el dispositivo)
- `xa` (*eXtended Away*) Hace mucho tiempo que no interactúa
- `dnd` (*Do Not Disturb*) ocupado, no desea charlar

Protocolo

Presencia: Prioridades

- XMPP prevé que el usuario pueda estar conectado desde varios dispositivos a la vez.
- La *stanza* `<presence>` puede incluir el tag `<priority>`
  - Es un valor entre -127 y 128
  - Los recursos con prioridad negativa no recibirán mensajes
  - Los de prioridad positiva recibirán mensajes según su prioridad

> Un servidor XMPP usa el valor de prioridad cuando tiene que enviar un mensaje a un destinatario y el campo `to=` sólo especifica el *bare JID*.

Protocolo

Presencia: Extensiones relacionadas con presencia

Existen múltiples extensiones para incluir información más rica en la *stanza* `<presence>`, como por ejemplo:

- Las coordenadas GPS del contacto
- Información sobre qué música está escuchando
- etc.

De hecho a veces se usa la *presencia* para implementar la difusión de otra información de tipo *publisher/subscriber*, aunque no debería usarse para esto ya que XMPP tiene otras extensiones específicas.

Protocolo

Mensajes

La *stanza* `<message>` llava los siguientes atributos:

- `id=` Identificador único del mensaje, generado algorítmicamente
- `from=` *Full JID* del emisor del mensaje
- `to=` JID (*bare* o *full*) del destinatario
- `type=` Uno de los siguientes valores:
  - `"normal"` Especie de "SMS"
  - `"chat"` Conversación en tiempo real
  - `"groupchat"` Conversación entre varios usuarios (tipo IRC)
  - `"headline"` Aviso que no admite respuesta, sólo para quien esté conectado
  - `"error"` Transporta un mensaje de error

Protocolo

Mensajes: Extensiones relacionadas con mensajes

Muchas extensiones permiten incluir más información en los mensajes, tales como:

- Notificaciones de *estado del chat* (si el usuario teclea o para de teclear)
- Inclusión de XHTML en el cuerpo del mensaje (y así permitir tipos de letra, enlaces, etc.)
- Envío de información detallada de un usuario según el estándar `vCard` (nombre, dirección, teléfono, etc.)
- Bloqueo de usuarios para no recibir sus mensajes
- Listas de privacidad (bloqueo de ciertos tipos de *stanzas* desde ciertos contactos)
- Notificaciones de *recepción del mensaje* (para saber cuándo el mensaje llegó al cliente de destino)

Protocolo

IQ (*Info Query*)

El tag `<iq>` lleva los siguientes atributos:

- `from=` JID de quien hace la consulta, o quien la responde.
- `to=` A quién va dirigida la consulta (*bare* o *full* JID)
- `id=` Identificador único de la petición (el mismo en la respuesta)
- `type=` Tipo de *iq*. Puede ser:
  - `"get"` Solicita información (equivalente al HTTP GET)
  - `"set"` Cambia información (equivalente al HTTP PUT)
  - `"result"` Contiene la respuesta a una *iq* previa
  - `"error"` Error relativo a una *iq* previa

Protocolo

IQ: ¿Qué se solicita en un get?

- Cuando el atributo es `type="get"` se solicita *algo* al servidor (o al cliente de otro usuario).
- Qué se está solicitando viene dado por el contenido del tag `iq`
- Suele ser simplemente otro tag llamado `<query/>`

> La semántica de ese `query` (lo que significa) está definida en un espacio de nombres externo.
>
> Por tanto es ese espacio de nombres el que dice qué se está solicitando

```
<iq ...atributos... type="get">
  <query xmlns="espacio de nombres"/>
</iq>
```

Protocolo

IQ: Ejemplo: manejo del *roster*

El manejo del *roster* se hace mediante *stanzas* tipo `iq`

Solicitar el roster

```
<iq from="alice@wonderland.lit/pda"
    id="rr82a1z7"
    to="alice@wonderland.lit"
    type="get">
  <query xmlns="jabber:iq:roster"/>
</iq>
```

Observa el espacio de nombres de `query`

Protocolo

IQ Ejemplo: manejo del *roster* (2)

Respuesta a la solicitud anterior

```
<iq from="alice@wonderland.lit"
    id="rr82a1z7"
    to="alice@wonderland.lit/pda"
    type="result">
  <query xmlns="jabber:iq:roster">
    <item jid="whiterabbit@wonderland.lit"/>
    <item jid="lory@wonderland.lit"/>
    <item jid="mouse@wonderland.lit"/>
    <item jid="sister@realworld.lit"/>
  </query>
</iq>
```

Protocolo

IQ Ejemplo: manejo del *roster* (3)

Añadir un contacto al *roster*

```
<iq from="alice@wonderland.lit/pda"
    id="ru761vd7"
    to="alice@wonderland.lit"
    type="set">
  <query xmlns="jabber:iq:roster">
    <item jid="madhatter@wonderland.lit"/>
  </query>
</iq>
```

Respuesta (confirmación de que ha sido añadido, sin contenido)

```
<iq from="alice@wonderland.lit"
    id="ru761vd7"
    to="alice@wonderland.lit/pda"
    type="result"/>
```

Extensiones importantes de XMPP

XMPP tiene muchas extensiones, estandarizadas en documentos XEP. Algunas de las más importantes son:

- **XEP-0045: Multiuser Chat**
  - Implementa una especie de IRC con "salas de chat"
  - Pero tiene peor escalabilidad que IRC
- **XEP-0060: Publish-subscribe**
  - Implementa un mecanismo genérico para "notificaciones push"
  - Tiene muchas aplicaciones y sub-extensiones según lo que se publique (información sobre música, estado de ánimo del usuario, cambios de *nick*, cambios de *avatar*, etc)
  - Y puede usarse para tareas de control o automatización si los "usuarios" son *bots*

Extensiones

- **XEP-0166: Jingle**
  - Implementa un protocolo de inicio de sesión para poner en contacto directo dos dispositivos de cliente
  - Incorpora una fase de negociación de IPs, puertos, transportes, códecs, etc.
  - Permite realizar VoiP o videoconferencia directamente entre ellos
  - O también transferencia directa de archivos

Era usado por Google Talk