

**INFORME SOBRE SISTEMAS DISTRIBUIDOS**

**CURSO**

SISTEMAS DISTRIBUIDOS

**DOCENTE**

SAENZ TARAZONA, MANUEL ENRIQUE

**INTEGRANTES:**

* ARIAS SALCEDO, JEANPIERRE JESUS
* COCHACHIN QUITO, FRANK RONALD
* GONZALES ASTOCONDOR, MICHAEL
* MAZA CERNA, DENNIS ALEXIS
* MULLISACA BARRIENTOS, LUCIO ALBERTO

**2018**

<https://github.com/pensante-equivocado/utp/tree/master/Practica3>

**COMUNICACIÓN ORIENTADA A MENSAJES**

En los sistemas distribuidos, las llamadas a procedimientos remotos y las invocaciones a objetos remotos contribuyen a ocultar la comunicación, es decir, mejoran la transparencia de acceso. Por desgracia, ningún mecanismo es siempre adecuado. En particular, cuando no es posible asumir que el lado receptor está en ejecución al momento en que se hace una petición, se necesitan servicios de comunicación alternos. De igual manera, la naturaleza síncrona de las RPC, por la cual un cliente se bloquea hasta que su petición es procesada, algunas veces necesita remplazarse con algo más. Ese algo más es la mensajería. En esta sección nos concentraremos en la comunicación orientada a mensajes en sistemas distribuidos. Primero veremos qué es exactamente el comportamiento síncrono y cuáles son sus implicaciones; después explicaremos los sistemas de mensajería que suponen que las partes se ejecutan en tiempo de comunicación; y por último, analizaremos los sistemas de colas de mensajes que permiten a los procesos intercambiar información, incluso si la otra parte no se ejecuta al momento en que se inicia la comunicación.

**Comunicación transitoria orientada a mensajes**

Muchos sistemas y aplicaciones distribuidos se construyen directamente sobre el sencillo modelo orientado a mensajes ofrecido por la capa de transporte. Para comprender esto y apreciar a los sistemas orientados a mensajes como parte de las soluciones middleware, primero explicaremos la mensajería a través del nivel de transporte mediante sockets.

**Sockets Berkeley**

Se ha puesto especial atención a la estandarización de la interfaz de la capa de transporte para permitir a los programadores utilizar la suite completa de los protocolos (de mensajería) mediante un simple conjunto de primitivas. Además, las interfaces estándar facilitan llevar una aplicación a una máquina diferente.

Como un ejemplo, explicaremos brevemente la **interfaz de sockets** tal como se presentó en la década de 1970 en Berkeley UNIX. Otra interfaz importante es la **XTI**, que significa **X/Open Transport Interface** (Interfaz Abierta de Transporte/X), antes llamada Interfaz de la Capa de Transporte (TLI), y desarrollada por AT&T. Las interfaces sockets y XTI son muy parecidas en sus modelos de programación de red, pero difieren en sus conjuntos de primitivas.

En cuanto a su concepto básico, un **socket** 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. Un socket forma una abstracción sobre el punto final real de comunicación, el cual utiliza el sistema operativo local para un protocolo de transporte específico.

En el texto siguiente, nos concentraremos en las primitivas de sockets para TCP, las cuales aparecen en la figura.

Por lo general, los servidores ejecutan primero cuatro primitivas, normalmente en el orden dado.

Cuando se llama a la primitiva socket, el que llama crea un nuevo punto final de comunicación para un protocolo de transporte específico. De manera interna, crear un punto final de comunicación significa que el sistema operativo local reserva recursos para alojar los mensajes enviados y recibidos por el protocolo especificado.

La primitiva bind asocia una dirección local con el socket creado recientemente. Por ejemplo, un servidor debe enlazar la dirección IP de su máquina con un número de puerto (probablemente muy conocido) para un socket. El enlace le indica al sistema operativo que el servidor sólo desea recibir mensajes en la dirección y el puerto especificados.



Primitivas de socket para TCP/IP.

La primitiva listen es llamada sólo en el caso de la comunicación orientada a conexión. Es una llamada que no bloquea y que permite al sistema operativo local reservar los búferes suficientes para un número máximo especificado de conexiones que quien llama está dispuesto a aceptar.

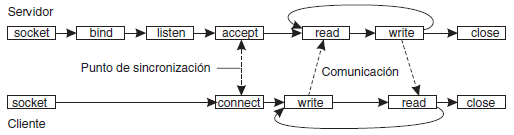
Una llamada a accept bloquea a quien llama hasta que llega una petición de conexión. Cuando llega una petición, el sistema operativo local crea un nuevo socket con las mismas propiedades del original, y lo devuelve a quien llama. Por ejemplo, este método permitirá al servidor generar un proceso que posteriormente manejará la comunicación real a través de la nueva conexión. Mientras tanto, el servidor puede volver y esperar otra petición de conexión con el socket original.

Ahora veamos del lado del cliente. Aquí también es necesario crear primero un socket mediante la primitiva socket, pero no es necesario enlazar explícitamente al socket con la dirección local ya que el sistema operativo puede destinar dinámicamente un puerto cuando se establece la conexión.

La primitiva connect requiere que quien llama especifique la dirección al nivel de transporte a la que se enviará la petición de conexión. El cliente es bloqueado hasta que se establece exitosamente una conexión, después de lo cual ambos lados pueden iniciar con el intercambio de información a través de las primitivas send y receive. Por último, cerrar una conexión resulta simétrico cuando se utilizan sockets, y se logra haciendo que tanto el cliente como el servidor llamen a la primitiva close. La figura 4-15 muestra el patrón general que sigue un cliente y un servidor para implementar la comunicación orientada a conexiones mediante sockets. En Stevens (1998) encontrará todos los detalles necesarios sobre programación de redes mediante sockets y otras interfaces en un ambiente UNIX.

**La interfaz de paso de mensajes (MPI)**

Con la llegada de multicomputadoras de alto rendimiento, los desarrolladores han buscado primitivas orientadas a mensajes que les permitan escribir fácilmente aplicaciones altamente eficientes. Esto significa que las primitivas deben estar en un nivel de abstracción conveniente (para un desarrollo sencillo de aplicaciones), y que su implementación sólo implique una sobrecarga mínima.



Patrón de comunicación orientada a conexiones mediante sockets. Que los sockets eran insuficientes por dos razones. Primero, se encontraban en el nivel de abstracción erróneo con el solo soporte de las primitivas simples para enviar y recibir. Segundo, los sockets se habían diseñado para comunicación a través de redes mediante protocolos de pilas de propósito general, como TCP/IP. Los sockets no se consideraron adecuados para los protocolos propietarios desarrollados para redes de interconexión de alta velocidad, tales como las utilizadas en clusteres de servidores de alto rendimiento. Aquellos protocolos requerían una interfaz que pudiese manejar características más avanzadas, como diferentes formas de utilizar el bufer y de sincronización.

El resultado fue que la mayoría de las redes de interconexión y multicomputadoras de alto rendimiento se pusieron en bibliotecas de comunicación propietarias. Estas bibliotecas ofrecían primitivas de comunicación valiosas, de alto nivel, y eficientes en general. Desde luego, todas las bibliotecas eran mutuamente incompatibles, por lo que los desarrolladores de aplicaciones tuvieron entonces un problema de portabilidad.

La necesidad de lograr independencia del hardware y de la plataforma derivó en algún momento en la definición de un estándar para el paso de mensajes, conocido simplemente como **Interfaz de Paso de Mensajes** o (**MPI**, por sus siglas en inglés). La MPI está diseñada para aplicaciones paralelas, y como tal, fue confeccionada para comunicación transitoria. Utiliza directamente la red subyacente.

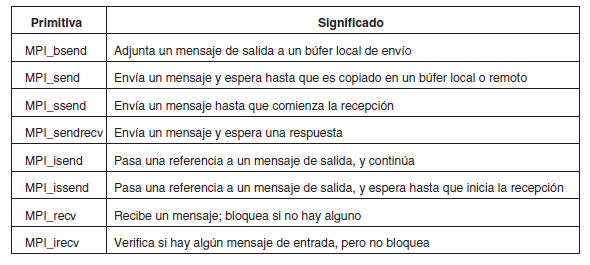
Además, asume que fallas serias, tal como caídas de procesos o particiones de red, son fatales y no requieren recuperación automática.

La MPI asume que la comunicación ocurre dentro de un grupo conocido de procesos. A cada grupo se le asigna un identificador. Dentro de un grupo, a cada proceso también se le asigna un identificador (local). Por lo tanto, un par (*IDgrupo*, *IDproceso*) únicamente identifica la fuente o el destino de un mensaje, y se utiliza en lugar de una dirección al nivel de transporte. Puede haber diversos grupos de procesos sobrepuestos involucrados en un cálculo, y que todos estén en ejecución al mismo tiempo.

En la parte central de la MPI se encuentran las primitivas de mensajería para soportar la comunicación transitoria, de las cuales las más conocidas aparecen en la figura 4-16.

La comunicación asíncrona transitoria se soporta mediante la primitiva MPI\_bsend. El remitente presenta un mensaje para su transmisión, el cual, generalmente se copia primero en un bufer local de la MPI del sistema en ejecución. Cuando se ha copiado el mensaje, el remitente continúa.

La MPI del sistema en ejecución eliminará el mensaje de su búfer local y se encargará de la transmisión tan pronto como un destinatario haya llamado a una primitiva de recepción.



Algunas de las primitivas para paso de mensajes más conocidas de la MPI.

También hay una operación de envío que causa bloqueo, conocida como MPI\_send, cuya semántica depende de la implementación. La primitiva MPI\_send puede bloquear al que llama hasta que el mensaje especificado se haya copiado en la MPI del sistema en ejecución del lado del remitente, o hasta que el destinatario haya iniciado una operación de recepción. La comunicación síncrona mediante la que el remitente se bloquea hasta que su petición es aceptada para un mayor procesamiento está disponible a través de la primitiva MPI\_ssend. Por último, también es soportada la forma más fuerte de comunicación síncrona: cuando un remitente llama a MPI\_sendrecv, ésta envía una petición al destinatario y se bloquea hasta que le devuelve una respuesta. Básicamente, esta primitiva corresponde a una RPC normal.

Tanto MPI\_send como MPI\_ssend tienen variantes que evitan la copia de mensajes desde buferes de usuarios a buferes internos y a la MPI del sistema en ejecución. Estas variantes corresponden a una forma de comunicación asíncrona. Con MPI\_isend, un remitente pasa un apuntador al mensaje, y después la MPI del sistema en ejecución se ocupa de la comunicación. El remitente continúa de inmediato. Para prevenir la sobreescritura en el mensaje antes de que la comunicación se complete, la MPI ofrece primitivas para verificar que se complete, o incluso que se bloquee si es necesario. Así como con MPI\_send, no está especificado si el mensaje se ha transferido hacia el destinatario o si simplemente lo ha copiado la MPI local del sistema en ejecución hacia un bufer interno.

De igual manera, con MPI\_issend, también un remitente pasa únicamente un apuntador a la MPI del sistema en ejecución. Cuando el sistema en ejecución indica que ha procesado el mensaje, el remitente está seguro de que el destinatario ha aceptado el mensaje y trabajará en él.

La operación MPI\_recv es llamada para recibir un mensaje; ésta bloquea a quien llama hasta que llega un mensaje. También hay una variante asíncrona, conocida como MPI\_irecv, mediante la cual un destinatario indica que está preparado para aceptar un mensaje. El destinatario puede verificar si en realidad ha llegado un mensaje, o se bloquea hasta que llega alguno.

La semántica de las primitivas de comunicación MPI no siempre es directa, y en ocasiones es posible intercambiar diferentes primitivas sin afectar la integridad de un programa. La razón oficial por la que se soportan muchas formas diferentes de comunicación es que proporciona a quienes implementan sistemas MPI suficientes posibilidades para optimizar el rendimiento. Los cínicos podrían decir, que el comité no pudo superar su mente colectiva, por lo que agregaron todo. La MPI se diseñó para aplicaciones paralelas de alto rendimiento, ello facilita comprender su diversidad en diferentes primitivas de comunicación.

En Gropp y colaboradores (1998b) puede encontrarse más sobre MPI. La referencia completa en la que se explican con detalle más de 100 funciones de MPI es Snir y colaboradores (1998), y Gropp y colaboradores (1998a).

**Comunicación persistente orientada a mensajes**

Ahora, explicaremos una clase importante de servicios middleware orientados a mensajes, generalmente conocidos como **sistemas de colas de mensajes**, o simplemente **Middleware Orientado a** **Mensajes (MOM**, por sus siglas en inglés). Los sistemas de colas de mensajes proporcionan un amplio soporte para comunicación asíncrona persistente. La esencia de estos sistemas es que ofrecen capacidad de almacenamiento de término medio para mensajes, sin la necesidad de que el remitente o el destinatario estén activos durante la transmisión del mensaje. Una diferencia importante con los sockets Berkeley y la MPI, es que los sistemas de colas de mensajes están dirigidos al soporte de transferencias de mensajes que toman minutos en lugar de segundos o milisegundos.

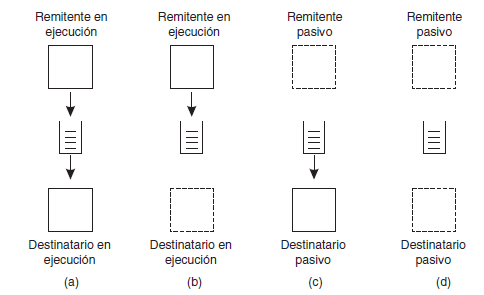
Primero explicaremos un método general de los sistemas de colas de mensajes, y concluiremos esta sección comparándolos con sistemas más tradicionales tales como los sistemas de correo electrónico de internet.

**Modelo de colas de mensajes**

La idea básica detrás de un sistema de colas de mensajes, es que las aplicaciones se comunican insertando mensajes en colas específicas. Estos mensajes son reenviados a una serie de servidores de comunicación y en algún momento se entregan en su destino, incluso si éste no estaba disponible cuando se envió el mensaje. En la práctica, la mayoría de los servidores de comunicación está directamente conectados uno con otro. En otras palabras, por lo general, un mensaje se transfiere directamente hacia un servidor destino. En principio, cada aplicación tiene su propia cola privada a la que otras aplicaciones pueden enviar mensajes. Una cola puede ser leída sólo por su aplicación asociada, pero también es posible que varias aplicaciones compartan una sola cola.

Un aspecto importante de los sistemas de colas de mensajes es que, por lo general, al remitente sólo se le garantiza que su mensaje se insertará en algún momento en la cola del destinatario. No se dan garantías sobre cuándo, o de si el mensaje en realidad será leído, lo cual es completamente definido por el comportamiento del destinatario.

Estas semánticas permiten la comunicación muy poco acoplada en tiempo. Entonces, no hay necesidad de que el destinatario se encuentre en ejecución cuando un mensaje se envía a su cola. De igual manera, no hay necesidad de que el remitente se encuentre en ejecución al momento en que su mensaje es recogido por el destinatario. El remitente y el destinatario pueden ejecutarse completamente independientes uno de otro. De hecho, una vez que se ha depositado un mensaje en una cola, permanecerá ahí hasta que sea eliminado, sin tomar en cuenta si el remitente o el destinatario se encuentran en ejecución. Esto nos proporciona cuatro combinaciones con respecto al modo de ejecución del remitente y del destinatario, como ilustra la figura.



Cuatro combinaciones de comunicaciones muy poco acopladas mediante el uso de colas.

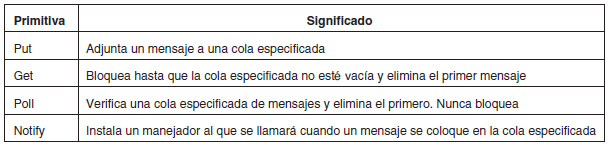
En la figura (a), tanto el remitente como el destinatario están en ejecución durante toda la transmisión de un mensaje. En la figura 4-17(b), sólo el remitente está en ejecución mientras el destinatario se encuentra pasivo, es decir, en un estado en el que no es posible la entrega del mensaje.

Sin embargo, el remitente puede enviar mensajes. La combinación de un remitente pasivo y un destinatario en ejecución aparece en la figura 4-17(c). En este caso, el destinatario puede leer mensajes que le fueron enviados, pero no es necesario que sus respectivos remitentes se encuentren en ejecución. Por último, en la figura 4-17(d), vemos la situación en que el sistema almacena (y posiblemente transmite) mensajes, incluso mientras el remitente y el destinatarios se encuentran pasivos.

En principio, los mensajes pueden contener cualquier información. El único aspecto importante a cubrir desde la perspectiva del middleware es que los mensajes se direccionen adecuadamente.

En la práctica, el direccionamiento se hace proporcionando un nombre único de la cola de destino. En algunos casos el tamaño del mensaje puede limitarse, aunque también es posible que el sistema subyacente se encargue de fragmentar y ensamblar grandes mensajes de manera completamente transparente para las aplicaciones. Un efecto de este método, es que la interfaz básica ofrecida a las aplicaciones puede ser extremadamente simple, como indica la figura 4-18.

Un remitente llama a la primitiva put para pasar un mensaje al sistema subyacente que va a adjuntarse a la cola especificada. Como explicamos, ésta es una llamada que no bloquea. La primitiva get es una llamada bloqueadora mediante la cual un proceso autorizado puede eliminar el mensaje pendiente más antiguo de la cola especificada. El proceso es bloqueado sólo si la cola está vacía. Variantes de esta llamada permiten la búsqueda de un mensaje específico en la cola, por ejemplo, mediante una prioridad o un patrón de coincidencias. La variante que no bloquea está dada por la primitiva poll. Si la cola está vacía, o si un mensaje específico no pudo encontrarse, el proceso de llamada simplemente continúa.

Interfaz básica de una cola en un sistema de colas de mensajes.

Por último, la mayoría de los sistemas de colas también permiten desarrollar un proceso para instalar un manejador como una función de *devolución de llamada*, la cual, es automáticamente invocada cuando un mensaje se coloca en la cola. Las devoluciones de llamada también pueden utilizarse para iniciar automáticamente un proceso que traerá mensajes de la cola si ningún proceso se encuentra en ejecución. Este método se implementa con frecuencia mediante un demonio del lado del destinatario, quien continuamente monitorea los mensajes entrantes a la cola y los maneja según corresponda.

**Arquitectura general de un sistema de colas de mensajes**

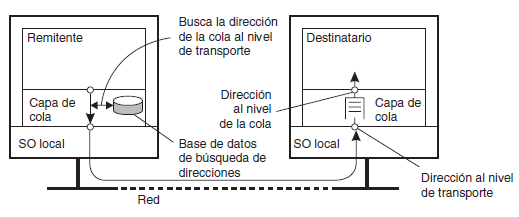
Ahora veamos cómo luce en general un sistema de colas de mensajes. Una de las primeras restricciones que hacemos es que los mensajes sólo pueden colocarse en colas que sean *locales* para el remitente, es decir, colas en la misma máquina, o nada peor que en una máquina cercana tal como en la misma LAN que puede ser alcanzada eficientemente a través de una RPC. A tal cola se le conoce como **cola fuente**. De igual manera, los mensajes sólo pueden leerse desde colas locales. Sin embargo, un mensaje colocado en una cola contendrá la especificación de una **cola de destino** a la que debe ser transferido. Es responsabilidad del sistema de colas de mensajes proporcionar colas a los remitentes y destinatarios, y encargarse de que los mensajes se transfieran desde su cola fuente hasta su cola destino.

Es importante advertir que la colección de colas se distribuye a través de diversas máquinas.

En consecuencia, para que un sistema de colas de mensajes transfiera mensajes, es necesario que mantenga un mapa de las colas para localización en la red. En la práctica, esto significa que debe mantener una base de datos (probablemente distribuida) de los **nombres de las colas** para localizarlas en la red, como ilustra la figura. Observe que tal mapeo es completamente análogo al uso del

Servicio de Nombres de Dominio (DNS, por sus siglas en inglés) para correo electrónico en internet.

Por ejemplo, cuando se envía un mensaje a la dirección lógica de *correo steen@cs.vu.nl*, el sistema de correo consultará al DNS para localizar la dirección de *red* (es decir, la IP) del servidor de correo del destinatario para usarlo en la transferencia real del mensaje.



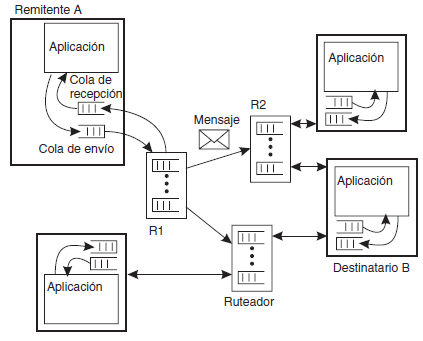
Relación entre el direccionamiento al nivel de cola y el direccionamiento al nivel de red.

Las colas son manejadas por **administradores de colas**. Por lo general, un administrador de colas interactúa directamente con la aplicación que está enviando o recibiendo un mensaje. Sin embargo, también hay administradores especiales de colas que operan como ruteadores, o **retransmisores**: éstos retransmiten mensajes entrantes hacia otros administradores de colas. De esta manera, un sistema de colas de mensajes puede convertirse en toda una **red sobrepuesta**, al nivel de aplicaciones, en el nivel superior de una red de computadoras existente. Este método es similar a la construcción de Mbone en internet, donde los procesos ordinarios de usuario se configuraron como ruteadores de multitransmisión. Como podemos ver, la multitransmisión a través de redes sobrepuestas aún es importante, y la explicaremos más adelante en este capítulo.

Los retransmisores pueden resultar convenientes por varias razones. Por ejemplo, en muchos sistemas de colas de mensajes no hay un servicio de asignación de nombres disponible y que pueda mantener dinámicamente mapas para localización de colas. En cambio, la topología de la red de colas es estática, y cada administrador de colas necesita una copia de dichos mapas. No hay necesidad de decir que en los grandes sistemas de colas este método, puede derivar en problemas de administración de red.

Una solución es utilizar algunos ruteadores que conozcan la topología de la red. Cuando un remitente *A* coloca un mensaje en su cola para el destinatario *B*, ese mensaje se transfiere primero al ruteador más cercano, digamos *R1*, como ilustra la figura 4-20. En ese punto, el ruteador sabe qué hacer con el mensaje y lo reenvía en la dirección de *B*. Por ejemplo, *R1* puede deducir, a partir del nombre de *B*, que el mensaje debe reenviarse al ruteador *R2*. De este modo, sólo es necesario actualizar los ruteadores cuando las colas se agregan o eliminan, mientras que todos los demás administradores de colas sólo necesitan saber en dónde se encuentra el ruteador más cercano.

Así, los retransmisores pueden ayudar a construir sistemas escalables de colas de mensajes. Sin embargo, cuando las redes de colas crecen, resulta claro que la configuración manual de las redes se volverá rápidamente no administrable. La única solución es adoptar esquemas de enrutamiento dinámico tal como se hace con redes de computadoras. Con respecto a esto, de alguna manera resulta sorprendente que tales soluciones aún no se integran en algunos de los sistemas de colas de mensajes más populares.



Organización general de un sistema de colas de mensajes con ruteadores.

Otra razón por la cual se utilizan retransmisores es que permiten el procesamiento secundario de mensajes. Por ejemplo, los mensajes pueden necesitar un registro por razones de seguridad o de tolerancia a fallas. Una forma especial de retransmisor, que explicaremos en la siguiente sección, es una que actúa como puerta de enlace, cambiando los mensajes a un formato que el destinatario puede entender.

Por último, los retransmisores pueden utilizarse con propósitos de multitransmisión. En ese caso, un mensaje entrante simplemente se coloca en cada cola de envío.

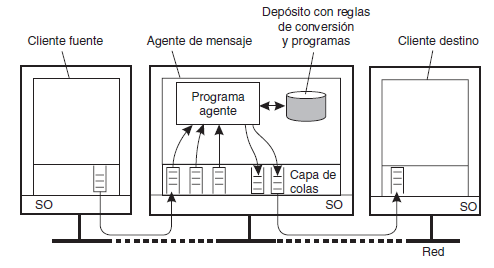
**Agentes de mensajes**

Un área de aplicación importante de los sistemas de colas de mensajes, es la integración de aplicaciones nuevas y existentes en un solo sistema de información distribuido y coherente. La integración requiere que las aplicaciones puedan comprender los mensajes recibidos. En la práctica, esto requiere que el remitente tenga sus mensajes de salida en el mismo formato que el destinatario.

El problema con este método es que cada vez que una aplicación agregada al sistema necesite un formato diferente de mensajes, cada remitente potencial tendrá que ajustarse para poder producir ese formato.

Una alternativa es coincidir con un formato común de mensajes, como se hace con los protocolos tradicionales de red. Por desgracia, este método generalmente no funciona con los sistemas de colas de mensajes. El problema subyacente es el nivel de abstracción en el que estos sistemas operan. Un formato común de mensajes tiene sentido sólo si la colección que utiliza dicho formato tiene en realidad suficientes cosas en común. Si la colección de aplicaciones que forman un sistema de información distribuido es muy diversa (y con frecuencia es así), entonces el mejor formato común puede ser no más que una secuencia de bytes.

Aunque se han definido algunos formatos comunes de mensajes para dominios de aplicaciones específicas, el método general es aprender a vivir con formatos diferentes, e intentar proveer los medios para que las conversiones sean tan sencillas como sea posible. En sistemas de colas de mensajes, las conversiones son manejadas por nodos especiales de una red de colas conocidos como **agentes de mensajes**. Un agente de mensaje actúa como una puerta de enlace al nivel de aplicaciones en un sistema de colas de mensajes. Su propósito principal es convertir los mensajes entrantes de tal manera que la aplicación de destino pueda comprenderlos. Observe que para un sistema de colas de mensajes, un agente de mensaje es sólo otra aplicación, como indica la figura 4-21. En otras palabras, un agente de mensaje no se considera, por lo general, parte integral del sistema de colas.



Organización general de un agente de mensaje en un sistema de colas de mensajes.

Un agente de mensajes puede ser tan sencillo como un reformateador de mensajes. Por ejemplo, supongamos que un mensaje entrante contiene una tabla de una base de datos en la que los registros están separados por un delimitador especial de *fin de registro*, y que los campos localizados dentro de un registro tienen una longitud fija conocida. Si la aplicación destino espera un delimitador diferente entre registros, y también espera que los campos tengan longitudes variables, se puede utilizar un agente de mensajes para convertirlos al formato esperado por el destinatario.

En una configuración avanzada, un agente de mensaje puede actuar como una puerta de enlace al nivel de aplicaciones similar a la puerta que maneja la conversión entre dos aplicaciones de bases de datos diferentes. En tales casos, no es posible garantizar que toda la información contenida en el mensaje de entrada pueda transformarse realmente en algo adecuado para el mensaje de salida.

Sin embargo, es más común utilizar un agente de mensajes para la **integración de aplicaciones empresariales** (**EAI**, por sus siglas en inglés), como explicamos en el capítulo 1. En este caso, en lugar de (sólo) convertir mensajes, un agente es responsable de hacer coincidir aplicaciones basadas en los mensajes que se están intercambiando. En tal modelo, llamado de **publicación-suscripción**, las aplicaciones envían mensajes en la forma de *publicaciones*. En particular, pueden publicar un mensaje sobre el tema *X*, el cual después se envía al agente. Las aplicaciones que han expresado su interés por el tema *X*, es decir, que han *suscrito* dichos mensajes, recibirán entonces estos mensajes desde el agente. También hay formas más avanzadas de mediación, pero postergaremos su explicación hasta el capítulo 13.

En el núcleo de un agente de mensaje se encuentra un depósito de reglas y programas que pueden transformar un mensaje del tipo *T1* en uno del tipo *T2*. El problema es definir las reglas y desarrollar los programas. La mayoría de los productos de agentes de mensajes, vienen con herramientas sofisticadas de desarrollo, pero el resultado final es que el depósito debe ser llenado por expertos. Aquí tenemos un ejemplo perfecto, donde productos comerciales afirman proveer, a menudo engañosamente, “inteligencia”, cuando en realidad la inteligencia radica en las cabezas de los expertos.

**Nota sobre sistemas de colas de mensajes**

Si consideramos lo que hemos dicho sobre los sistemas de colas de mensajes, parecería que han existido desde hace mucho en forma de implementaciones para servicios de correo electrónico.

Los sistemas de correo electrónico se implementan, por lo general, a través de una colección de servidores de correo que almacenan y reenvían mensajes en beneficio de los usuarios de los sistemas directamente conectados al servidor. El enrutamiento a menudo se deja de lado, ya que los sistemas de correo electrónico pueden utilizar directamente los servicios de transporte subyacentes.

Por ejemplo, en el protocolo de correo para internet SMTP (Postel, 1982), un mensaje se transfiere mediante la configuración de una conexión directa TCP hacia el servidor de correo de destino.

Lo que hace especiales a los sistemas de correo electrónico, comparados con los sistemas de colas de mensajes, es que se utilizan básicamente para proporcionar soporte directo a usuarios finales.

Esto explica, por ejemplo, por qué un número de aplicaciones groupware se basa directamente en un sistema de correo electrónico (Khoshafian y Buckiewicz, 1995). Además, los sistemas de correo electrónico pueden tener requerimientos muy específicos, como el filtrado automático de mensajes, soporte para bases de datos de mensajería avanzada (por ejemplo, para recuperar mensajes almacenados previamente), etcétera.

Los sistemas generales de colas de mensajes no se utilizan sólo para soporte de usuarios finales.

Una cuestión importante es que se configuran para habilitar la comunicación persistente entre procesos, sin importar si un proceso está ejecutando una aplicación de usuario, manejando el acceso a una base de datos, realizando cálculos, etc. Este método conduce a un conjunto de requerimientos diferente para los sistemas de colas de mensajes que para los sistemas de correo puros. Por ejemplo, los sistemas de correo electrónico generalmente no necesitan proporcionar garantías para entrega de mensajes, prioridades de mensajes, facilidades de registro, multitransmisión eficiente, balanceo de cargas, tolerancia a fallas, y demás elementos de uso general.

Por tanto, los sistemas de colas de mensajes de propósito general tienen un amplio rango de aplicaciones, incluyendo correo electrónico, carga de trabajo, groupware, y procesamiento por lotes. Sin embargo, como ya establecimos, el área de aplicación más importante es la integración de una colección (posiblemente muy dispersa) de bases de datos y aplicaciones en un sistema de información federado (Hohpe y Woolf, 2004). Por ejemplo, una consulta que se expande a diversas bases de datos podría necesitar dividirse en subconsultas que se reenvíen a bases de datos individuales. Los sistemas de colas de mensajes ayudan a proporcionar los medios básicos para empacar cada subconsulta en un mensaje y enrutarlo hacia la base de datos adecuada.

Otras facilidades de comunicación explicadas en este capítulo hasta el momento son más o menos adecuadas.

**Ejemplo: sistema de colas de mensajes WebSphere de IBM**

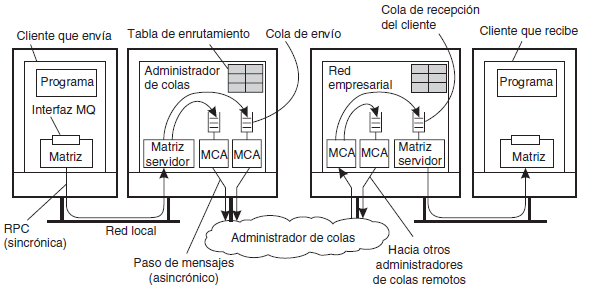
Para ayudarnos a comprender cómo funcionan en la práctica los sistemas de colas de mensajes, veamos un sistema específico, a saber, el sistema de cola de mensajes que es parte del producto WebSphere de IBM. Antes se le conocía como MQSeries, y ahora como **WebSphere MQ**. Hay mucha documentación sobre WebSphere MQ, y en adelante sólo podemos recurrir a los principios básicos. Muchos detalles arquitectónicos concernientes a redes de colas de mensajes pueden encontrarse en IBM (2005b, 2005d). Programar redes de colas de mensajes no es algo que pueda aprenderse en un domingo por la tarde, y la guía de programación de MQ (IBM, 2005a) es un buen ejemplo que muestra que ir de los principios a la práctica puede requerir de un gran esfuerzo.

**Visión general**

La arquitectura básica de una red de colas MQ es muy directa, según muestra la figura 4-22. Todas las colas son manejadas por **administradores de colas**. Un administrador de colas es responsable de eliminar mensajes de sus colas de envío, y de reenviarlos a otros administradores de colas. De igual manera, un administrador de colas es responsable de manejar mensajes de entrada; los toma de la red subyacente y luego almacena cada mensaje en la cola de entrada adecuada. Para tener una idea de lo que la mensajería puede significar: un mensaje tiene un tamaño máximo predeterminado de 4 MB, pero éste puede aumentar hasta 100 MB. Por lo general, una cola está restringida a 2 GB de información, pero según el sistema operativo subyacente, este máximo puede configurarse fácilmente para que sea mayor.

Los administradores de colas están conectados por pares a través de **canales de mensajes**, los cuales son una abstracción de las conexiones al nivel de transporte. Un canal de mensajes es una conexión unidireccional confiable, entre un administrador de cola de envío y uno de recepción, a través de la cual se transportan los mensajes en cola. Por ejemplo, un canal de mensajes basado en internet se implementa como una conexión TCP. Cada uno de los dos extremos de un canal de mensajes es administrado por un **agente de canal de mensajes** (**MCA**, por sus siglas en inglés). Un MCA de envío prácticamente nada más verifica las colas de envío para un mensaje, lo envuelve en un paquete al nivel de transporte, y lo envía a través de la conexión con su MCA de recepción asociado.

De igual manera, la tarea básica de un MCA de recepción es escuchar un paquete entrante, desenvolverlo, y posteriormente almacenarlo en la cola adecuada.

Organización general del sistema de cola de mensajes de IBM.

Los administradores de colas pueden vincularse al mismo proceso que la aplicación para la que administran las colas. En ese caso, las colas se ocultan de la aplicación detrás de una interfaz estándar, pero pueden ser manipuladas directamente por la aplicación. Una organización alterna es aquella donde los administradores de colas y las aplicaciones se ejecutan en máquinas separadas. En ese caso, a la aplicación se le ofrece la misma interfaz que cuando el administrador de colas se coloca en la misma máquina. Sin embargo, la interfaz se implementa como un proxy que se comunica con el administrador de colas mediante comunicación sincrónica basada en RCP tradicional. De esta manera, MQ básicamente conserva el modelo al que sólo pueden acceder las colas locales de una aplicación.

**Canales**

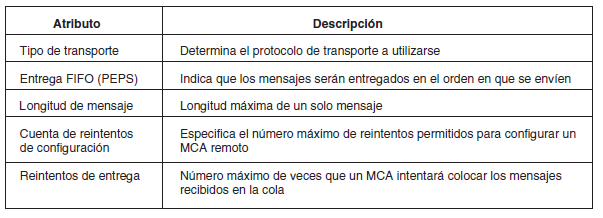
Un componente importante de MQ está formado por los canales de mensajes. Cada canal de mensaje tiene una cola de envío asociada desde la cual atrae los mensajes que debe transferir hacia el otro extremo. A lo largo del canal, la transferencia puede ocurrir sólo si sus MCA de envío y recepción están arriba y en ejecución. Además de poder iniciar ambos MCA manualmente, existen diversas formas alternas para iniciar un canal, algunas de las cuales explicaremos a continuación.

Una alternativa es hacer que una aplicación inicie directamente un extremo del canal, activando el MCA de envío o recepción. Sin embargo, desde un punto de vista de transparencia, ésta no es una alternativa atractiva. Un mejor método para iniciar un MCA de *envío* es configurar la cola de envío del canal para iniciar un disparador cuando un mensaje se coloque primero en la cola. Ese disparador está asociado con un manipulador para iniciar el MCA de envío en tal forma que pueda eliminar mensajes de la cola de envío.

Otra alternativa es iniciar un MCA sobre la red. En particular, si un lado de un canal ya está activo, el canal puede enviar un mensaje de control para solicitar se inicie el otro MCA. Tal mensaje de control se envía a un demonio que espera una dirección muy conocida en la misma máquina donde iniciará el otro MCA.

Los canales se detienen automáticamente después de pasado cierto tiempo durante el cual ningún otro mensaje llegó a la cola de envío.

Cada MCA tiene un conjunto de atributos asociados que determinan el comportamiento de todo el canal. Algunos de los atributos se encuentran en la figura 4-23. Los valores de los atributos de los MCA de envío y recepción deben ser compatibles, y tal vez negociados, antes de que pueda configurarse un canal. Por ejemplo, resulta evidente que ambos MCA deben soportar el mismo protocolo de transporte. Un ejemplo de un atributo no negociable, es si los mensajes se entregarán o no en el mismo orden en el que se colocan en la cola de envío. Si uno de los MCA quiere una entrega FIFO (PEPS), el orden debe respetarse. Un ejemplo de un valor de atributo negociable es la longitud máxima de mensaje, la cual se elegirá como el valor mínimo especificado por cualquiera de los MCA.

Algunos atributos asociados con agentes del canal de mensajes.

**Transferencia de mensajes**

Para transferir un mensaje desde un administrador de cola hacia otro (probablemente remoto), es necesario que cada mensaje lleve su dirección de destino, para lo cual se utiliza un encabezado de transmisión. Una dirección en MQ consta de dos partes: la primera parte contiene el nombre del administrador de cola al que se entregará el mensaje; y la segunda parte es el nombre de la cola de destino a la que el administrador adjuntará el mensaje.

Además de la dirección de destino, también es necesario especificar la ruta que debe seguir un mensaje. La especificación de la ruta se hace proporcionando el nombre de la cola local de envío a la que se adjuntará un mensaje. De esta manera, no es necesario proporcionar la ruta completa en un mensaje. Recuerde que cada canal de mensaje tiene exactamente una cola de envío. Al indicar a qué cola de envío se adjuntará un mensaje, especificamos efectivamente a qué administrador de cola se reenviará el mensaje.

En la mayoría de los casos, las rutas son almacenadas explícitamente dentro de un administrador de cola en una tabla de enrutamiento. Una entrada en una tabla de enrutamiento es un par (*destQM*, *sendQ*), donde *destQM* es el nombre del administrador de cola de destino, y *sendQ* es el nombre de la cola local de envío a la que debe adjuntarse el mensaje para ese administrador de cola. (En MQ, una entrada de la tabla de enrutamiento se conoce como alias.)

Es posible que sea necesario transferir un mensaje a través de varios administradores de cola antes de que llegue a su destino. Siempre que un administrador de cola intermediario reciba el mensaje, simplemente extraerá el nombre del administrador de cola de destino del encabezado del mensaje, y hará una búsqueda en la tabla de enrutamiento para encontrar la cola local de envío a la que el mensaje debe adjuntarse.

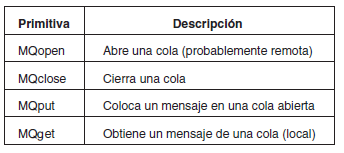
Es importante observar que cada administrador de cola tiene un nombre único en el sistema que efectivamente se usa como identificador para ese administrador de cola. El problema con el uso de estos nombres, es que al remplazar a un administrador de cola, o al cambiar su nombre, se afectarán todas las aplicaciones que le envían mensajes. Estos problemas pueden aligerarse con el uso de un **alias local** para nombres de administradores de cola. Un alias definido dentro de un administrador de cola *M1* es otro nombre para un administrador de cola *M2*, pero que está disponible sólo para aplicaciones de interfaz con *M1*. Un alias permite el uso del mismo nombre (lógico) de una cola, incluso si cambia el administrador de cola de esa cola. Cambiar el nombre de un administrador de cola requiere que cambiemos su alias en todos los administradores de cola. Sin embargo, las aplicaciones no se verán afectadas.



Organización general de una red de colas MQ que utiliza tablas de enrutamiento y de alias.

El principio para utilizar tablas de enrutamiento y alias aparece en la figura. Por ejemplo, una aplicación vinculada al administrador de cola *QMA* puede referirse a un administrador remoto de cola mediante el alias local *LA1*. El administrador de cola primero buscará el destino real en la tabla de alias para descubrir que es un administrador de cola *QMC*. La ruta para *QMC* está en la tabla de enrutamiento, la cual establece que los mensajes para *QMC* deben adjuntarse a la cola de salida *SQ1*, la cual se utiliza para transferir mensajes al administrador de cola *QMB*. Este último utilizará su tabla de enrutamiento para reenviar el mensaje a *QMC*.

Al seguir este método de enrutamiento y uso de alias se llega a una interfaz de programación que es, por principio, relativamente simple, conocida como **Interfaz de Cola de Mensajes** (**MQI**, por sus siglas en inglés). Las primitivas más importantes de la MQI aparecen en la figura.



Primitivas disponibles en la interfaz de cola de mensajes.

Para colocar mensajes en una cola, una aplicación llama a la primitiva MQopen, especificando una cola de destino en un administrador específico de cola. El administrador de cola puede nombrarse mediante el uso de un alias local disponible. Ya sea que la cola de destino sea remota o no, es completamente transparente para la aplicación. También es necesario llamar a MQopen si la aplicación quiere obtener mensajes a partir de su cola local. Solamente las colas locales pueden abrirse para leer mensajes entrantes. Cuando una aplicación termina su acceso a una cola, ésta debe cerrar mediante una llamada a MQclose.

Los mensajes pueden escribirse para una cola, o leerse desde una cola, mediante MQput y MQget, respectivamente. En principio, los mensajes se eliminan de una cola de acuerdo con su prioridad. Los mensajes con la misma prioridad se eliminan conforme al principio de primero en entrar, primero en salir; es decir, el mensaje pendiente de más larga permanencia se elimina primero.

También es posible solicitar mensajes específicos. Por último, MQ proporciona herramientas para indicar a las aplicaciones que ha llegado un mensaje, y así evitar que una aplicación tenga que preguntar continuamente por mensajes entrantes a la cola.

**Administración de redes sobrepuestas**

Por la descripción que hemos hecho hasta el momento, debe resultar claro que una parte importante de la administración de sistemas MQ es conectar a los diversos administradores de colas en una red sobrepuesta consistente. Más aún, con el tiempo, esta red necesita mantenimiento. Para redes pequeñas, este mantenimiento no requerirá más que un trabajo administrativo promedio, pero las cosas se complican cuando la cola de mensajes se utiliza para integrar y desintegrar grandes sistemas existentes.

Un asunto más importante es que las redes sobrepuestas deben administrarse manualmente.

Esta administración no sólo involucra la creación de canales entre administradores de cola, sino también el llenado de las tablas de enrutamiento. Evidentemente, esto puede crecer hasta convertirse en una pesadilla. Por desgracia, el soporte para administración de sistemas MQ es avanzado sólo en el sentido de que un administrador puede configurar virtualmente cualquier posible atributo, y ajustar cualquier configuración concebible. Sin embargo, el resultado final es que a los canales y a las tablas de enrutamiento se les debe dar mantenimiento manualmente.

En el núcleo de la administración sobrepuesta se encuentra la **función de control de canales**, la cual lógicamente se ubica entre los agentes del canal de mensajes. Este componente permite que un operador dé un seguimiento exacto a lo que sucede en dos puntos finales de un canal. Además, se utiliza para crear canales y tablas de enrutamiento, pero también para manejar a los administradores de cola que hospedan a los agentes del canal de mensajes. De cierto modo, este método para administración de sobrepuestas se parece bastante a la administración de servidores cluster donde se utiliza un solo servidor de administración. En el último caso, el servidor ofrece esencialmente sólo un shell remoto a cada máquina del cluster, junto con algunas operaciones colectivas para manejar grupos de máquinas. La buena noticia sobre la administración de sistemas distribuidos es que ofrece muchas oportunidades si usted está buscando un área para explorar nuevas soluciones a problemas serios.

**COMUNICACIÓN ORIENTADA A FLUJOS**

**Definición:**

La comunicación, se ha concentrado en el intercambio más o menos independiente y completo de unidades de información. En Halsall en el año 2001 se explican diversos protocolos de red que tratan con la comunicación orientada a flujos.

Steinmetz y Nahrstedt en el año 2004 proporcionan una introducción general a cuestiones multimedia, parte de la cual forma la comunicación orientada a flujos. Babcock y colaboradores en el año 2002 explican el procesamiento de consultas en flujos de datos.

En este tipo de comunicación es que no importa en qué punto del tiempo en particular ocurre la comunicación. Aunque un sistema puede comportarse de forma muy lenta o muy rápidamente, la sincronización no tiene efecto alguno sobre la integridad.

También hay formas de comunicación en las que la sincronización juega un papel crucial

**Soporte para medios continuos**

El soporte para intercambiar información dependiente del tiempo con frecuencia se conoce como soporte para medios continuos. Un medio se refiere al recurso mediante el cual se transmite la información. Estos recursos incluyen a los medios de almacenamiento y transmisión, medios de presentación como un monitor, etc. Un tipo importante de medio es la forma en que se *representa*

Generalmente se codifica en ASCII o Unicode. Las imágenes pueden representarse en diferentes formatos, como GIF o JPEG. En un sistema de cómputo, los flujos de audio pueden codificarse al tomar muestras de 16 bits empleando PCM.

En medios continuos, las relaciones temporales entre diferentes elementos de datos resultan fundamentales para interpretar correctamente lo que significan en realidad los datos.

Ejemplo: El movimiento puede representarse mediante una serie de imágenes en la que deben desplegarse imágenes sucesivas en un espacio uniforme en el tiempo, por lo general de 30 a 40 milisegundos por imagen.



**Soporte para medios discretos:**

La sincronización entre un texto y una imagen estática la diferencia puede no ser tan clara normalmente se considera que una aplicación es multimedia cuando se combina al menos un medio discreto con al menos un medio continuo.

Se caracterizan por el hecho de que las relaciones temporales entre elementos de datos no son fundamentales para interpretar correctamente los datos. Ejemplos típicos de medios discretos incluyen las representaciones de texto y fotogramas, pero también código objeto o archivos ejecutables

**Tipo de datos**

**Flujo de datos**

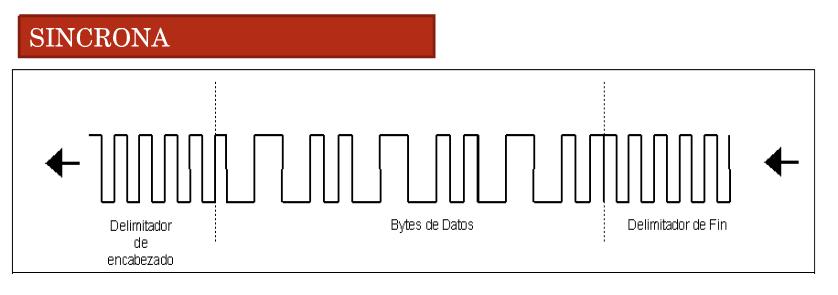
Un flujo de datos no es otra cosa más que una secuencia de unidades de datos.

Los flujos pueden ser simples o complejos

**Modo de transmisión síncrona**

Los elementos de datos de un flujo se transmiten uno después de otro, pero no hay más restricciones de sincronización en cuanto a cuándo debe ocurrir la transmisión de elementos. Éste generalmente es el caso para los flujos discretos de datos.

Gráfico de modo de transmisión síncrona



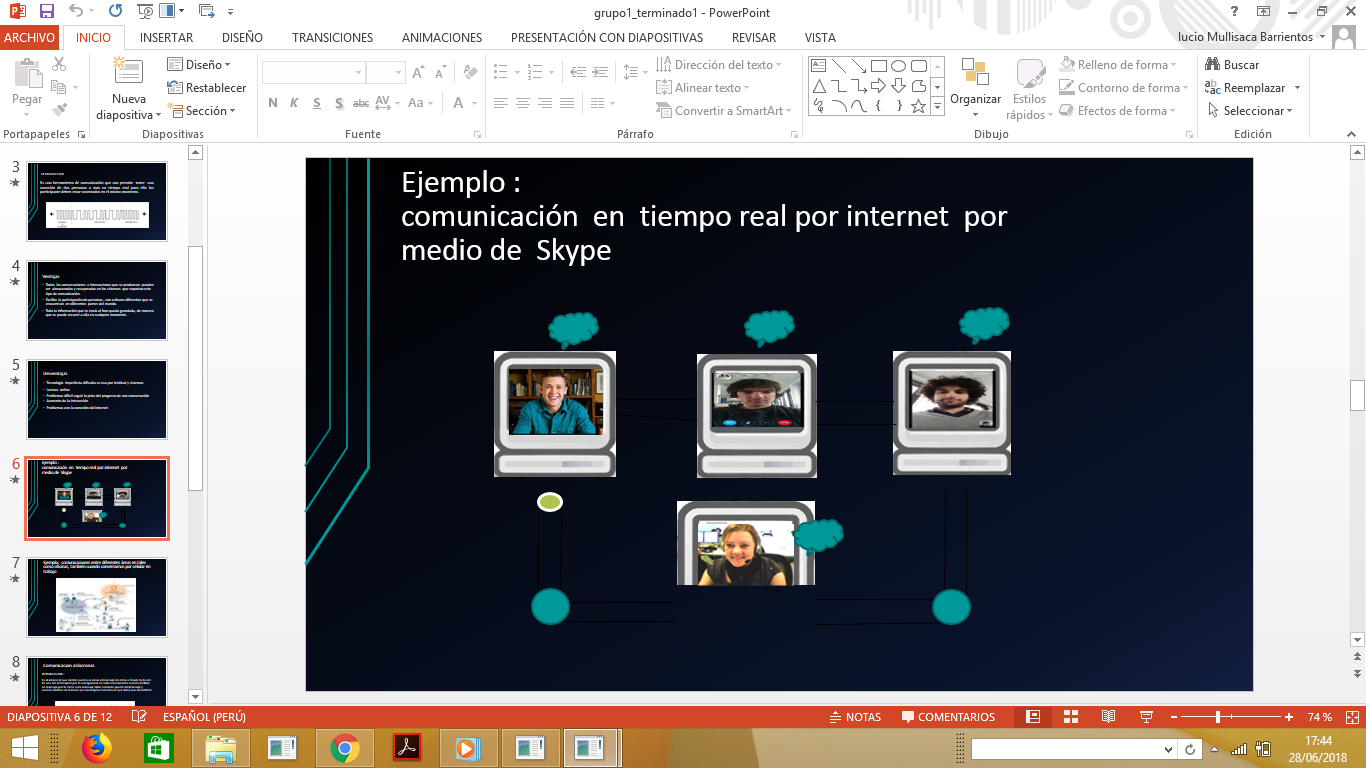
**Ventajas:**

* Todas las conversaciones e interacciones que se produzcan pueden ser almacenadas y recuperadas en los sistemas que soportan este tipo de comunicación.
* Facilita la participación de personas, con culturas diferentes que se encuentran en diferentes partes del mundo.
* Toda la información que se envía al foro queda guardada, de manera que se puede recurrir a ella en cualquier momento.

**Desventajas:**

* Tecnología imperfecta dificulta su uso por lentitud y sistemas
* Problemas difícil seguir la pista del progreso de una conversación
* Problemas con la conexión del internet

Ejemplo: En la empresa deltron peru los trabajadores de diferentes áreas utilizan el skyte para poder comunicarse con los diferentes clientes



**Modo de transmisión asíncrona**

Son unidades de datos que se transfieren a tiempo. Esto significa que la transferencia de datos está sujeta a un retraso máximo *y* mínimo fin a fin, también conocido como inestabilidad limitada (retraso) también resulta particularmente interesante para sistemas multimedia distribuidos, ya que desempeña un papel muy importante en la representación de audio y video.

 Grafica de modo de transmisión asíncrona

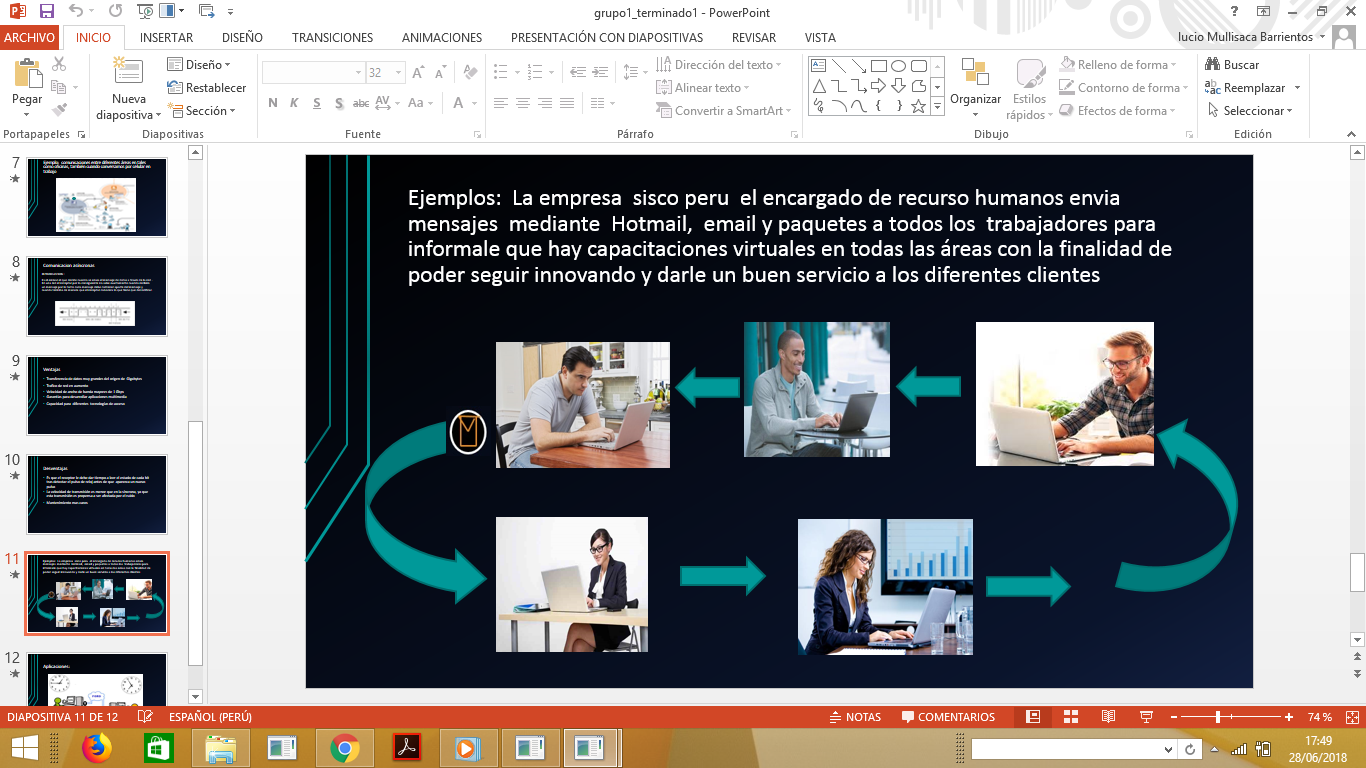
**Ventajas:**

* Transferencia de datos muy grandes del origen de Gigabytes
* Trafico de red en aumento
* Velocidad de ancho de banda mayores de 1 GBbps
* Garantías para desarrollar aplicaciones multimedia
* Capacidad para diferentes tecnologías de acceso

**Desventajas**:

* El receptor le debe dar tiempo a leer el astado de cada bit tras detectar el pulso de reloj antes de que aparezca un nuevo pulso
* La velocidad de transmisión es menor, ya que esta transmisión es propensa a ser afectada por el ruido.
* Mantenimiento más caro.

Ejemplo: La empresa sisco peru el encargado de recurso humanos envía mensajes mediante Hotmail, email y paquetes a todos los trabajadores para infórmale que hay capacitaciones virtuales en todas las áreas con la finalidad de poder seguir innovando y darle un buen servicio a los diferentes clientes.



**Flujo simple:**

Consiste únicamente en una sola secuencia de datos.

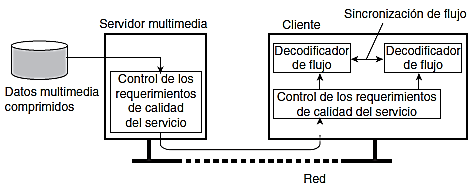
**Flujo complejo:**

Consta de varios flujos simples relacionados, llamados subflujos. La relación entre los subflujos de un flujo complejo a menudo también depende

Del tiempo. Sin embargo, es importante que esos dos subflujos estén continuamente sincronizados. En otras palabras, las unidades de datos de cada flujo se comunicarán por pares para garantizar el efecto estereofónico.

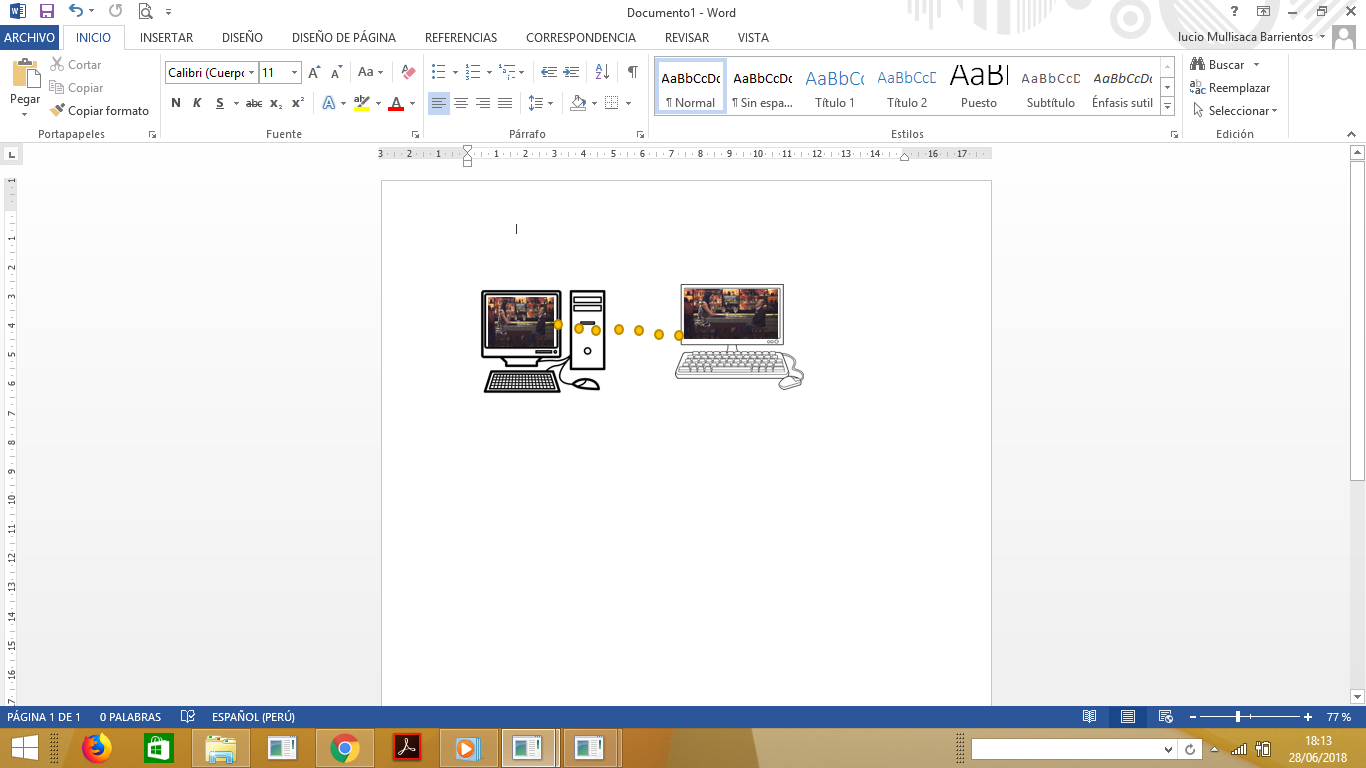
Si una la sincronización falla, la reproducción de la película también fallará. Más adelante retomaremos el tema de la sincronización de flujos.

Desde la perspectiva de los sistemas distribuidos, podemos distinguir diversos elementos necesarios para soportar los flujos. Por simplicidad, nos concentraremos en el flujo de datos almacenados, lo contrario al flujo de datos en vivo. En el último caso, los datos se capturan en tiempo real y se envían por la red hacia los destinatarios. La principal diferencia entre estos dos tipos de datos es que el flujo de datos en vivo brinda menos oportunidades de ajuste. De acuerdo con colaboradores (2001), podemos esquematizar una arquitectura general cliente-servidor para dar soporte a flujos continuos multimedia tal como indica



Arquitectura general para pasar a través de una red un flujo de datos multimedia almacenados

Ejemplo: un flujo complejo es el que se utiliza para transmitir una película. Tal flujo podría consistir en un solo flujo de video junto con dos flujos para la transmisión del sonido estereofónico de la película.



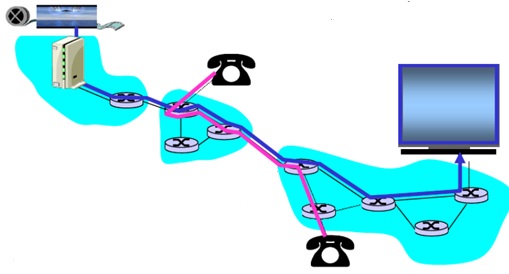
**Flujos y calidad del servicio:**

Los requerimientos de sincronización (y otros no funcionales) se expresan generalmente como requerimientos de Calidad del Servicio (QoS, por sus siglas en inglés). Estos requerimientos describen lo que se necesita del sistema distribuido subyacente y de la red para garantizar.

Mucho se ha dicho sobre cómo especificar la QoS requerida, a Jin y Nahrstedt, 2004 Desde la perspectiva de una aplicación, en muchos casos esto se reduce a especificar algunas propiedades importantes (Halsall, 2001)

* La velocidad de bits requerida a la que deben transportarse los datos
* El retraso máximo hasta que se haya configurado una sesión (es decir, cuando una aplicación puede comenzar el envío de datos).
* El retraso máximo fin a fin (es decir, cuánto tiempo le llevará a una unidad de datos llegar hasta un destinatario)
* La varianza del retraso máximo, o inestabilidad.
* El retraso máximo de un ciclo

Ejemplo: las relaciones temporales de un flujo puedan preservarse. La QoS para flujos continuos de datos tiene que ver principalmente con puntualidad, volumen, y confiabilidad. En esta sección veremos la QoS y su relación con la configuración de un flujo.



**Sincronización de flujos**

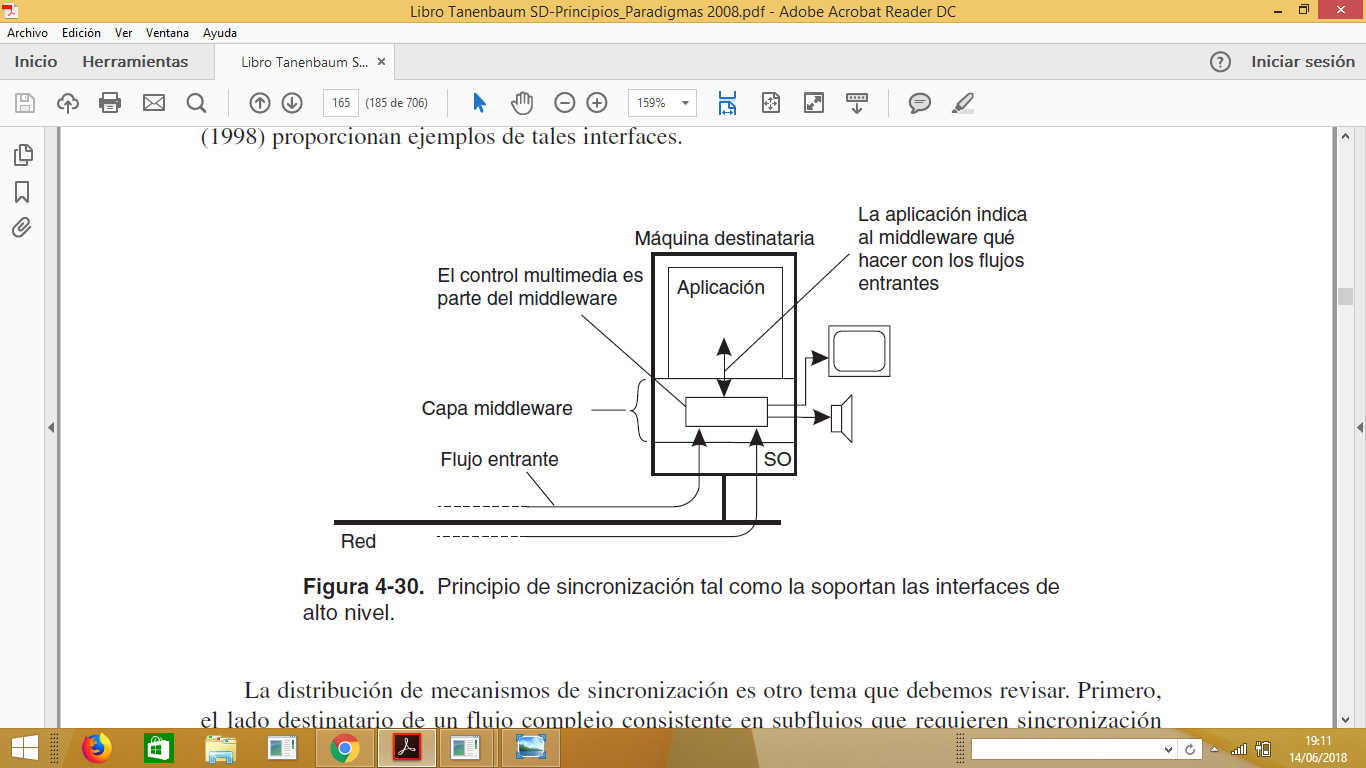
La sincronización de flujos tiene que ver con mantener las relaciones temporales entre flujos. Ocurren dos tipos de sincronización.

La forma de sincronización más sencilla sucede entre un flujo discreto de datos y un flujo continuo de datos.

Al mismo tiempo, el cliente debe reproducir una parte específica de un flujo de audio que coincida con la diapositiva en curso y también sea traído desde el servidor. En este caso, el flujo de audio se sincroniza con la presentación de diapositivas.

La sincronización ocurre al nivel de las unidades de datos que conforman el flujo. En otras palabras, podemos sincronizar dos flujos sólo entre unidades de datos. La elección de la unidad de datos depende en gran medida del nivel de abstracción con que se considere al flujo de datos. Para concretar, consideremos nuevamente un flujo de audio (de un solo canal) de la calidad de un disco compacto. Con la granulación más fina, tal flujo aparece como una secuencia de muestras de 16 bits. Con una frecuencia de muestreo de 44 100 Hz, la sincronización con otros flujos de audio podría, en teoría, ocurrir aproximadamente cada 23 μs. Para efectos estereofónicos de más alta calidad,

Es evidente que se necesita la sincronización a este nivel.



Principio de sincronización explícita al nivel de unidades de datos.

Ejemplo, consideremos una diapositiva mejorada con audio y mostrada en la

Web. La diapositiva se transfiere del servidor al cliente en forma de un flujo de datos discreto.

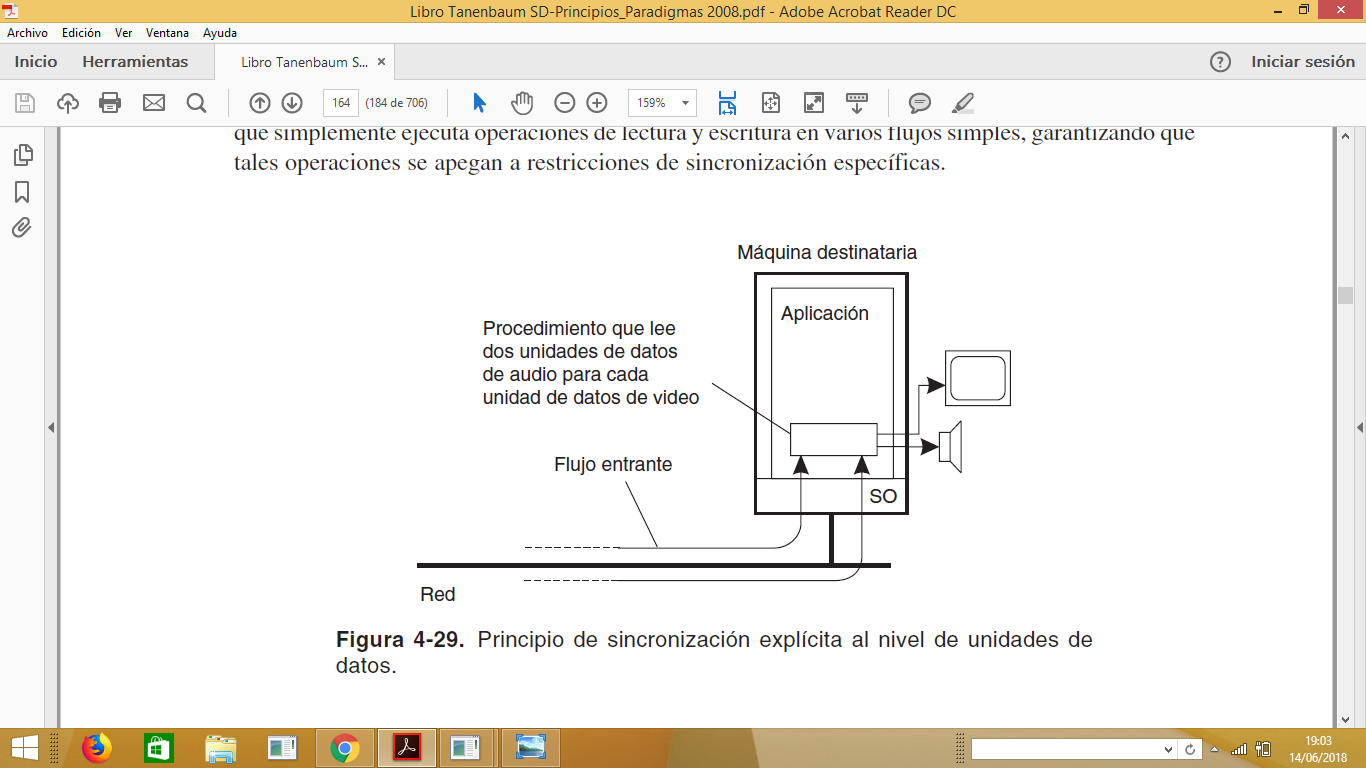


**Mecanismos de sincronización**

En este mecanismo debemos diferenciar dos puntos:

1. Los mecanismos básicos para sincronizar dos flujos,
2. La distribución de esos mecanismos en un ambiente de red.

Los mecanismos de sincronización pueden considerarse desde varios niveles de abstracción diferentes. En el nivel más bajo, la sincronización se realiza explícitamente operando las unidades de datos de flujos simples.



Principio de sincronización explícita al nivel de unidades de datos. Hay un proceso que simplemente ejecuta operaciones de lectura y escritura en varios flujos simples, garantizando que tales operaciones se apegan a restricciones de sincronización específicas. La sincronización cuando sólo tiene a su disposición herramientas de bajo nivel. Un mejor método es ofrecerle a una aplicación una interfaz que le permita controlar más fácilmente flujos y dispositivos. Volviendo a nuestro ejemplo, suponga que la reproducción del video tiene una interfaz de control que le permite especificar la velocidad a la cual deben aparecer las imágenes. Además, la interfaz ofrece la herramienta de registrar un controlador

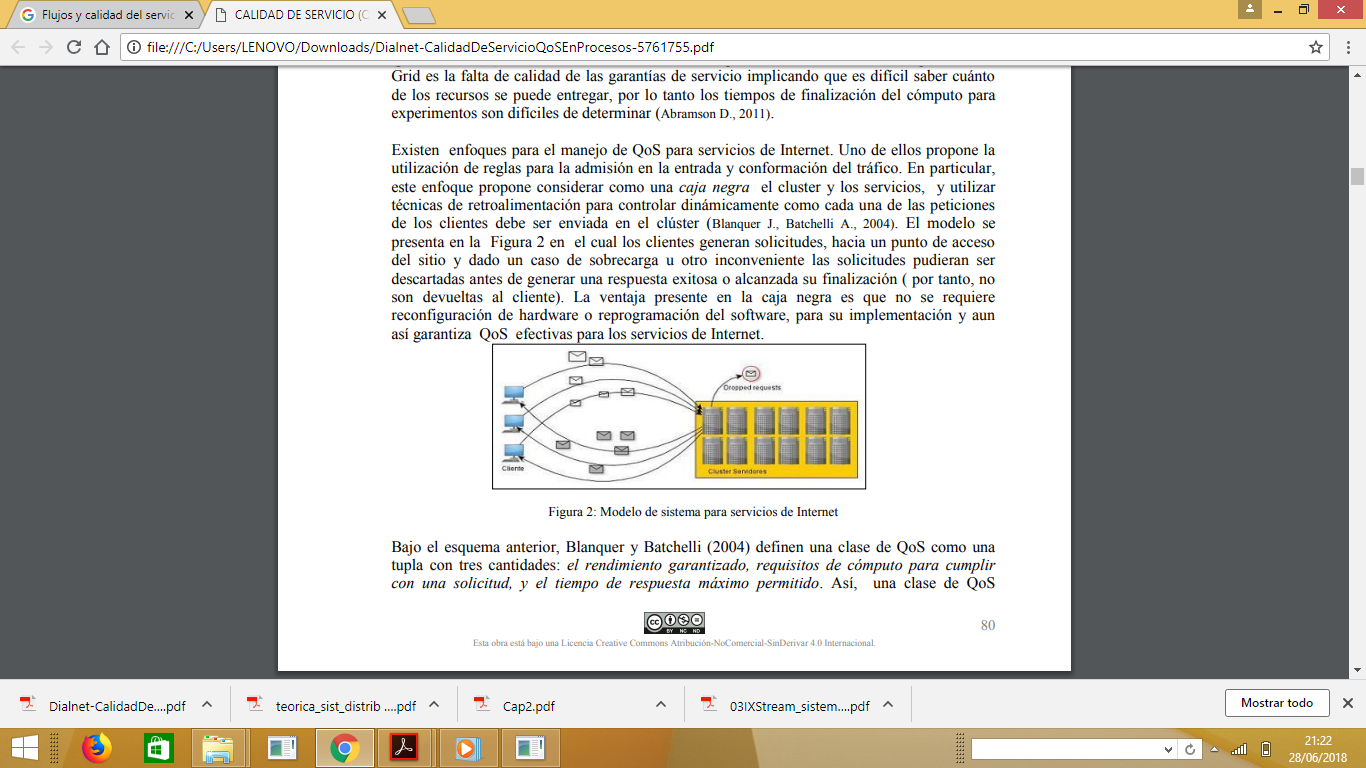


**Imponer de QoS:**

En el ámbito de las redes de comunicación LAN y WAN la QoS está asociada al manejo apropiado del tráfico de red, tal que se garantice la entrega de los paquetes de datos oportunamente. En ese sentido las redes de ordenadores están empezando a ofrecer garantías de calidad de servicio con respecto al retardo de paquetes y el ancho de banda de conexión. Estas garantías de QoS son de poca utilidad si no pueden extenderse a las aplicaciones que se ejecutan en los puntos finales

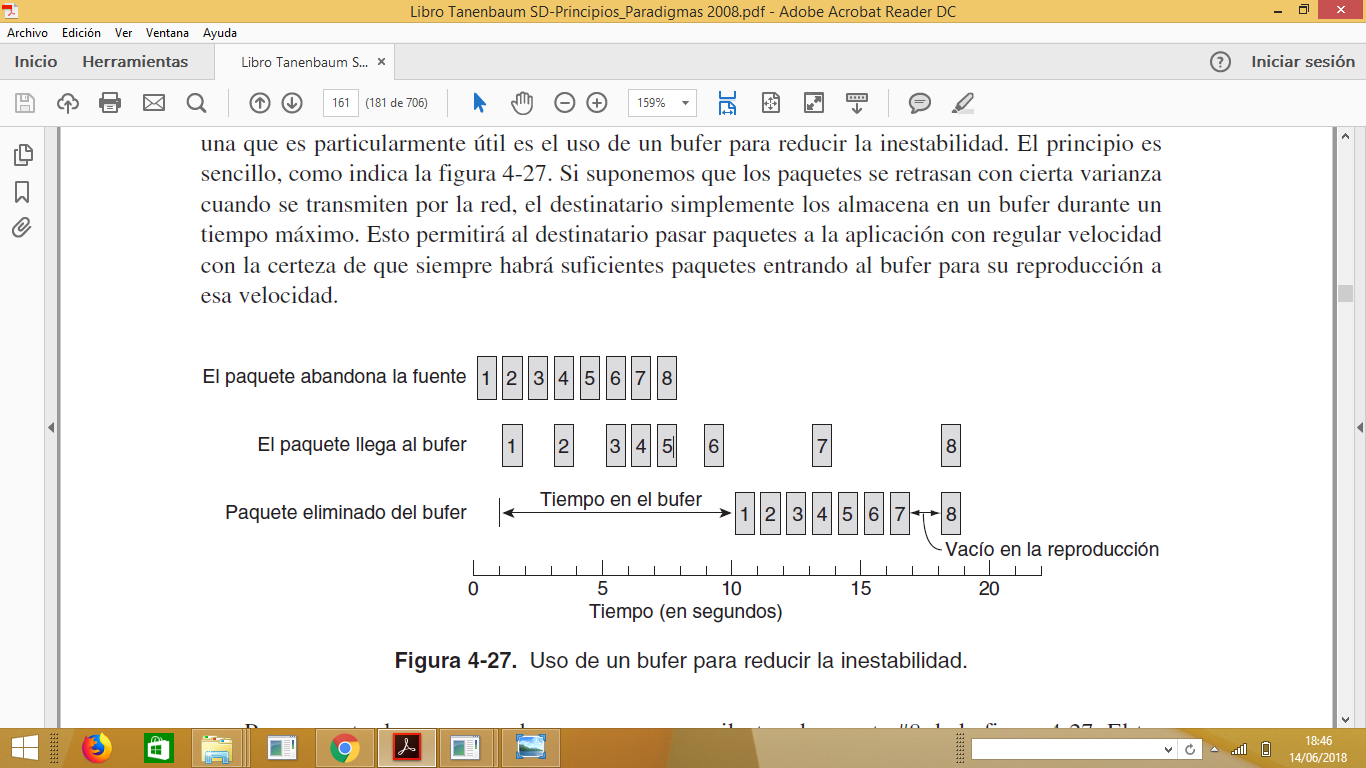
Por otro lado, dentro del modelo de calidad de servicios basado en asignación de recursos, la QoS aparece en formas diferentes como parte de las interfaces entre las capas de un sistema. En la interfaz entre una aplicación y el sistema operativo, el sistema se ve como un proveedor de servicios y la aplicación se considera como un cliente del servicio. La aplicación puede entonces solicitar una determinada QoS al sistema operativo (Hyden E.A., 1994). En otras palabras, el sistema operativo debe tener la capacidad de suministrar recursos del sistema a las solicitudes de manera que se logren los niveles deseados de rendimiento de forma predecible.

Por tanto, el reenvío garantizado define de manera efectiva un rango de prioridades que pueden asignarse a paquetes, y como tal permite a las aplicaciones diferenciar paquetes sensibles al tiempo de aquellos que no son críticos. Además de estas soluciones al nivel de red, un sistema distribuido también puede ayudar a que los destinatarios envíen información. Aunque generalmente no hay tantas herramientas disponibles, una que es Particularmente útil es el uso de un bufer para reducir la inestabilidad.



Modelo de sistema para servicios de Internet

<file:///C:/Users/LENOVO/Downloads/Dialnet-CalidadDeServicioQoSEnProcesos-5761755.pdf>



Uso de un búfer para reducir la inestabilidad

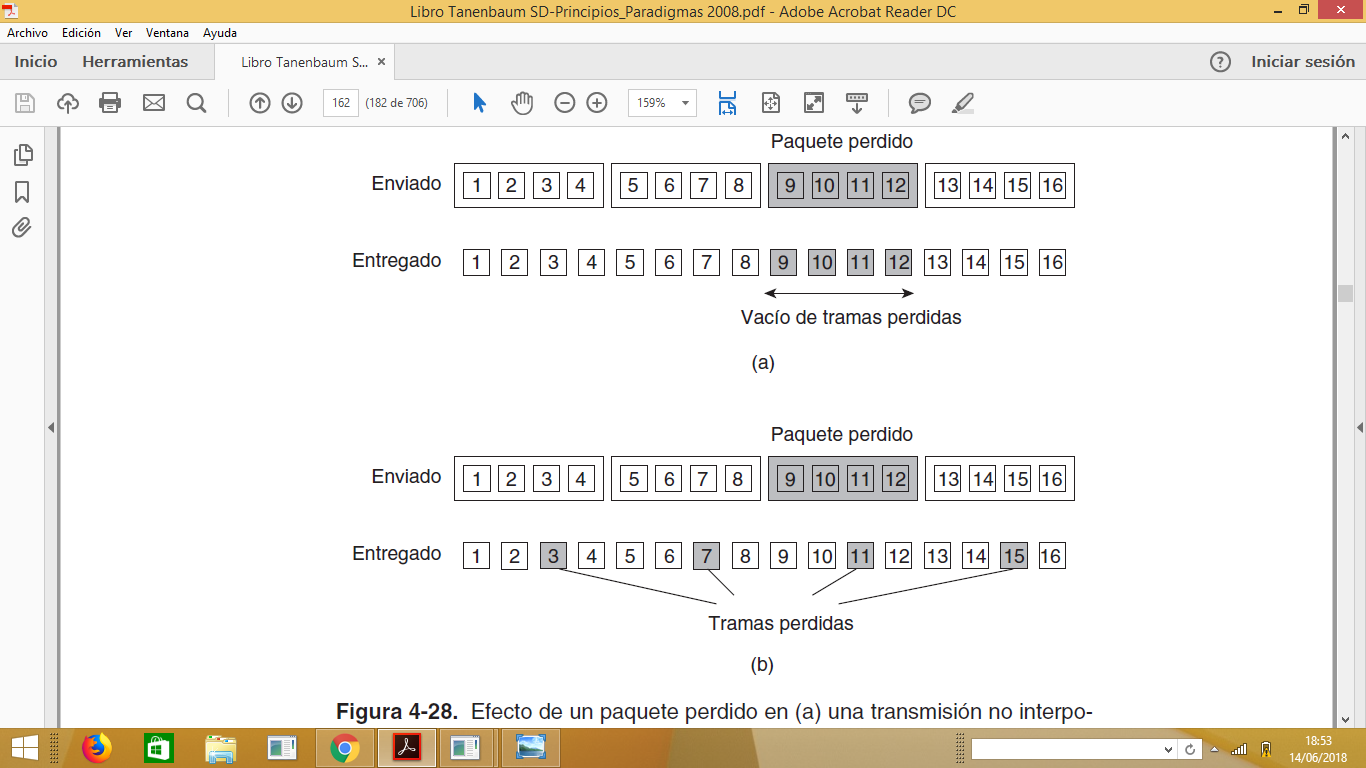
Ejemplo: internet proporciona recursos para diferenciar las clases de datos mediante sus servicios diferenciados.

Un servidor de envío puede marcar paquetes de salida como pertenecientes a una o a varias clases, incluyendo la clase de reenvío expeditoque esencialmente especifica que un paquete debe reenviarse mediante el ruteador actual con absoluta prioridad.



Si suponemos que los paquetes se retrasan con cierta varianza cuando se transmiten por la red, el destinatario simplemente los almacena en un búfer durante un tiempo máximo. Esto permitirá al destinatario pasar paquetes a la aplicación con regular velocidad con la certeza de que siempre habrá suficientes paquetes entrando al bufer para su reproducción a esa velocidad.

Un problema que puede presentarse es que un solo paquete contenga diversas tramas de audio y video. En consecuencia, cuando se pierde un paquete, el destinatario puede percibir un gran vacío cuando reproduzca las tramas. Es posible sortear este efecto interpolando las tramas, como ilustra



Efecto de un paquete perdido en (a) una transmisión no interpolada, y en (b) una transmisión interpolada

De esta manera, cuando se pierde un paquete, el vacío resultante en tramas

Posteriores se distribuye en el tiempo. Sin embargo, observe que este método requiere de un bufer destinatario más grande, en comparación con la no

Conclusiones:

* Al haber analizado los conceptos presentado en este trabajo pudimos llegar a la conclusión de que la comunicación orientada a nos sirve para entender los diferentes concepto y usos que se dan dentro de las organizaciones y se convierten en una importante al momento que enviamos un paquetes de mensajes a otro usuario que nos ayuda a poder garantizar que los datos serán enviados correctamente sin errores y en el mismo orden en que se transmitieron.
* Otros autores que han estudiado la QoS y AOP Engvig (2005) describe como se han realizado aplicaciones experimentales de QoS sobre casos de estudio en aplicaciones como Cálculo de PI o transmisiones distribuidas de reproductor de audio. Se hace foco en el desarrollo de componentes mediante arquitectura QuA para aplicaciones sensibles a la QoS.

**Comunicación de sistemas distribuidos**

**Comunicación por Multritransmisión (multicast)**

**Factores de comunicación**

Los diferentes mecanismos de comunicación se caracterizan por los siguientes factores:

* Rendimiento: Latencia, radio de transferencia, ancho de banda
* Escalabilidad: Número de elementos activos
* Fiabilidad: Perdida de mensajes
* Seguridad: Cifrado, certificación
* Movilidad: Equipos móviles
* Calidad de Servicio: Reserva y garantía de anchos de banda
* Comunicación en grupo: Multitransmision (Multicast)

Entre los diferentes tipos de middleware en Sistemas Distribuidos tenemos:

* Llamadas a procedimientos remotos
* Comunicación orientada a mensajes
* Comunicación orientada a flujos
* Comunicación por Multitransmision

Considerando que el middleware representa una capa de Software que oculta la heterogeneidad de redes subyacentes, Sistemas operativos y Lenguajes de programación.

**COMUNICACIÓN POR MULTITRANSMISION**

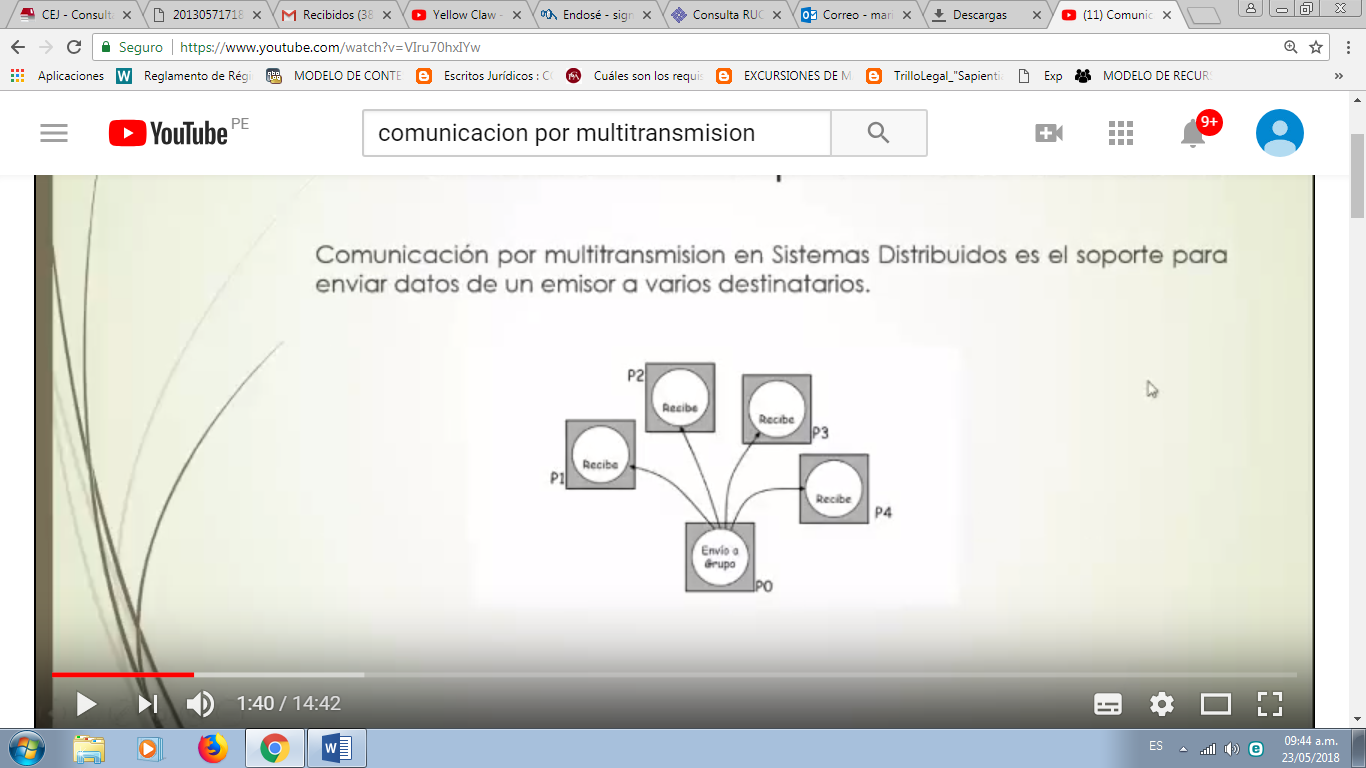
**FUNCIONAMIENTO**

Comunicación por multitransmisión de Sistemas Distribuidos es el soporte para enviar datos de un emisor o varios destinatarios.

Es importante en la comunicación de sistemas distribuidos, por el soporte para enviar datos a varios destinatarios, este aspecto a pertenecido al dominio de los protocolos de red.

Con la llegada de la tecnología punto a punto se volvió más sencillo configurar rutas de comunicación, debido a que se utilizan típicamente en la capa de aplicación.

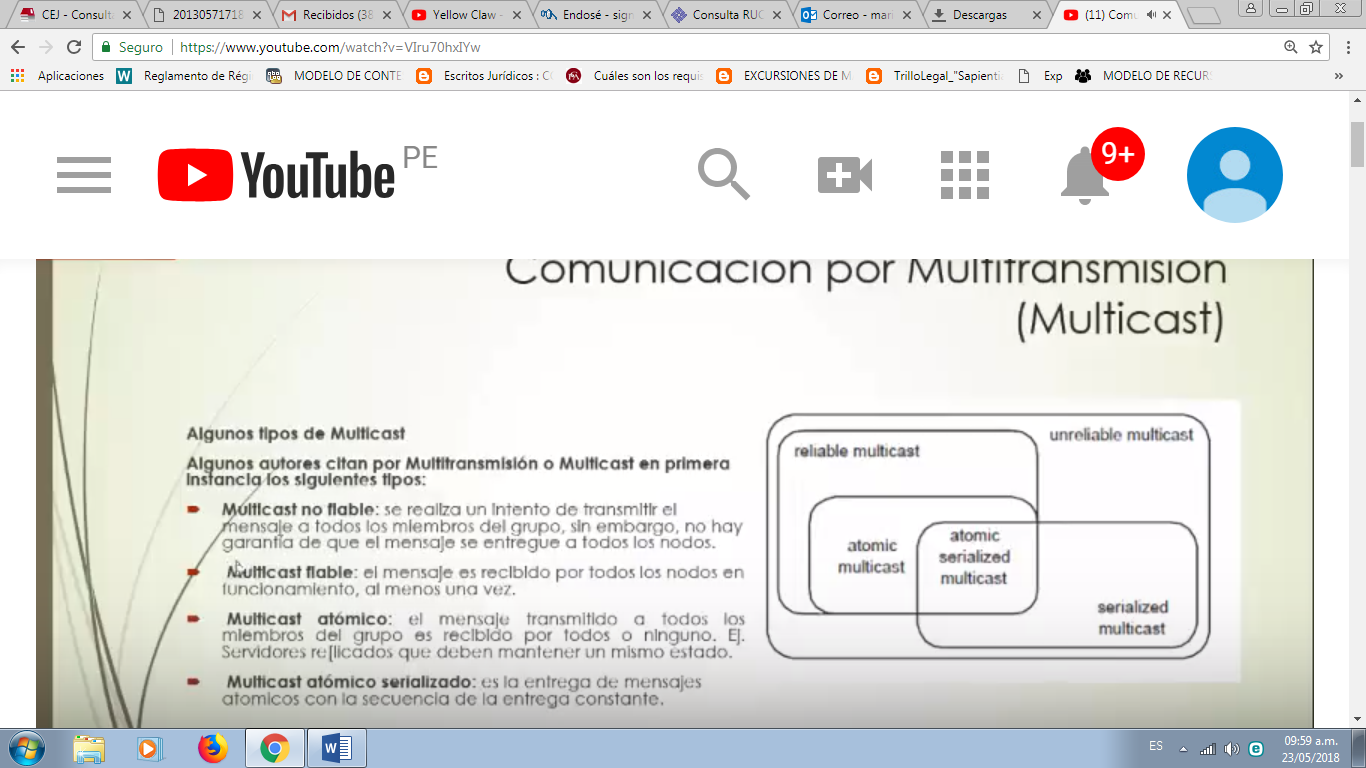
La comunicación por multitrasmisión también puede lograrse de maneras diferentes a la configuración explicita de rutas de comunicación.



**ALGUNOS TIPOS DE MULTICAST:**

Algunos autores citan por Multitransmisión o Multicast en primera instancia los siguientes tipos:

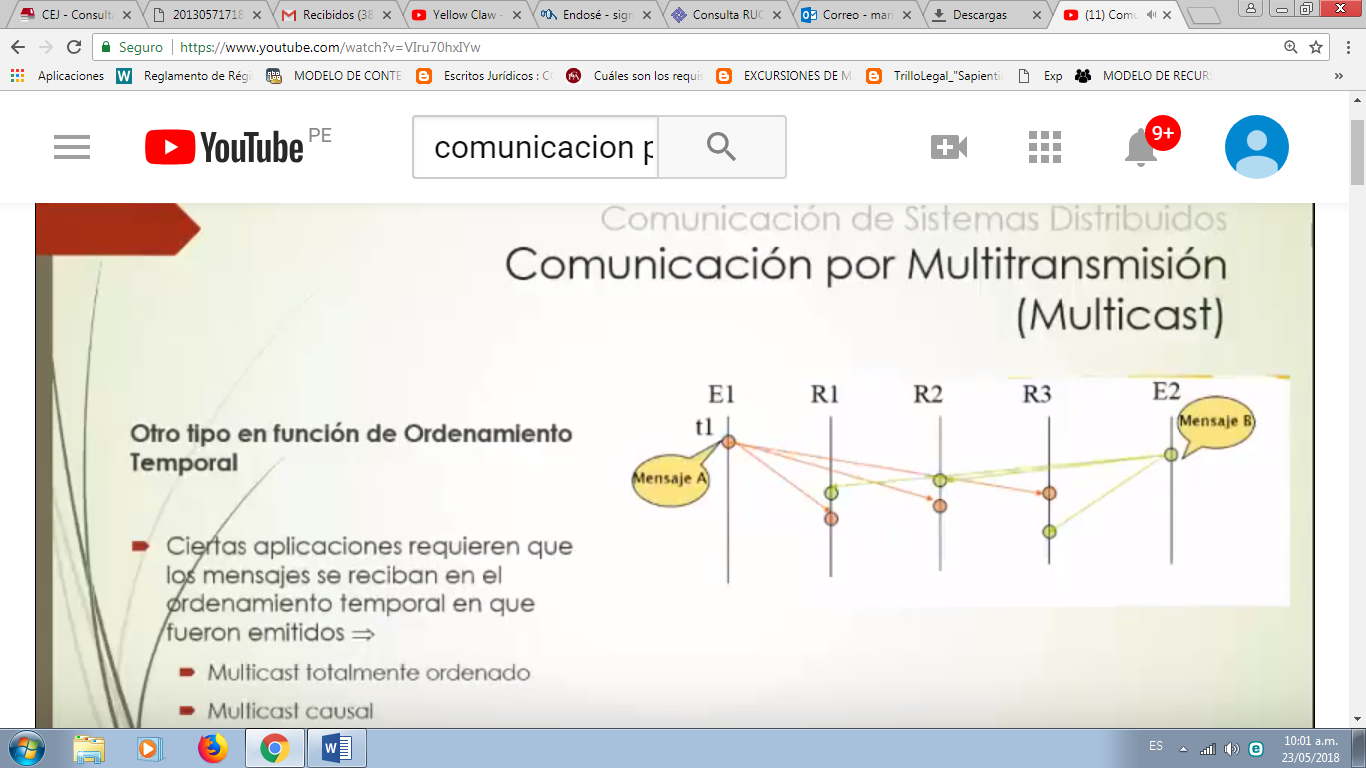
* **Multicast no flable:** se realiza un intento de transmitir el mensaje a todos los miembros del grupo. Sin embargo, no hay garantía de que el mensaje se entregue a todos los nodos.
* **Multicast flable:** el mensaje es recibido por todos los nodos en funcionamiento, al menos una vez.
* **Multicast atomico:** el mensaje transmitido a todos los miembros del grupo es recibido por todos o ninguno. Ej. Servidores relicados que deben mantener un mismo estado.
* **Multicast atómico serializado:** es la entrega de mensajes atómicos con la secuencia de la entrega constante.



1. **OTRO TIPO EN FUNCIÓN DE ORDENAMIENTO TEMPORAL**

Ciertas aplicaciones requieren que los mensajes se reciban en el ordenamiento temporal en que fueron emitidos:

* Multicast totalmente ordenado
* Multicast causal



1. **MULTITRANSMISION A NIVEL DE APLICACIÓN:**

Una idea de la multitransmición al nivel de aplicación es que los nodos se organizan en una **red sobrepuesta**, la cual después se utiliza para diseminar la información a sus miembros. (Tanenbaum A., Van Steen M - 2008)

Una observación importante es que los ruteadores de red no están organizados en grupos de miembros. En consecuencia, las conexiones de enrutamiento dentro de la red sobrepuesta pueden o no ser óptimos en comparación a un enrutamiento a nivel de red.

Básicamente existen 2 métodos para el desarrollo del diseño de la red sobrepuesta.

1. Primero, los nodos pueden organizarse por sí mismos de manera directa de un árbol, por ende significaría que existe una única ruta (SOBREPUESTA) entre cada par de nodos.
2. Un método alterno es que los nodos se organicen en una red acoplada en la que cada nodo tendrá varios vecinos y en general existían múltiples rutas entra cada par de nodos.

La principal diferencia entre los 2, es que generalmente el último proporciona mayor fuerza: si una conexión falla (digamos, porque un nodo falla), aun existiría una oportunidad de diseminar la información sin tener que organizar de inmediato toda la red sobrepuesta.

Ahora se utilizan técnicas de multitransmisión a nivel de aplicación que pueden ser:

* Arboles de Intercambio
* Gossiping.

**CONTRUCCION SOBREPUESTA**

Resulta ser que la construcción de un árbol resulta ser nada difícil, sin embargo construir un árbol eficiente es otra historia muy distinta

**CALIDAD DE UN ÁRBOL**

**MÉTODO I:** **ÁRBOL**

Donde los nodos pueden organizarse por sí mismos de manera directa en un árbol, por tanto existe una ruta única entre cada para los nodos. La calidad de un árbol de multitransmisión se mide por tres métricas:

* **Tensión del vínculo:** Mide la frecuencia con que un paquete cruza un mismo vínculo.

Proviene que aunque a nivel lógico un paquete puede reenviarse a lo largo de dos conexiones diferentes, parte de esas conexiones puede en realidad corresponder al mismo vínculo físico.

Mide la frecuencia con que un paquete cruza un mismo vínculo.

En nuestra gráfica se puede notar que cuando A multidifunda un mensaje a los otros nodos, este menaje atravesará dos veces los vínculos:

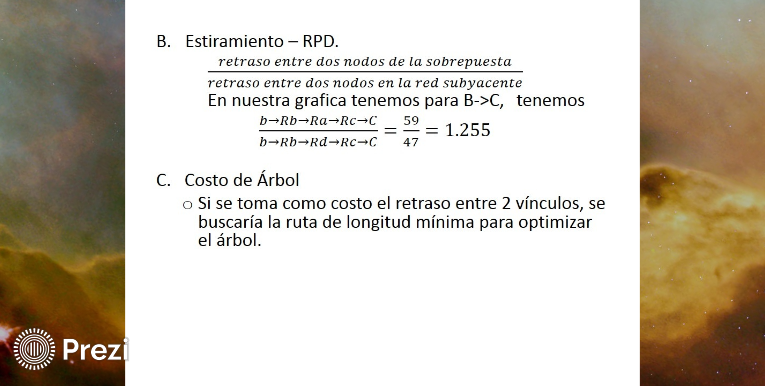
<R,Rb> <Ra,Rb> <Rc,Rd> <D,Rd>

Si en vez de haber construido un vínculo sobrepuesto entre B y D, se hubiese hecho entre Ay C se habría ahorrado el paso doble de los vínculos:

<Ra,Rb> Rc,Rd>

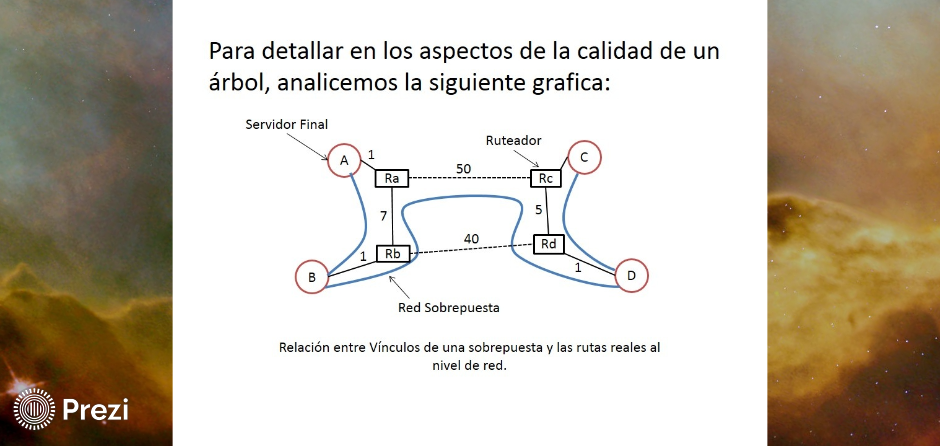
Así se tendría un árbol más óptimo

* **Estiramiento o Castigo Relativo por Retraso (RPD):** mide la relación que hay en el retraso entre dos nodos de la red sobrepuesta y el retraso que esos dos nodos experimentan en la red subyacente.



* **Costo del árbol:** la optimización global de retrasos en relación a los vínculos.

Para detallar en los aspectos de la calidad de un árbol, analicemos la siguiente gráfica:



1. **METODO 2: DISEMINACION DE DATOS BASADA EN EL GOSSIP:**

* Se basa en el comportamiento epidémico,
* El objetivo principal es propagar rápidamente la información entre un gran grupo de nodos, utilizando solo información local.
* En otras palabras, no hay un componente central mediante el cual se coordine la diseminación de información.

**MODELOS EPIDEMICO**

El objetivo principal de los protocolos epidémicos es propagar rápidamente información entre una gran colección de nodos utilizando solo información local.

**TIPOS DE NODOS EN UN SISTEMA DISTRIBUIDO:**

* Infectado, Mantiene información que vale la pena propagar.
* Susceptible, Nodo que aún no ha visto la información.
* Eliminado, Nodo infectado que no está dispuesto o no es capaz de propagar la información.

**MODELO DE ANTIENTROPIA**

En este modelo, un nodo P elige al azar a otro nodo Q, y posteriormente intercambia actualizaciones con Q. Existen tres métodos para intercambiar actualizaciones.

1. P solo empuja sus propias actualizaciones hacia Q.
2. P sólo jala nuevas actualizaciones desde Q.
3. P y Q se envían actualizaciones entre sí.

**GOSSIPING**

Funciona de la siguiente manera. Si el nodo P se acaba de actualizar con el elemento de datos X contacta a cualquier otro nodo Q e intenta empujar la actualización en Q.

Sin embargo, es posible que q ya haya sido actualizado por otro nodo, en este caso P podría perder el interés en propagar la actualización nuevamente. En otras palabras, la actualización se elimina.

Una ventaja de los modelos epidémicos es su escalabilidad.

Propagar la eliminación de un dato es difícil. Cuando un dato es eliminado de un Nodo, dicho nodo recibirá en algún momento viejas copias del dato y la interpretara como actualizaciones.

**IMPLEMENTACIÓN**

* Para realizar la implementación es tener en cuenta el hardware y el desempeño de cada elemento.
* Para comunicarnos mediante mensajes podemos crear una red especial para que el mensaje enviado llegue a todas las máquinas que están conectadas y que estén escuchando la red.
* La multitransmisión se la puede realizar siempre teniendo una red multicast, por un emisor que envía paquetes a cada una de las máquinas que están conectadas, esta implantación se la realizada en una comunicación pequeña empresa.ta teniendo una dirección
* Se implementa teniendo una dirección en cada paquete que va ser enviado y luego va ser direccionado a las distintas máquinas, por la cual no es tan eficiente pues debe verificarse si ha llegado mediante software.
* Se implementa en una base de datos distribuidos, con lo que con frecuencia debe enviar un mensaje a todos las demás máquinas conectadas.
* La multitransmision está implementada en video bajo demanda ( VoD) hace uso de técnicas de transmisión por multitransmision (multicast), permite compartir recursos entra un número elevado de clientes.

**USO Y APLICACIONES**

Dentro de las aplicaciones o usos tenemos la propagación de actualizaciones, esto lo podemos observar en un servidor de antivirus que la misma descarga las actualizaciones y mediante la consola propaga la misma hacia las máquinas conectadas al mismo.

También en sistemas modulares al actualizar módulos, dichas actualizaciones las podemos propagar mediante esta técnica.

La multitransmision nos permite dividir las tareas de copia de seguridad en varias subtareas (flujos) que se ejecutan simultáneamente y que envían los datos al destino (dispositivo de cintas o dispositivo de sistemas de archivos).

La multitransmisiones útil cuando se ejecutan tareas de copia de seguridad grandes.

Un sistema que me permite realizar actualizaciones mediante la comunicación por transmisión, es CA TECHNOLOGIES.

**APLICACIÓN DE MULTICAST:**

* Búsqueda de un recurso.
* **Tolerancia a fallos;** un servicio de replicado se compone de varios servidores que forman un grupo (grupo de nodos). Las solicitudes del cliente son tratados por todos los servidores permitiendo que algunos servidores caigan in afectar la disponibilidad del servicio.
* **Actualizaciones múltiples;** como son los servidores de antivirus, hace referencia a que el servidor consola actualiza las nuevas definiciones de virus a todos sus nodos clientes.

La velocidad del envío multicast depende del algoritmo utilizado y del soporte hardware disponible.

**Multicast con el ejemplo de configuración de los reguladores (WLCs) y de los Puntos de acceso ligeros del Wireless LAN (revestimientos)**

Este documento proporciona un ejemplo de configuración sobre cómo configurar controladores de LAN inalámbricos (WLC) y Lightweight Access Points (LAP) para la multidifusión y comunicación con una red alámbrica habilitada de multidifusión.

### Componentes Utilizados

La información que contiene este documento se basa en las siguientes versiones de software y hardware.

* WLC de Cisco 4400 que funciona con la versión de firmware 4.0
* Cisco 1000 Series LAP
* Adaptador de red inalámbrica de cliente de Cisco 802.11a/b/g que funciona con la versión de firmware 2.6
* Cisco 2500 Router que funciona con la versión del Cisco IOS ® Software 12.4(2)
* Dos 3500 XL Series Switch de Cisco que funcionan con el Cisco IOS Software Release 12.0(5)WC3b

## Configuración de la red

En esta configuración, la red alámbrica se comprende del tres Routers, del r1, del r2 y del R3, ese funcionamiento OSPF entre ellos.

Los host atados con alambre conectan con la red a través de un 2 Switch de la capa que esté conectado con el r1 del router. La red inalámbrica conecta con la red a través del router R3, tal y como se muestra en del [diagrama](https://www.cisco.com/c/es_mx/support/docs/wireless-mobility/wireless-lan-wlan/81671-multicast-wlc-lap.html#network).

Los dispositivos necesitan ser configurados para la conectividad de IP básica y habilitan la multidistribución en la red. Por lo tanto, los usuarios pueden enviar y recibir el tráfico Multicast de la cara tela al lado de la Tecnología inalámbrica y vice versa.

Este documento utiliza estos IP Addresses para el WLC, el REVESTIMIENTO y los clientes de red inalámbrica:

WLC Management Interface IP address: 172.16.1.30/16

WLC AP Manager Interface IP address: 172.16.1.31/16

LAP IP address: 172.16.1.50/16

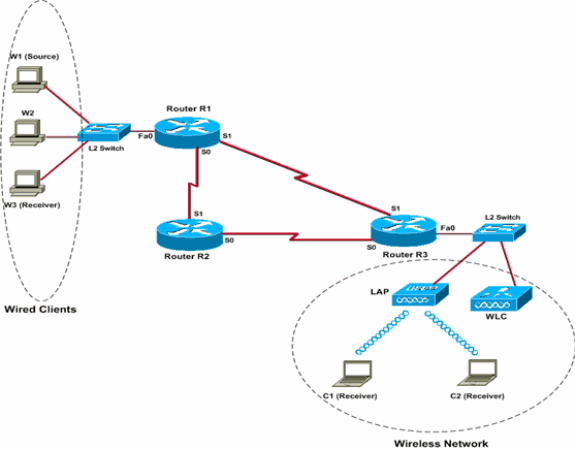
Wireless Client C1 IP address: 172.16.1.75/16

Wireless Client C2 IP address: 172.16.1.76/16

Wired Client W1 IP address: 192.168.0.20/16

Wired Client W2 IP address: 192.168.0.30/16

Wired Client W3 IP address: 192.168.0.40/16

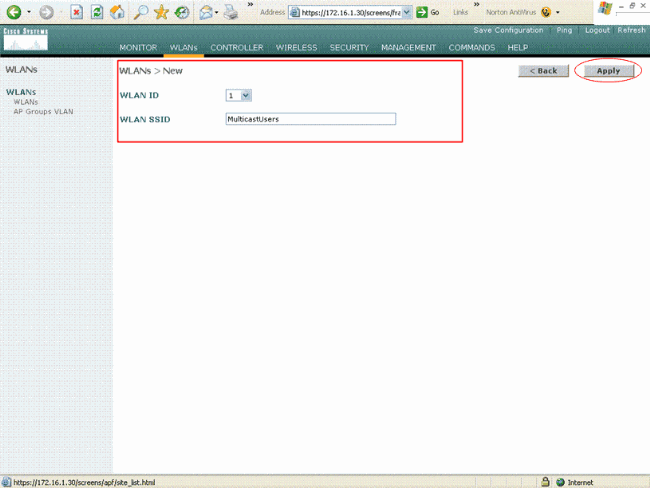


#### Configure la red inalámbrica (WLAN) para los clientes

El primer paso es crea una red inalámbrica (WLAN) a la cual los clientes de red inalámbrica puedan conectar y recibir el acceso a la red. Complete estos pasos para crear una red inalámbrica (WLAN) en el WLC:

1. Haga clic los **WLAN del** regulador GUI para crear una red inalámbrica (WLAN).
2. Haga clic **nuevo** para configurar una nueva red inalámbrica (WLAN).

En este ejemplo, la red inalámbrica (WLAN) se nombra MulticastUsers y el ID DE WLAN es



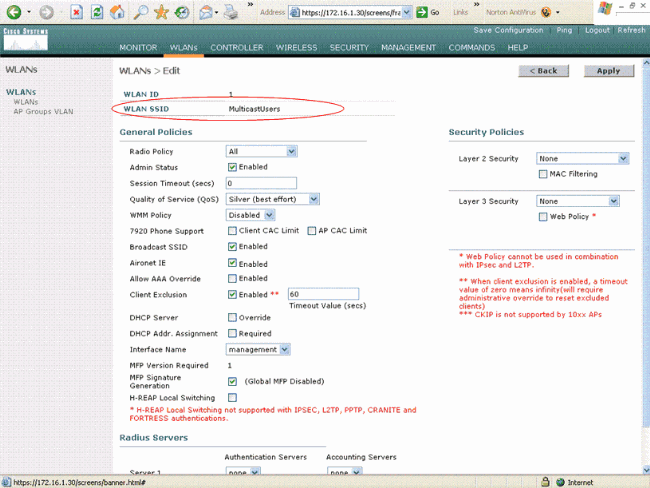
1. Haga clic en Apply (Aplicar).
2. En la red inalámbrica (WLAN) > edite la ventana, definen los parámetros específicos a la red inalámbrica (WLAN).
   * 1. Para la red inalámbrica (WLAN), elija la interfaz apropiada del campo de nombre de la interfaz.

Este ejemplo asocia la interfaz de administración a la red inalámbrica (WLAN).

* + 1. Seleccione los otros parámetros, que depende de los requisitos de diseño.

Los valores predeterminados se utilizan en este ejemplo.

* + 1. Haga clic en Apply (Aplicar)



En este ejemplo, acode 2 métodos de seguridad para autenticar a los usuarios de red inalámbrica no se utilizan. Por lo tanto, no elija ninguno en el campo de Seguridad de la capa 2. Por abandono, la opción de seguridad de la capa 2 es 802.1x.

En vez de asociar la red inalámbrica (WLAN) (SSID) a la interfaz de administración, las interfaces dinámicas se pueden también configurar en el WLC para dividir a los usuarios de red inalámbrica en segmentos y la red inalámbrica (WLAN) se puede asociar a las interfaces dinámicas. Refiera a los [VLA N en el ejemplo de configuración de los reguladores del Wireless LAN](https://www.cisco.com/en/US/tech/tk722/tk809/technologies_configuration_example09186a00805e7a24.shtml) para la información sobre cómo configurar las interfaces dinámicas en el WLCs.

Publique estos comandos para configurar los WLAN en el WLC usando el CLI:

1. Publique los config wlan crean <wlan-identificación > <wlan-nombre > comando para crear una nueva red inalámbrica (WLAN). Para la WLAN-identificación, ingrese un ID a partir de la 1 a 16. Para el WLAN-nombre, ingrese un SSID hasta 31 caracteres alfanuméricos.
2. Publique el permiso wlan de los config <wlan-identificación > comando para habilitar una red inalámbrica (WLAN).

Por el ejemplo en este documento, los comandos son:

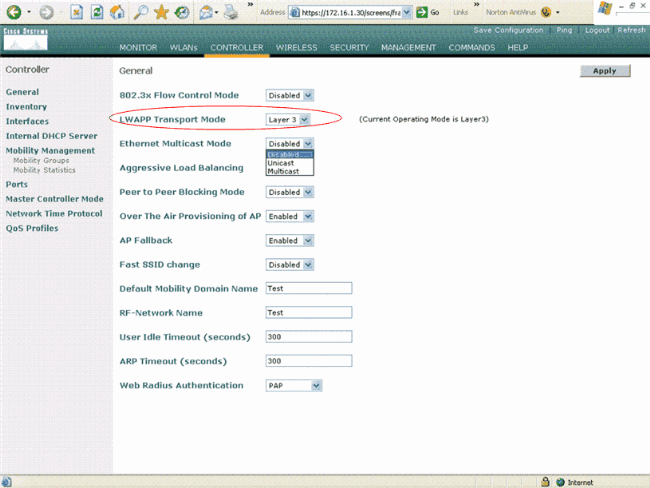
config wlan create 1 MulticastUsers

config wlan enable 1

El siguiente paso es configurar el WLC para multicasting. Complete estos pasos:

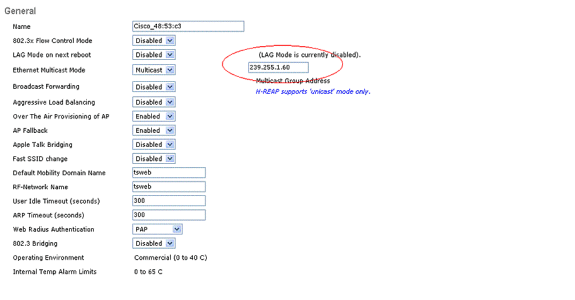
1. De la página web del general de regulador, asegúrese de que fijen al modo de transporte LWAPP para acodar 3.

La característica del funcionamiento del Multicast trabaja solamente en este modo.



1. Cuando el Multicast se habilita como unicast del Multicast, los paquetes se replican para cada AP; éste puede ser hace un uso intensivo del procesador, así que utilícelo con cautela. El Multicast habilitado como Multicast del Multicast utiliza a la dirección Multicast asignada usuario para hacer un Multicast más tradicional hacia fuera a los AP.
2. Del menú desplegable para el modo de multidifusión de los Ethernetes, elija el Multicast y ingrese a una dirección de grupo de multidifusión.

En este ejemplo, el direccionamiento es 239.255.1.60



Haga clic en Apply (Aplicar).

El WLC 4100 no soporta el modo del Multicast. El Multicast se hace solamente en el modo unidifusión. Esto significa que el regulador tiene que replicar el paquete de multidifusión para cada AP y el unicast el paquete de multidifusión a cada uno de los AP.

Publique estos comandos para habilitar el Multicast con el CLI:

1. De la línea de comando, publique el **comando enable global del Multicast de la red de los config**.
2. De la línea de comando, publique el <multicast-grupo-IP-**direccionamiento > el** comando del **Multicast del modo del Multicast de la red de los config**.

Por el ejemplo en este documento, los comandos son:

**config network multicast global enable**

**config network multicast mode multicast 239.255.1.60**

### Configure la red alámbrica para Multicasting

Para configurar la red alámbrica para esta configuración, usted necesita configurar la Conectividad de los routeres para básica y habilitar la multidistribución en la red alámbrica.

Según lo mencionado anterior, el OSPF se utiliza como el Unicast Routing Protocol.

Cualquier Multicast Protocol se puede utilizar en la red alámbrica. Este documento utiliza el PIM-DM como el Multicast Protocol. Refiera a la [guía de configuración del Multicast IP del Cisco IOS](https://www.cisco.com/en/US/products/ps6350/products_installation_and_configuration_guides_list.html) para información detallada sobre los diversos protocolos que se pueden utilizar para multicasting en una red alámbrica.

Éstos son el r1, r2 y R3 de las configuraciones para Routers

**ROUTER 1**

RouterR1#**show run**

Building configuration...

Current configuration : 836 bytes

!

version 12.2

service timestamps debug uptime

service timestamps log uptime

no service password-encryption

!

hostname RouterR1

!

!

ip subnet-zero

!

ip multicast-routing

!--- Enables IP Multicasting on the network.

!

!

!

interface Ethernet0

ip address 192.168.0.1 255.255.0.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

ip cgmp

!--- Enables Cisco Group Management Protocol (CGMP) on the interface !--- connected to the Layer 2 switch.

!

interface Serial0

description Connected to RouterR2

ip address 10.2.3.2 255.255.255.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

!

interface Serial1

description Connected to RouterR3

ip address 10.2.4.1 255.255.255.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

!

interface Serial2

no ip address

shutdown

!

interface Serial3

no ip address

shutdown

!

interface BRI0

no ip address

encapsulation hdlc

shutdown

!

router ospf 1

!--- Configures OSPF as the unicast routing protocol.

log-adjacency-changes

network 10.0.0.0 0.255.255.255 area 0

network 192.168.0.0 0.0.255.255 area 0

!

ip classless

ip http server

!

!

!

line con 0

line aux 0

line vty 0 4

!

end

**ROUTER 2**

RouterR2#**show run**

Building configuration...

Current configuration : 616 bytes

!

version 12.2

service timestamps debug uptime

service timestamps log uptime

no service password-encryption

!

hostname RouterR2

!

!

ip subnet-zero

!

ip multicast-routing

!--- Enables IP Multicasting on the network.

interface Ethernet0

no ip address

shutdown

!

interface Serial0

description Connnected to RouterR3

ip address 10.2.2.2 255.255.255.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

!

interface Serial1

description Connected to RouterR1

ip address 10.2.3.1 255.255.255.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

!

router ospf 1

!--- Configures OSPF as the unicast routing protocol.

log-adjacency-changes

network 10.0.0.0 0.255.255.255 area 0

!

ip classless

ip http server

!

!

!

line con 0

line aux 0

line vty 0 4

!

end

**ROUTER 3**

RouterR3#**show run**

Building configuration...

Current configuration : 711 bytes

!

version 12.2

service timestamps debug datetime msec

service timestamps log datetime msec

no service password-encryption

!

hostname RouterR3

!

!

ip subnet-zero

!

ip multicast-routing

!--- Enables IP Multicasting on the network.

!

!

!

interface Ethernet0

ip address 172.16.1.1 255.255.0.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

ip cgmp

!--- Enables Cisco Group Management Protocol (CGMP) on the interface !--- connected to the Layer 2 switch.

!

interface Serial0

description Connected to RouterR2

ip address 10.2.2.1 255.255.255.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

!

interface Serial1

description Connected to RouterR1

ip address 10.2.4.2 255.255.255.0

ip pim dense-mode

!--- Enables PIM-Dense Mode Multicast Protocol on the interface.

!

router ospf 1

!--- Configures OSPF as the unicast routing protocol.

log-adjacency-changes

network 172.16.0.0 0.0.255.255 area 0

network 10.0.0.0 0.255.255.255 area 0

!

ip classless

ip http server

!

!

!

!

line con 0

line aux 0

line vty 0 4

!

end

Para los 2 Switch de la capa, no se requiere ninguna configuración para multicasting. Todos los 2 Switch de la capa basada en IOS tienen CGMP habilitado por abandono. Por lo tanto, el Switches procesa automáticamente los mensajes CGMP del Routers.

## Verificación y resolución de problemas

Use esta sección para confirmar que su configuración funciona correctamente.

[La herramienta Output Interpreter Tool](https://www.cisco.com/cgi-bin/Support/OutputInterpreter/home.pl) ([clientes registrados solamente](https://tools.cisco.com/RPF/register/register.do)) (OIT) soporta ciertos comandos show. Utilice la OIT para ver un análisis del resultado del comando show.

Para verificar la configuración, usted necesita enviar el tráfico Multicast de la fuente W1 y marcar si el tráfico Multicast atraviesa la red alámbrica y alcanza a los miembros atados con alambre y inalámbricos del grupo, W2, c1 y C2.

Realice esta tarea para probar si el Multicast IP se configura correctamente en su red.

Si todo el Routers del multicast capaz es miembros de un grupo de multidifusión, haciendo ping que el grupo hace a todo el Routers responder, que pueden ser administrativa y una herramienta de debugging útiles.

Otra razón para tener un router unirse a un grupo de multidifusión es cuando otros host en la red tienen una configuración del Interior Gateway Routing Protocol (IGRP) que evite que contesten correctamente a las interrogaciones IGMP. Cuando usted tiene el router unirse a al grupo de multidifusión, éste hace a los routeres ascendentes mantener la información del tabla de Multicast Routing para ese grupo y guardar las trayectorias para ese active del grupo. Para configurar a un router para ser parte del grupo de multidifusión, publique este comando del modo de configuración de la interfaz:

ip igmp join-group <group-address>

Example: Router(config-if)#ip igmp join-group 239.255.1.60

Aquí está la salida del ping del router R3:

RouterR3#ping 239.255.1.60

Type escape sequence to abort.

Sending 1, 100-byte ICMP Echos to 239.255.1.60, timeout is 2 seconds:

Reply to request 0 from 10.2.2.2, 40 ms

Reply to request 0 from 10.2.3.1, 84 ms

Reply to request 0 from 10.2.4.1, 44 ms

Localización de un salto defectuoso

Realice esta tarea para monitorear y diagnosticar una configuración básica del Multicast IP. Usted puede utilizar este procedimiento cuando un receptor y una fuente no actúan como se esperaba.



Aquí están las salidas de la **calidad de miembro del igmp del IP de la demostración** y los **comandos show ip mroute count** para el ejemplo de configuración. Estas salidas fueron tomadas del router R3.

RouterR3#**sh ip igmp membership**

Flags: A - aggregate, T - tracked

L - Local, S - static, V - virtual, R - Reported through v3

I - v3lite, U - Urd, M - SSM (S,G) channel

1,2,3 - The version of IGMP the group is in

Channel/Group-Flags:

/ - Filtering entry (Exclude mode (S,G), Include mode (\*,G))

Reporter:

<ip-address> - last reporter if group is not explicitly tracked

<n>/<m> - <n> reporter in include mode, <m> reporter in exclude

Channel/Group Reporter Uptime Exp. Flags Interface

\*,224.0.1.40 10.2.2.1 1d21h stop 2LA Se0

\*,239.255.1.60 172.16.1.1 1d06h 02:17 1LA Et0

RouterR3#**sh ip mroute count**

IP Multicast Statistics

5 routes using 3094 bytes of memory

2 groups, 1.50 average sources per group

Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second

Other counts: Total/RPF failed/Other drops(OIF-null, rate-limit etc)

Group: 239.255.1.60, Source count: 3, **Packets forwarded: 6860**,

**Packets received: 7087**

Source: **172.16.1.30/32**, Forwarding: 304/1/147/0, Other: 304/0/0

Source: **172.16.1.75/32**, Forwarding: 6329/8/57/3, Other: 6329/0/0

Source: **192.168.0.20/32**, Forwarding: 227/1/69/0, Other: 454/227/0

Group: 224.0.1.40, Source count: 0, Packets forwarded: 0, Packets received: 0

De estas salidas, usted puede ver que el tráfico Multicast fluye de la fuente W1 y es recibido por los miembros del grupo.

Ciertas aplicaciones requieren que los mensajes se reciban en el ordenamiento temporal en que fueron emitidos:

* Multicast totalmente ordenado
* Multicast causal

**MULTITRANSMISION A NIVEL TEMPORAL**

**METODO II: DISEMINACION DE DATOS BASADA EN EL GOSSIP:**

Se basa en el comportamiento epidémico, el objetivo principal es propagar rápidamente la información entre un gran grupo de nodos, utilizando solo información local.

En otras palabras, no hay un componente central mediante el cual se coordine la diseminación de información.

**MÉTODO I:** **ÁRBOL,** donde los nodos pueden organizarse por sí mismos de manera directa en un árbol, por tanto existe una ruta única entre cada para los nodos

**Existen 2 métodos**

Cuando los nodos se organizan en una **red sobrepuesta**, la cual después se utiliza para diseminar la información a sus miembros.

Comunicación por multitransmisión de Sistemas Distribuidos es el soporte para enviar datos de un emisor o varios destinatarios, este aspecto ha pertenecido al dominio de los protocolos de red. Con la llegada de la tecnología punto a punto se volvió más sencillo configurar rutas de comunicación, debido a que se utilizan típicamente en la capa de aplicación.

**MULTITRANSMISION A NIVEL DE APLICACIÓN**

**TIPOS**

* **Multicast no flable:** se realiza un intento de transmitir el mensaje a todos los miembros del grupo. Sin embargo, no hay garantía de que el mensaje se entregue a todos los nodos.
* **Multicast flable:** el mensaje es recibido por todos los nodos en funcionamiento, al menos una vez.
* **Multicast atomico:** el mensaje transmitido a todos los miembros del grupo es recibido por todos o ninguno. Ej. Servidores relicados que deben mantener un mismo estado.
* **Multicast atómico serializado:** es la entrega de mensajes atómicos con la secuencia de la entrega constante.

**DEFINICIÓN**

**COMUNICACIÓN POR MULTITRANSMISION**

**Algoritmos de elección**

Muchas aplicaciones y servicios distribuidos se basan en la existencia de un proceso diferenciado que coordina el trabajo de un conjunto de procesos.

En todas estas situaciones se requiere detectar que el proceso coordinador falla y elegir un nuevo proceso que asuma el papel de coordinador. La elección requiere el acuerdo sobre quién va a ser el nuevo y único coordinador. De nuevo, las decisiones se basan en la existencia de plazos para la recepción de los mensajes de respuesta.

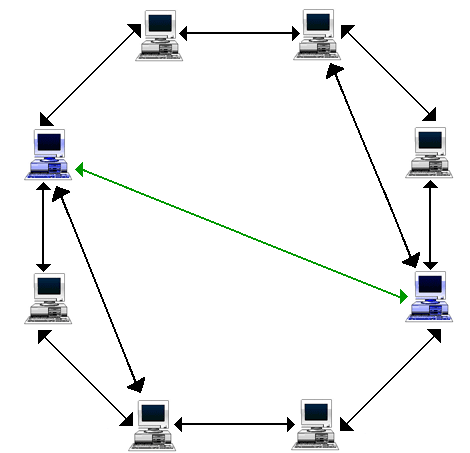
Por otra parte, para determinar el criterio de elección de un proceso como coordinador se requiere definir un orden total entre el conjunto de procesos.

Supondremos que los procesos tienen asociados identificadores únicos según los cuales pueden ordenarse

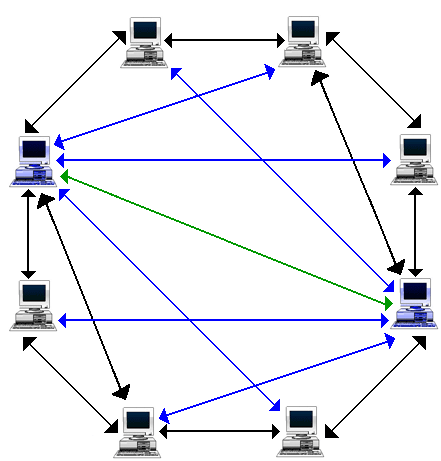
**Elecciones en sistemas de gran escala**

**Trackerless y DHT (Tabla de Hash Distribuido)**

El trackerless es un sistema que permite intercambiar entre clientes de BitTorrent los seeds y peers que se está conectado sin necesidad de hacer uso de ningún tracker. Los clientes que actúan de esta manera se les conocen como nodos DHT.

  
URL: <http://www.ayudabittorrent.com/introduccion-trackerless>

En esta imagen se puede ver como los dos ordenadores de color azul intercambian los datos a través de la flecha de color verde de los clientes a los cuales están conectados.

  
URL: <http://www.ayudabittorrent.com/introduccion-trackerless>

Ahora ya podemos ver las nuevas conexiones en las flechas de color azul. Estas nuevas conexiones se han conseguido sin necesidad de tracker y corresponden a los seeds y peers que tenía el otro nodo (ordenador azul) y que nos faltaban.

* **Seeds:** Son los ordenadores de los usuarios que están compartiendo el 100% del archivo que queremos descargar. Estos usuarios envían el archivo al resto de usuarios. Lo ideal sería que los seeds tuvieran buenas líneas. Para ver su funcionamiento de forma gráfica mira el funcionamiento del protocolo BitTorrent. Cuando no hay ningún seed o hay muy pocos es posible que la descarga se quede a medias y nunca termine de descargar sobre todo si no hay ninguna copia distribuida, ya que pueden faltar trozos que nadie tiene.
* **Tracker:** El tracker es el único ordenador que conoce dónde se encuentran todos los usuarios y dónde todos los usuarios se conectaran para poder encontrar al resto de usuarios y conocer cuando se conectan nuevos usuarios. Dicho de otro modo es al ordenador que gestiona a los usuarios.
* **Peer o cliente:** Son los usuarios conectados al tracker para la descarga de un archivo que no tienen entero, por lo tanto no se cuentan a los seeds.
* **Torrent:** Un torrent es un archivo pequeñito de aproximadamente 50 Kb que contiene toda la información sobre el archivo que queremos descargar: la ubicación del tracker, el nombre del archivo, la información del hash que evitará poder bajar un archivo corrupto, etc…

**Nodos DHT**

A diferencia de un tracker, la única función de un nodo DHT es devolver un listado de IPs de los usuarios y nodos que contienen ese torrent.

Un nodo DHT consume muy pocos recursos, si a eso añadimos que no hay límite en el número de nodos no existen problemas de sobrecarga. Es más el servidor de nodos no es indispensable ya que los propios clientes de BitTorrent que soportan trackerless también hacen de nodos.

**Problemas al conectar con otro nodo**

No se podrá conectar con otro nodo en los siguientes casos:

* Si no se soporta trackerless.
* Si el torrent está marcado como privado.
* Entre dos nodos que no usan el mismo sistema trackerless. (Azureus implementa su propio sistema de trackerless y es incompatible con el oficial)
* Entre dos nodos que tengan los puertos UDP (User Datagram Protocol) cerrados.

**Puerto del BitTorrent trackerless:** Generalmente para usar el trackerless hay que abrir el mismo puerto que necesita el BitTorrent para conectar con otros usuarios pero en vez de ser TCP (Protocolo de control de transmisión) es UDP.

**BitTorrent sin trackers:** El BitTorrent puede funcionar sin trackers siempre que el torrent no este marcado como privado. En cuanto se contacta con otro nodo que dispone de trackerless se comienza a intercambiar los datos de los nodos y clientes a los que están conectados.

**Contactar con el primer cliente**

Para contactar con el primer cliente hay 4 formas posibles:

* Lo más habitual es a través de un tracker de BitTorrent. Por norma general viene indicado en el propio torrent.
* A través de un nodo indicado en el propio torrent. Puede usarse si en vez de querer instalar un tracker, se instala un servidor de nodos que consume muchos menos recursos. El nodo puede ser el seed original y puede ser usado para intercambiar archivos entre unos pocos amigos. Si el nodo indicado tiene ip dinámica es muy recomendable usar DNS para evitar no poder contactar con él.
* A través del nodo por defecto que incorporan algunos clientes de BitTorrent y a través del cual conecta a todos los usuarios de ese cliente:
  + El BitComet en router.bitcomet.com:554 (UDP)
  + El BitTorrent Oficial en router.bittorrent.com:6881 (UDP)
  + El uTorrent en router.utorrent.com:6881 (UDP)
* Algunos clientes de BitTorrent permiten introducir a mano la IP de otro cliente.

**Diferencia entre trackerless y DHT**

Aunque es común usar trackerless o DHT (Tabla de Hash Distribuido) para referirse a lo mismo hay una pequeña diferencia.

El DHT es un sistema para compartir datos mediante nodos en una red descentralizada y hay muchos programas que lo usan. El trackerless en cambio es como han adaptado el DHT para funcionar en una red BitTorrent y así conseguir reducir la carga de procesos del tracker. El trackerless se usa exclusivamente en el BitTorrent.

**Ventajas y desventajas del trackerless**

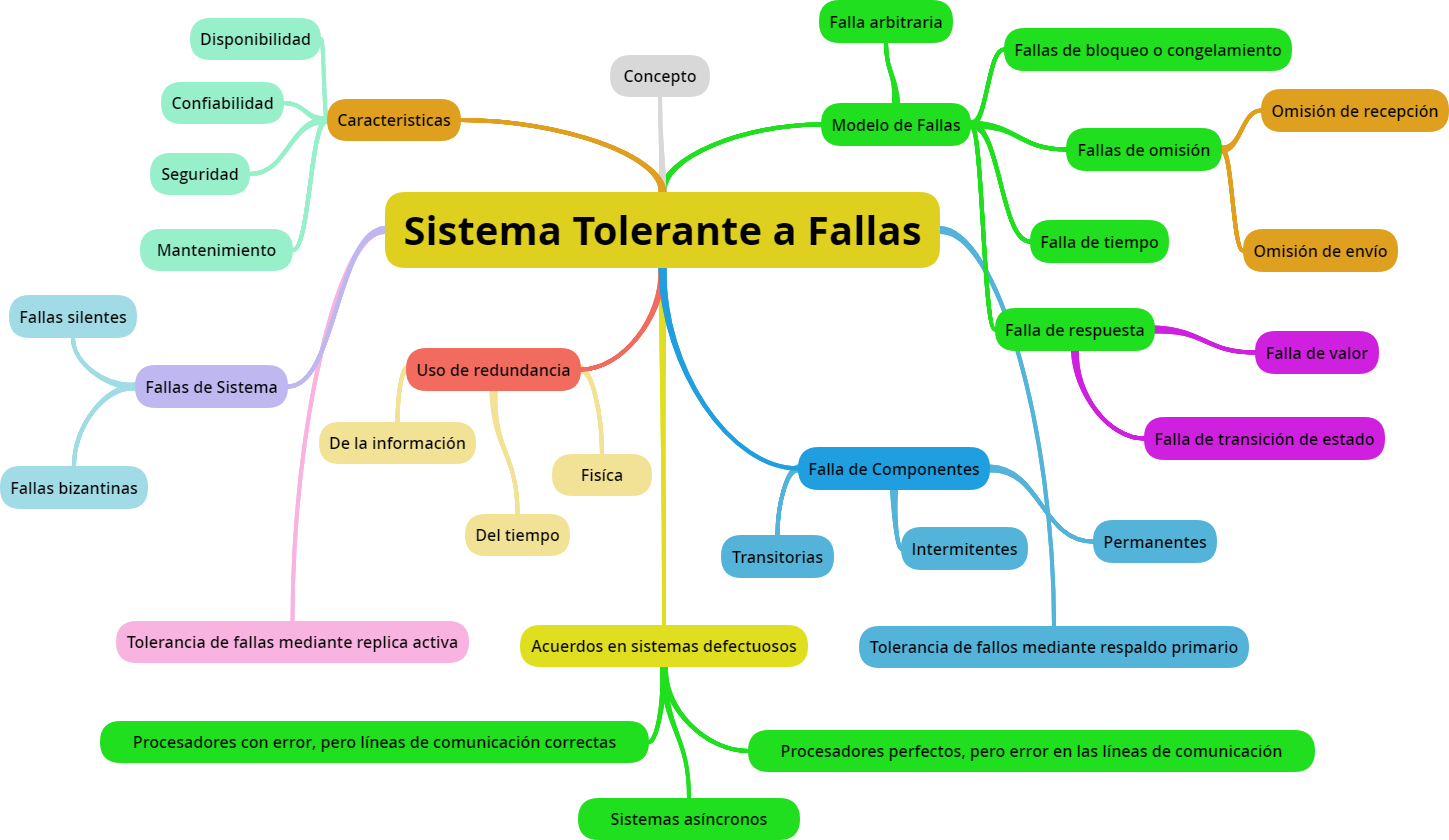
Ventajas:

* Reducen la carga del tracker.
* Si el tracker se cae se puede conectar con nuevos peers y seeds.
* No es necesario el uso de un tracker.

Desventajas:

* El tracker no puede controlar lo que suben y bajan los usuarios y por lo tanto no puede tomar medidas con los usuarios que no son solidarios.
* El tracker no puede controlar que solo se conecten los usuarios registrados. Las redes privadas dejan de serlo.

Modelo de fallas



1. Tolerancia de Fallas

El fallo tanto del hardware como el software es algo prácticamente inevitable, y por más confiable que pueda parecer algún componente, siempre es importante estar preparado para cuando este falle. En un sistema centralizado por lo general el fallo de cualquier componente del sistema provoca que todos los servicios que este ofrece dejen de funcionar, en cambio, en un sistema distribuido, los fallos son parciales, puesto que solo afectan a los servicios que el componente que fallo este prestando, mientras que otros servicios que prestan otros componentes siguen funcionando.

El tratamiento de fallos en un sistema distribuido es una tarea difícil, pero que se puede lograr si se utilizan las técnicas adecuadas, según el sistema que se desee proteger.

(<http://www.unap.edu.pe/cidiomas/licing/pdf/sd.pdf>)

* 1. Características

Para que un sistema distribuido pueda ser tolerante a fallos, se ocupan las siguientes características:

* + 1. Disponibilidad: Es la capacidad de un activo o componente para estar en un estado (arriba) para realizar una función requerida bajo condiciones dadas en un instante dado de tiempo o durante un determinado intervalo de tiempo, asumiendo que los recursos externos necesarios se han proporcionado. Un sistema con alta disponibilidad es aquel que puede trabajar en cualquier tiempo.
    2. Confiabilidad: Es la capacidad de un activo o componente para realizar una función requerida bajo condiciones dadas para un intervalo de tiempo dado. cuando hablamos de confiabilidad el componente trabaja continuamente durante un periodo de tiempo dado, en otras palabras, la función del componente no se interrumpe, el componente se pone en operación (arriba) y se mantiene arriba. Un sistema con alta confiabilidad es aquel que funciona por largos periodos de tiempo sin fallo alguno.
    3. Seguridad: Si el sistema falla, no ocurre nada grave. se refiere a la situación en la que no acontece nada catastrófico cuando un sistema deja de funcionar correctamente durante un tiempo. Esta característica demuestra lo difícil que es construir sistemas seguros. Ejemplo: algunos sistemas que controlan plantas nucleares, si algunos de esos sistemas fallan, pueden traer consecuencias catastróficas.
    4. Mantenimiento: Cuando u sistema falla, se pude reparar de manera fácil y rápida. Es la capacidad (o probabilidad si hablamos en términos estadísticos), bajo condiciones dadas, que tiene un activo o componente de ser mantenido o restaurado en un periodo de tiempo dado a un estado donde sea capaz de realizar su función original nuevamente.

(<https://so2-caece.wikispaces.com/M%C3%B3dulo%208.%20Tolerancia%20a%20Fallos%20y%20Seguridad.#6>)

(<http://www.unap.edu.pe/cidiomas/licing/pdf/sd.pdf>)

(<https://sites.google.com/site/mrtripus/home/sistemas-operativos-2/3-6-tolerancia-a-fallos-nivel-proceso-nivel-almacenamiento>)

(<https://es.slideshare.net/dabiddo/tolerancia-a-fallos>)

(<https://maintenancela.blogspot.com/2011/10/confiabilidad-disponibilidad-y.html>)

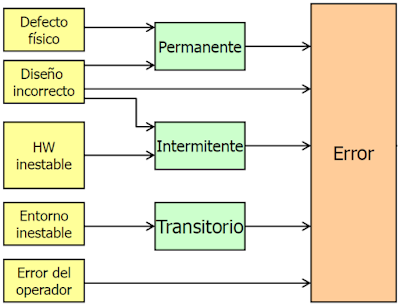
* 1. Modelo de fallas
     1. Fallas de bloqueo o congelamiento: Un servidor se detiene, pero estaba trabajando correctamente hasta que se detuvo.
     2. Falla de omisión: un servidor falla en responder a las peticiones.
        1. Omisión de recepción: un servidor falla en recibir mensaje.
        2. Omisión de envío: un servidor falla en enviar mensajes.
     3. Falla de tiempo: Aplicable en sistemas distribuidos síncronos
* Respuestas que no están disponibles para los clientes en un tiempo especificado intervalo.
* Las garantías de tiempo requieren acceso garantizado a los recursos cuando ellos son necesarios.
* Ejemplo: control y sistemas de monitoreo, sistemas multimedia, time out en la conexión con una base de datos.
  + 1. Falla de respuesta: El servidor responde de manera incorrecta.
       1. Falla de valor: Un servidor responde de forma equivocada a una petición.
       2. Falla de transición de estado: Este tipo de falla ocurre cuando el servidor responde de forma inesperada ante una solicitud.
    2. Falla arbitraria: Ocurren cuando el servidor produce una salida que nunca debió producir, la cual no es detectada como incorrecta. (<http://programacion-js.blogspot.com/2018/01/sistemas-distribuidos-tolerancia-fallos.html>)

(<https://es.slideshare.net/dabiddo/tolerancia-a-fallos>)

* 1. Fallas de componentes

Los sistemas de cómputo pueden fallar debido a una falla en algún componente, como procesador, la memoria, un dispositivo de E/S, un cable o el software. Una falla es un desperfecto, causado tal vez por un error de diseño, un error de fabricación, un error de programación, un daño físico, el deterioro con el curso del tiempo, condiciones ambientales adversas (pudo nevar sobre la computadora), entradas inesperadas, un error del operador, roedores comiendo parte del sistema y muchas otras causas. No todo esto conduce (de inmediato) a fallas del sistema, pero algunas de estas cosas sí.

Las fallas se clasifican por lo general como transitorias, intermitentes o permanentes.

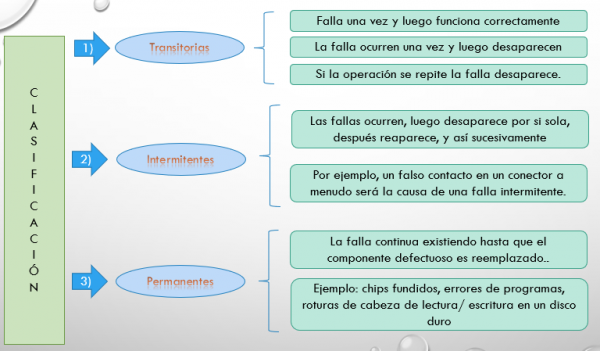


<http://programacion-js.blogspot.com/2018/01/sistemas-distribuidos-tolerancia-fallos.html>

* + 1. Transitorias: Son aquellos fallos que aparecen una vez y después desaparecen aun cuando la misma operación se repite. Ejemplo: un pájaro que vuela a través del rayo de un transmisor de microondas provoca la pérdida de bits en una red (por no mencionar un pájaro frito). Si la transmisión expira y se repite, es probable que funcione la segunda vez.
    2. Intermitentes: Ocurre, desaparece, pero vuelve a aparecer más tarde. Ejemplo: un mal contacto de un conector causa con frecuencia una falla intermitente, las cuales son graves por su difícil diagnóstico.
    3. Permanentes: Son aquellos fallos que aparecen y no desaparecen hasta que el componente erróneo es reemplazado o es arreglado el problema. Ejemplo: Los circuitos quemados, los errores del software y el rompimiento de la cabeza del disco provocan con frecuencia fallas permanentes.

(<https://es.slideshare.net/dabiddo/tolerancia-a-fallos>)

(<http://www.academia.edu/34833896/Tanenbaum_Andrew_-_Sistemas_Operativos_Distribuidos.PDF>) - página 212



<http://tolerancia-a-fallas.blogspot.es/>

* 1. Fallas de sistema

En un sistema distribuido crítico, con frecuencia nos interesa que el sistema pueda sobrevivir a las fallas de los componentes (en particular, del procesador), en vez de hacer que las fallas sean poco probables. La confiabilidad de un sistema es en particular importante en un sistema distribuido, debido a la gran cantidad de componentes presentes; de ahí la mayor posibilidad de que falle uno de ellos.

Se distinguen dos tipos de fallas:

* + 1. Fallas silentes: un procesador que falla solo se detiene y no responde a las entradas subsecuentes ni produce más entradas, excepto que puede anunciar que ya no está funcionando. También se llaman fallas de detención.
    2. Fallas bizantinas: un procesador que falla continua su ejecución, proporcionando respuestas incorrectas a las preguntas y posiblemente trabajando de manera maliciosa junto con otros procesadores que han fallado, para dar la impresión de que todos funcionan de manera correcta, aunque no sea así. Los errores no detectados en el software exhiben con frecuencia fallas bizantinas. Es claro que enfrentarse a las fallas bizantinas será más difícil que enfrentarse a las fallas silentes.

(<http://www.academia.edu/34833896/Tanenbaum_Andrew_-_Sistemas_Operativos_Distribuidos.PDF>) - página 213

* 1. Uso de redundancia

El método general para la tolerancia de fallas consiste en el uso de redundancia. Existen tres tipos posibles: redundancia de la información, redundancia del tiempo y la redundancia física.

* + 1. Con la redundancia de la información, se agregan algunos bits para poder recuperar los bits revueltos. Por ejemplo, se puede agregar un código Hamming para transmitir los datos y recuperarse del ruido en la línea de transmisión.

Código de Hamming: Este algoritmo es, por así decirlo, un juego de detectives. Donde tenemos varias personas (los bits), y hay uno, varios o ningún malo (bits erróneos) entre ellas. A su vez tenemos varias pistas que van a hacer que descartemos a los buenos, hasta quedarnos con los malos; aunque ojo, también puede haber pistas que mientan. Realmente es bastante sencillo.

Es decir, hay unos bits que corrigen a otros. Estos bits que corrigen a otros se llaman “bits de redundancia”, pues son bits extra a la información para intentar corregir los errores o al menos comprobar que todo esté correcto. Un ejemplo en el mundo físico, sería como tener en un papel un texto cualquiera como “Mi mamá me mima”, y más abajo en el mismo papel la suma de todas las letras del texto “M=6, I=2, A=3, E=1” (es información repetida/redundante, pues ya está contenido en el dato). De este modo si se nos cae líquido encima de la tinta del texto haciendo ilegible algunas palabras del texto “Xi XaXá me Xima” (las “X” representan la ilegibilidad del texto); todavía podemos recuperar el texto original con la suma de letras pues sabemos que hay 6 “Ms” y solo vemos legibles 2, por lo que se deduce que las letras ilegibles tienen que ser “Ms” también (lo mismo ocurriría si las equis “X” anteriores fueran letras y no borrones, la letra equis “X” no existe en las letras sumadas por lo que es una letra errónea).

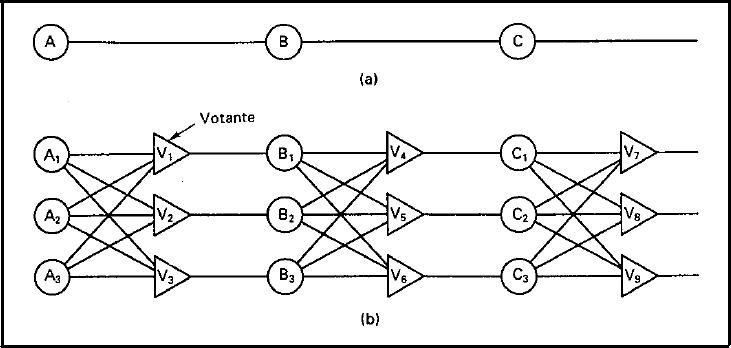
* + 1. Con la redundancia del tiempo, se realiza una acción, y entonces, en caso necesario, se vuelve a realizar. Si una transacción aborta, puede volverse a realizar sin daño alguno. La redundancia de tiempo es de particular utilidad cuando las fallas son transitorias o intermitentes.
    2. Con la redundancia física, se agrega un equipo adicional para permitir que el sistema como un todo tolere la pérdida o el mal funcionamiento de algunos componentes. Por ejemplo, se pueden agregar más procesadores, de modo que, si unos pocos de ellos fallan, el sistema pueda seguir funcionando de manera correcta.

(<http://www.academia.edu/34833896/Tanenbaum_Andrew_-_Sistemas_Operativos_Distribuidos.PDF>) - página 215

* 1. Tolerancia de fallas mediante replica activa

Es una técnica muy conocida para proporcionar la tolerancia de fallas mediante la redundancia física. Se utiliza en la biología (los mamíferos tienen dos ojos, dos oídos, dos pulmones, etc.), aviación (los modelo 747 tienen cuatro motores, pero pueden volar con tres) y los deportes (varios árbitros, en caso de que alguno omita un evento).

([http://www.academia.edu/34833896/Tanenbaum\_Andrew\_-\_Sistemas\_Operativos\_Distribuidos.PDF](http://www.academia.edu/34833896/Tanenbaum_Andrew_-_Sistemas_Operativos_Distribuidos.PDF%20página%20215)) - página 215

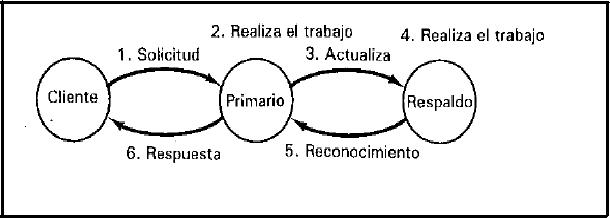


(<https://so-ii-itst.wikispaces.com/so-ii-u3-itst>)

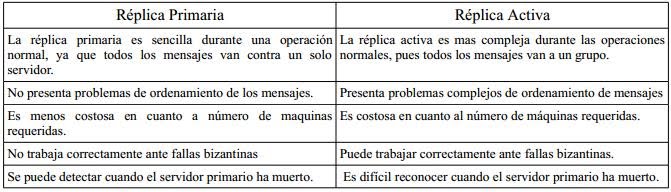
* 1. Tolerancia de fallas mediante respaldo primario

La idea esencial del método de respaldo primario es que, en cualquier instante, un servidor es el primario y realiza todo el trabajo. Si el primario falla, el respaldo ocupa su lugar. En forma ideal, el remplazo debe ocurrir de manera limpia, y ser notado únicamente por el sistema operativo cliente, no por los programas de aplicación. Algunos ejemplos son el gobierno (el vicepresidente), la aviación (los copilotos), los automóviles (llantas de refacción), etc.

([http://www.academia.edu/34833896/Tanenbaum\_Andrew\_-\_Sistemas\_Operativos\_Distribuidos.PDF](http://www.academia.edu/34833896/Tanenbaum_Andrew_-_Sistemas_Operativos_Distribuidos.PDF%20página%20218)) - página 218



<https://so-ii-itst.wikispaces.com/so-ii-u3-itst>



<https://sites.google.com/site/mrtripus/home/sistemas-operativos-2/3-6-tolerancia-a-fallos-nivel-proceso-nivel-almacenamiento>

* 1. Acuerdos en sistemas defectuosos

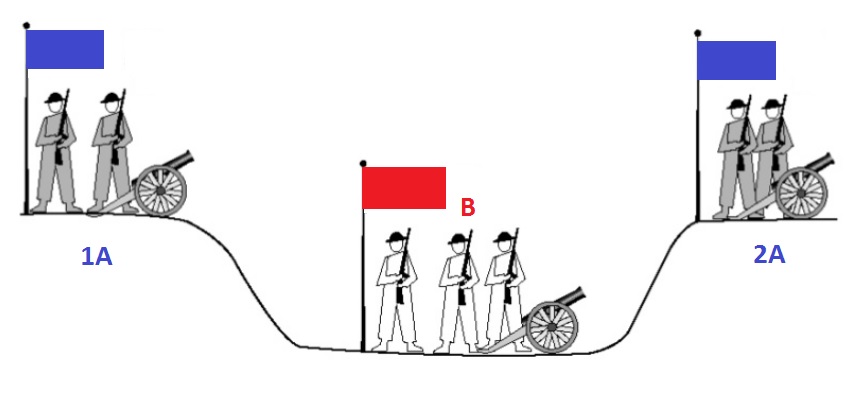
En muchos sistemas distribuidos existe la necesidad de que los procesos coincidan en algo. Algunos ejemplos son la elección de un coordinador, decidir cuándo comprometer una transacción o no, dividir las tareas entre trabajadores, la sincronización, etc.

Consideramos los siguientes casos como temas centrales en los acuerdos:

* + 1. Procesadores perfectos, pero error en las líneas de comunicación (problemas de los dos ejércitos).

Imagínese dos ejércitos que se encuentran en un valle de tal forma: el ejército A se divide en dos, es decir, uno está en una montaña(1A) y el otro en otra montaña (2A), quedando entre ambos un ejército B. Existe el siguiente problema: 1A y 2A solamente pueden comunicarse entre sí mediante un mensajero, del que no se tiene la certeza de que pudiese comunicarse con la otra parte, es decir, si 1A desease comunicarse con 2A, no podría tener la certeza de que el mensaje les llegase ya que el ejército B podría matarlo. Lo que ocurre es que el ejército 1A desearía atacar al amanecer y se precisa de un protocolo que permita que esta información le llegue a 2A. ¿Existe dicho protocolo?

(<http://secomparteosepierde.blogspot.com/2010/07/problema-de-los-dos-ejecitos.html>)



<https://es.slideshare.net/jcbp_peru/ul-rccap7el-nivel-de-transporte-en-internet>

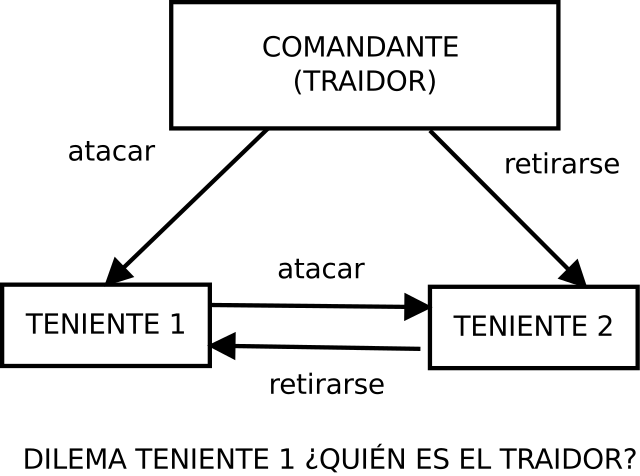
* + 1. Procesadores con error, pero líneas de comunicación correctas (problema de los generales bizantinos).

Supongamos un escenario de guerra en el que tenemos un grupo de m generales bizantinos que están asediando una ciudad desde distintos lugares y tienen que ponerse de acuerdo para atacar o retirarse de forma coordinada. Entre los generales hay solo uno que puede cursar la orden por ser el comandante. El resto se dice que son tenientes.

Los generales se comunican a través de mensajeros y las dos posibles órdenes del comandante son "atacar" y "retirarse".

Uno o más de los generales puede ser un traidor (al resto se les llama leales), por lo que su objetivo es conseguir que todos los generales leales no se pongan de acuerdo. Para ello pueden ofrecer información errónea. Por ejemplo, si el comandante es el traidor, podría mandar órdenes contradictorias a los distintos tenientes. Si el teniente es un traidor podría indicarles a otros tenientes, con el fin de confundirlos y que creyeran que el traidor es el comandante, que el comandante les envió la orden contraria a la que realmente les envió.

(<https://wiki2.org/es/Problema_de_los_generales_bizantinos>)



(<http://www.wikiwand.com/es/Problema_de_los_generales_bizantinos>)

* + 1. Sistemas asíncronos: no se pueden distinguir los procesadores lentos de los muertos.

(<http://studylib.es/doc/7376235/tolerancia-a-fallas>)

Bibliografía

<http://www.unap.edu.pe/cidiomas/licing/pdf/sd.pdf>

<https://docplayer.es/61744779-Francisco-de-asis-lopez-fuentes-sistemas-distribuidos.html>

<https://so2-caece.wikispaces.com/M%C3%B3dulo%208.%20Tolerancia%20a%20Fallos%20y%20Seguridad.#6>

<http://www.unap.edu.pe/cidiomas/licing/pdf/sd.pdf>

<https://sites.google.com/site/mrtripus/home/sistemas-operativos-2/3-6-tolerancia-a-fallos-nivel-proceso-nivel-almacenamiento>

<https://es.slideshare.net/dabiddo/tolerancia-a-fallos>

<http://programacion-js.blogspot.com/2018/01/sistemas-distribuidos-tolerancia-fallos.html>

<https://es.slideshare.net/dabiddo/tolerancia-a-fallos>

<http://programacion-js.blogspot.com/2018/01/sistemas-distribuidos-tolerancia-fallos.html>

<https://es.slideshare.net/dabiddo/tolerancia-a-fallos>)

<http://www.academia.edu/34833896/Tanenbaum_Andrew_-_Sistemas_Operativos_Distribuidos.PDF>

<http://tolerancia-a-fallas.blogspot.es/>

<https://so-ii-itst.wikispaces.com/so-ii-u3-itst>

<https://sites.google.com/site/mrtripus/home/sistemas-operativos-2/3-6-tolerancia-a-fallos-nivel-proceso-nivel-almacenamiento>

<http://secomparteosepierde.blogspot.com/2010/07/problema-de-los-dos-ejecitos.html>

<https://es.slideshare.net/jcbp_peru/ul-rccap7el-nivel-de-transporte-en-internet>

<https://wiki2.org/es/Problema_de_los_generales_bizantinos>

<http://www.wikiwand.com/es/Problema_de_los_generales_bizantinos>

<http://studylib.es/doc/7376235/tolerancia-a-fallas>