Universidad Nacional de La Plata
Facultad de Informática

Maestría en Redes de Datos

Transporte de datos multicast confiable con control de congestión dinámico

Tesis presentada para obtener el grado de Magister en Redes de Datos

Director de Tesis: Ing. Luis Marrone

Alumno: Lic. José Gabriel Gomiz
Cooperativa Obrera Ltda.

Mayo de 2015

Índice de contenido

Motivación del trabajo realizado....5
Capítulo 1: Introducción....6
Transmisión de datos “multicast”....6
Esquema de direccionamiento en TCP/IP....6
Direccionamiento en IP versión 6 (IPv6)....7
Tipos de direcciones en IPv6....8
Direcciones IPv6 unicast....8
Multicast en redes Ethernet....9
Multicast en redes IP....10
Direcciones IP Multicast....11
Mapeo de multicast IP a multicast Ethernet....11
Multicast en IPv6....12
Equivalencia de ámbitos multicast en IPv4-IPv6....13
Capítulo 2: Multicast confiable....14
Transmisión confiable....14
Multicast Confiable....15
Mecanismos basados en ACKs....16
Mecanismos basados en NACKs....16
Grupo de trabajo RMT....17
Alternativas de transporte multicast....18
Capítulo 3: Arquitectura PGM....23
Descripción general del protocolo....23
Jerarquía....24
Supresión de NAKs....26
Forward error correction....27
Ventana de transmisión....27
Reparaciones locales (DLRs)....28
Polling....28
Control de congestión....29
Capítulo 4: Escenario de Aplicación....30
Estructura del sistema....30
Equipamiento....31
Comunicaciones....31
Transmisión Multicast en POSEIDON / ZEUS....33
Preparación de la transmisión....34
Transmisión....35
Función Throttle....36
Reparaciones....36
Consistencia....37

Parametrización....37
Inconvenientes y limitaciones....37
Capítulo 5: OpenPGM....39
Introducción....39
Plataformas Soportadas....39
Conceptos....39
Transporte....40
Parámetros de transporte de red....41
Puerto de datos destino....41
Puerto de datos fuente....41
Parámetros de red....41
Sesiones PGM....42
Identificadores globales de envío (GSI)....42
Identificadores de sesión de transporte (TSI)....43
Sesión....43
Encapsulado sobre UDP....44
Interfaz de programación....45
Formato de direcciones....45
Unidades de datos....46
Envío y recepción....46
Capítulo 6: Diseño de la solución....50
Estrategia de envío con control de congestión....50
Fórmula matemática AIMD....51
Diseño de la función de envío....51
Parámetros configurables....52
Algoritmo de envío....52
Diseño de la librería compartida....53
Modificaciones necesarias a la librería OpenPGM....54
Capítulo 7: Implementación....58
Estructura de la librería....58
Descripción del código fuente....59
Mecanismos de recepción no bloqueante....61
Implementación del control de congestión....62
Implementación de utilidades para envío y recepción....62
Utilidad filesend....63
Opciones filesend:....63
Utilidad filerecv....63
Opciones filerecv:....64
Utilidades para pruebas de rendimiento....64
Capítulo 8: Mediciones y performance....66
Escenarios de testing....66
Pruebas básicas de latencia....69

Pruebas de ancho de banda multicast disponible a nivel red (IP).....69
Pruebas con buffer de 100 bytes.....70
Pruebas con buffer de 1500 bytes.....71
Pruebas de ancho de banda multicast a nivel aplicación (PGM).....71
Performance MDP, PGM y PGMCOOP.....73
Esquema genérico de las pruebas.....73
Escenarios de prueba.....75
Escenario A: Sucursal 08 – 10 Mb/s – 3 receptores.....76
Escenario B: Sucursal 21 – 10 Mb/s – 5 receptores.....78
Escenario C: Sucursal 02 – 10 Mb/s – 8 receptores.....79
Escenario D: Sucursal 96 – 100 Mb/s – 4 receptores.....80
Escenario E: Sucursal 36 – 100 Mb/s – 9 receptores.....81
Escenario F: Sucursal 42 – 100 Mb/s – 20 receptores.....82
Capítulo 9: Conclusiones.....84
Tabla de resultados.....84
Conclusiones generales.....85
Conclusiones acerca de PGMCOOP.....86
Inconvenientes del protocolo PGMCOOP.....88
Análisis del protocolo en función de los objetivos planteados.....89
Posibilidades de investigación y/o mejora.....89
Bibliografía.....91

Índice de ilustraciones

Ilustración 1: Esquema de direccionamiento classful.....8
Ilustración 2: Formato de direcciones multicast en IPv6.....13
Ilustración 3: http://www.cisco.com/web/about/ac123/ac147/images/ipj/ipj_1-2/figure_rmpa5.gif.....21
Ilustración 4: Diagrama lógico sucursal.....32
Ilustración 5: Multicast vs Unicast.....35
Ilustración 6: Esquema original de transmisión multicast.....36
Ilustración 7: http://miru.hk/wiki/OpenPGM_operating_environment.png.....42
Ilustración 8: http://miru.hk/wiki/Udp_encapsulated_ports.png.....46
Ilustración 9: Curva de utilización de capacidad de red (curva roja).....53
Ilustración 10: Diseño de librería pgmcoop.....57
Ilustración 11: Medición de latencia con PING.....72
Ilustración 12: Pruebas iperf con buffers de 100 bytes.....73
Ilustración 13: Pruebas iperf con buffers de 1500 bytes.....74
Ilustración 14: Gráfico comparativo de rendimiento con pgmping.....75
Ilustración 15: Formato planillas de resultados.....79
Ilustración 16: Tabla general de resultados.....87
Ilustración 17: Resultados de pruebas en Sucursal 08.....89
Ilustración 18: Resultados de pruebas en Sucursal 21.....89
Ilustración 19: Resultados de pruebas en Sucursal 02.....90

Ilustración 20: Resultados de pruebas en Sucursal 96.....90
Ilustración 21: Resultados de pruebas en Sucursal 36.....91
Ilustración 22: Resultados de pruebas en Sucursal 42.....91

Indice de Tablas

Tabla 1: Valores del campo scope (IPv6).....14
Tabla 2: Detalle del encabezado PGM.....26
Tabla 3: Posibles valores para la variable PGM_TIMER.....52
Tabla 4: Opciones posibles para utilidad filesend.....66
Tabla 5: Opciones posibles para utilidad filerecv.....67
Tabla 6: Resultados de pruebas de latencia.....72
Tabla 7: Resultados de pruebas iperf con buffers de 100 bytes.....73
Tabla 8: Resultados de pruebas iperf con buffers de 1500 bytes.....74
Tabla 9: Resultados de pruebas con pgmping.....75

Motivación del trabajo realizado

En escenarios donde se tienen múltiples estaciones equivalentes para realizar una función determinada, normalmente es necesario transmitir a las mismas datos “bulk” de considerable tamaño y es necesario enviarlos a todas las estaciones. En ambientes 1 a N de este tipo, la transmisión “unicast” resulta inadecuada e ineficiente; la transmisión “multicast” surge como la mas adecuada para dicha función ya que está diseñada para ser escalable en cantidad de estaciones destino ya que la transmisión se realiza una vez independientemente del número de destinatarios.

La naturaleza de la transmisión multicast hace que sea un mecanismo no orientado a conexión, por lo que la mayoría de los protocolos multicast se desarrollan sobre la capa de transporte UDP y sobre este es necesario construir un esquema de confiabilidad. Para aplicaciones de “streaming” de audio o video, la pérdida ocasional de un segmento es aceptable, pero al transmitir datos críticos (archivos, programas, etc.) es necesario un mecanismo que garantice confiabilidad.

Actualmente se están desarrollando varios protocolos multicast con esquema de confiabilidad que se encuentran en estado experimental[1].

NORM (Negative-acknowledgement Oriented Reliable Multicast Protocol). Este protocolo se define dentro del contexto de participantes comunicandose con paquetes sin orientación a conexión (IP / UDP) y el diseño se basó principalmente en un único transmisor que envía datos “bulk” a un grupo de receptores.
Otras alternativas para utilizar en los casos que se necesita confiabilidad en el transporte multicast son los protocolos SRM (Scalable Reliable Multicast) y RMTP (Reliable Multicast Transport Protocol).
FLUTE (File Delivery over Unidirectional Transport), es un protocolo experimental para el envío unidireccional de archivos a través de Internet, que es una red muy conveniente para las transmisiones multicast.

Por último, PGM: Pragmatic General Multicast (en estado experimental desde Diciembre de 2001) es un protocolo diseñado pensando en la simplicidad como objetivo principal. Este protocolo está ideado para ser extremadamente escalable, posee mecanismos que controlan la retransmisión y un esquema basado en jerarquías.

PGM es el protocolo considerado para desarrollar el presente trabajo debido a los siguientes factores:

- Mecanismo de retransmisión basado en NAKs / NCFs
- Implementación para Linux (librería OpenPGM) disponible y con desarrollo activo
- Amplia documentación, ejemplos, etc.

[1] Por ahora no definen ningún estándar en Internet

En base a lo expuesto, la motivación de la Tesis es investigar las técnicas y mecanismos utilizados en estos protocolos experimentales y utilizar alguno de éstos o diseñar una nueva variante para la aplicación en cuestión, lo más simple posible, que cumpla con los objetivos de:

- ser capa de transporte multicast para aplicaciones con requerimientos básicos de confiabilidad
- parametrización para ajustar las condiciones de transmisión
- adaptabilidad dinámica al escenario (condiciones de equipamiento) y las condiciones de la red mediante feedback de los receptores en las transmisiones y control de throughput en el emisor

Capítulo 1: Introducción

Transmisión de datos "multicast"

En transmisión de datos, la tecnología multicast representa un servicio de red en el cual un único flujo de datos, proveniente de una fuente determinada, se puede enviar simultáneamente a varios receptores interesados. Es función de la infraestructura de red transportar este flujo de datos, replicándolo cuando sea necesario, a todos los receptores que registren interés en recibir los mismos.

El multicast está orientado a aplicaciones del tipo "uno a muchos" (1 a N) o "muchos a muchos" (N a M). En estos casos, presenta claras ventajas cuando se lo compara con los mecanismos de transmisión unicast y broadcast. En unicast, es necesario que la fuente replique varios flujos de datos idénticos con el objeto de transmitirlos a cada uno de los receptores, generando desperdicio de ancho de banda de la red. Por otro lado, el sistema broadcast envía los datos a toda la red de forma indiscriminada. Esto también da como resultado el desperdicio de recursos, ya que implica el transporte de datos para todas las estaciones de la red, aunque el número de receptores interesados en los datos sea reducido. Con multicast, la fuente de la transmisión envía una única copia de los paquetes hacia una dirección de grupo multicast. La infraestructura de red replica estos paquetes de forma inteligente, encaminando los datos de acuerdo con la topología de receptores interesados en esa información. La ventaja principal de la transmisión de datos multicast es una clara reducción en la carga de la red ("network load"). [BANIK00]

Entre las diversas aplicaciones que se pueden beneficiar con el uso de multicast encontramos: videoconferencia; aprendizaje a distancia; distribución de archivos y software, conciertos en vivo, actualizacion de bases de datos, juegos distribuídos, streaming, etc.

Esquema de direccionamiento en TCP/IP

En las redes TCP/IP, a cada estación conectada se le asigna un identificador universal de 32 bits conocido como su "dirección IP". Esta dirección está compuesta por un prefijo para identificar la red y este prefijo es común a todas las estaciones de la misma red.
Conceptualmente, cada dirección es un par (idred, idhost) donde idred identifica la red y idhost identifica la estación. En la práctica, sin embargo, la forma de particionar las direcciones en prefijo y sufijo no es uniforme debido a que los diseñadores no especificaron un único separador. En el esquema de direccionamiento original (conocido como "classful" [TANEN00]), cada dirección IP tenía una de las primeras tres formas de la ilustración 1. La cuarta forma es, justamente, la que se designó para las direcciones multicast y que utilizaremos frecuentemente en este trabajo.

En este esquema de direccionamiento, cada dirección era auto-identificable, porque el límite entre prefijo y sufijo se podía calcular a partir de la dirección y no se necesitaba ninguna información adicional. En particular, la clase de una dirección se podía determinar de los 3 bits mas significativos.

Las direcciones de clase A, que se utilizaban para las pocas redes que tenían mas de 2^{16} estaciones, usaban 7 bits para el idred y 24 bits para el idhost. Las direcciones de clase B, utilizadas para redes de tamaño intermedio que tenían entre 2^8 y 2^{16} estaciones, usaban 14 bits para el idred y 16 bits para el idhost. Finalmente, las clases C, utilizadas para redes que tenían menos de 2^8 estaciones, usaban 21 bits para el idred y solo 8 bits para el idhost. Notemos que las direcciones IP se definieron originalmente de manera que se pudieran dividir el idred y el idhost rápidamente. La eficiencia era especialmente importante en los "routers" que usan el idred de la dirección para determinar hacia donde encaminar el paquete.

Ilustración 1: Esquema de direccionamiento classful

Debido a que el esquema classful requería un único prefijo de red para cada red física y que las tecnologías de red LAN se volvieron muy populares, rápidamente se agotaría el espacio de direccionamiento. Consecuentemente, se desarrolló una extensión de direccionamiento para conservar los prefijos de red. Esta extensión fue conocida como "direccionamiento por subredes" y permite a múltiples redes físicas compartir un prefijo.
Luego, en 1990, se desarrolló una segunda extensión que ignora la jerarquía classful y permite realizar la división entre prefijo de red y sufijo en cualquier punto arbitrario. Esta extensión conocida como "classless" permite una utilización mas completa del espacio de direccionamiento.

A pesar de estas reformas al esquema de direccionamiento, el protocolo IP mantuvo la característica adicional de su esquema de direccionamiento que permitía una entrega eficiente de datagramas en un esquema multipunto.

Direccionamiento en IP versión 6 (IPv6)

La característica distintiva mas obvia de IPv6 es, por supuesto, que utiliza 128 bits para cada dirección, lo que significa que tenemos disponibles 2^{128} direcciones. Esta decisión se tomó para tener un espacio de direcciones lo suficientemente grande como para poder dividirlo en dominios de ruteo jerárquicos que reflejen la topología de la Internet moderna. El uso de 128 bits permite utilizar jerarquías de múltiples niveles y flexibilidades al diseñar direccionamientos y ruteos jerárquicos que hoy no es posible con las redes basadas en IPv4. [DAVIE00]

Tipos de direcciones en IPv6

Existen tres tipos de direcciones en IPv6:

1. *Unicast:* una dirección unicast identifica una única interfaz dentro del ambiente del tipo de dirección. El ambiente de una dirección es la región de la red IPv6 sobre la cual la direción es única. Con la topología adecuada de ruteo unicast, los datagramas dirigidos a una dirección unicast son entregados solo a una interfaz.
2. *Multicast:* una dirección multicast identifica 0 o mas interfaces. Con la topología adecuada de ruteo multicast, un datagrama dirigido a una dirección multicast es entregado a todas las interfaces identificadas con esa dirección.
3. *Anycast:* una dirección anycast identifica a múltiples interfaces. Con la topología adecuada de ruteo anycast, un datagrama dirigido a una dirección anycast es entregado solo a una interfaz – la interfaz mas cercana que sea identificada con dicha dirección. La interfaz mas cercana se define como la mas cercana en términos de distancia de ruteo.

En todos los casos, las direcciones IPv6 no identifican nodos, sino que identifican interfaces. Un nodo es identificado por cualquier dirección unicast asignada a alguna de sus interfaces.

IPv6 soporta direcciones para diferentes ambientes. Existen ambientes globales y no globales (link-local). Operativamente, la utilización de direcciones no globales fue introducida en IPv4 al utilizar direcciones IP de los rangos privados o direcciones multicast con ambiente administrativo. El diseño de IPv6 incluye el ambiente de las direcciones en la arquitectura base. Toda dirección IPv6 tiene un ambiente especifico, que es un sector de la topología dentro del cual la dirección puede ser utilizada como un identificador único para una interfaz o un conjunto de interfaces. El ambiente de la dirección está codificado dentro de la misma. [HAGEN00]

Direcciones IPv6 unicast

En IPv6 existen direcciones unicast de varios tipos:

- *Direcciones globales*[2]*:* estas direcciones son equivalentes a las direcciones públicas de IPv4. Son ruteables globalmente y alcanzables desde la porción de IPv6 designada para Internet. Estas direcciones están diseñadas para ser “agregadas” o “agrupadas” para producir una infraestructura eficiente de ruteo. El ambiente de una dirección global es la Internet IPv6 completa y se identifican con el FP[3] 001.
- *Direcciones de enlace local:* estas direcciones se identifican con el FP 1111 1110 10 y son utilizadas por los nodos para comunicarse con otros nodos a través del mismo enlace. El ambiente de las direcciones de enlace local es el enlace local (“local link”). Se requiere una dirección de enlace local para el proceso de descubrimiento de vecinos y siempre es configurada automáticamente, aún en la ausencia de otras direcciones unicast. Un router IPv6

[2] “Aggregatable global unicast addresses”
[3] FP: “Format Prefix”, prefijo de formato.

nunca reenvía tráfico del enlace local a otros enlaces.

- *Direcciones de sitio local:* son identificadas con el FP 1111 1110 11 y son equivalentes a las direcciones el espacio de direccionamiento privado de IPv4 (10.0.0.0/8, 172.16.0.0/12 y 192.168.0.0/16). Por ejemplo, las intranets privadas que no tienen una conexión directa y ruteada con la Internet IPv6 pueden utilizar direcciones de sitio local sin tener conflictos con las direcciones globales. Las direcciones de sitio local no son alcanzables desde otros sitios y los routers no deben reenviar el tráfico de direcciones de sitio local fuera del mismo. El ambiente de las direcciones de sitio local es el sitio en cuestión.
  A diferencia de las direcciones de enlace local, las direcciones de sitio local no se configuran automáticamente, y deben ser asignadas por el administrador.
- *Direcciones especiales:*
  - Dirección no especificada: esta dirección solo se utiliza para indicar la ausencia de una dirección. Es equivalente en IPv4 a la dirección no especificada 0.0.0.0. Se utiliza como dirección fuente cuando todavía no se ha podido determinar una dirección única, pero nunca se utiliza como dirección destino ni se puede asignar a una interfaz. Su notación es “0:0:0:0:0:0:0:0” o “::”
  - Dirección de “loopback”: se utiliza para identificar la interfaz de “loopback” que de manera similar a IPv4 se utiliza para que un nodo pueda mandar tráfico a si mismo y es equivalente a la dirección 127.0.0.1. Los datagramas enviados a la dirección de loopback nunca deben ser enviados por un enlace o reenviados por los routers. La notación utilizada es “0:0:0:0:0:0:0:1” o “::1”
- *Direcciones para compatibilidad:* para facilitar la migración de IPv4 a IPv6 y la coexistencia de ambos tipos de hosts, se definen las siguientes direcciones:
  - Dirección compatible con IPv4: esta dirección, 0:0:0:0:0:0:w.x.y.z o ::w.x.y.z (donde w.x.y.z es la representación en notación decimal separada por puntos de una dirección IPv4 pública) es utilizada por nodos IPv6/IPv4 que se comunican con IPv6 a través de una infraestructura IPv4 que utiliza direcciones públicas IPv4, como por ejemplo Internet.
  - Dirección mapeada de IPv4: esta dirección, 0:0:0:0:0:FFFF:w.x.y.z o ::FFFF:w.x.y.z es utilizada para mapear un nodo que solo usa IPv4 a un nodo IPv6.
  - Dirección 6 sobre 4: estas direcciones son del tipo [prefijo de 64 bits]:0:0:WWXX:YYZZ, donde WWXX:YYZZ es la representación hexadecimal de w.x.y.z (dirección IPv4 pública o privada) y se usan para representar un nodo en el mecanismo de túnel conocido como 6 sobre 4 (“6over4”).
  - Dirección 6 a 4: direcciones del tipo 2002:WWXX:YYZZ:[SLA ID]:[Interface ID], donde WWXX:YYZZ es la representación hexadecimal de w.x.y.z (dirección IPv4 pública o privada) y se usan para representar un nodo en el mecanismo de túnel conocido como 6 a 4 (“6to4”).
  - Dirección ISATAP: direcciones del tipo [prefijo de 64 bits]:0:5EFE:w.x.y.z que se usan para representar un nodo en el mecanismo de asignación de direcciones conocido como “Intra Site Automatic Tunnel Addressing Protocol” (ISATAP).

Multicast en redes Ethernet

Ethernet es, hoy en día, la tecnología de red de área local (LAN) mas utilizada en el mundo. Las ventas de tarjetas de interfaz de red (NICs), repetidores (hubs), switches se miden por cientos de millones y las cifras continuan ascendiendo [SPURG00]. Por esto, y debido a la aplicación final de este trabajo en una organización que basa sus redes locales en tecnología Ethernet, solo nos concentraremos en estudiar una solución multicast sobre esta tecnología.

Una red LAN (Red de Area Local) puede fácilmente proveer direccionamiento multicast y broadcast en el nivel de capa de enlace del modelo OSI [STALL00]. En particular las redes Ethernet implementan estos direccionamientos utilizando direcciones especiales para indicarlos dentro de sus tramas (“frames”). La mitad de las direcciones Ethernet se reservan para multicast - el bit menos significativo del octeto de mayor orden se utiliza para distinguir las direcciones “unicast” (0) de las direcciones “multicast” (1). En notación hexadecimal con puntos, el bit de multicast se representa:

$$\mathbf{01.00.00.00.00.00}_{16}$$

Cuando se inicializa una tarjeta de red Ethernet, la misma comienza a aceptar paquetes destinados a su dirección de hardware o a la dirección de broadcast de Ethernet. Sin embargo, los controladores de dispositivos (software) pueden reconfigurar la misma para aceptar también tramas destinadas a una o mas direcciones de multicast.

Multicast en redes IP

El multicast a nivel IP es la transmisión de un datagrama IP a un “grupo de estaciones”, que es un conjunto de cero o mas equipos identificados por una única dirección IP. Un datagrama multicast es entregado a todos los miembros del grupo de estaciones destino con la misma definición de confiabilidad de tipo “mejor esfuerzo” que los datagramas IP unicast convencionales. Esto significa que no se garantiza que dicho datagrama se reciba intacto en todos los miembros del grupo de estaciones o en el mismo orden relativo a los demás datagramas.

También se puede describir como la abstracción utilizada en una internet del multicast a nivel hardware. Sigue con el paradigma de permitir la transmisión a un subconjunto de estaciones, pero generaliza el concepto para permitir que este subconjunto se pueda repartir a través de varias redes físicas en forma arbitraria a lo largo de una internet. En terminología IP, a un subconjunto dado se lo conoce como “grupo multicast”. El multicast IP tiene las siguientes características:

- *Direcciones de grupo:* cada grupo multicast es representado por una única dirección de clase D. Unas pocas direcciones multicast son asignadas de manera permanente por la autoridad que maneja las direcciones en Internet y corresponden a grupos que siempre deben existir por mas que no tengan miembros. Otras direcciones son temporales y se encuentran disponibles para uso privado.
- *Número de grupos:* el protocolo IP provee direcciones para un número de 2^{28} grupos multicast simultáneos. Por lo tanto, en la práctica, el número de grupos está limitado por el tamaño de las tablas de ruteo y no por el esquema de direccionamiento en si mismo.
- *Pertenencia dinámica de grupo:* una estación puede unirse a un grupo o dejarlo en cualquier momento que desee. Además, una estación puede ser miembro de un número arbitrario de grupos.
- *Uso del hardware:* si el hardware que existe por debajo de la capa de red soporta multicast, IP

utiliza multicast por hardware para enviar multicast IP. Si el hardware no soporta multicast, IP utiliza broadcast o unicast para enviar multicast IP.

- *Enrutamiento entre redes:* debido a que los miembros de un grupo multicast pueden estar conectados a distintas redes físicas, existen “routers multicast” especiales que se ocupan de reenviar los datagramas multicast. Esta característica, usualmente, es agregada a los routers convencionales.
- *Semántica de entrega:* el IP multicast utiliza la misma semántica de mejor esfuerzo que usa IP para la entrega de los datagramas; esto significa que los datagramas multicast se pueden perder, demorar, duplicarse o entregarse fuera de orden.
- *Pertenencia y transmisión:* cualquier estación puede enviar datagramas a un grupo multicast; la pertenencia al grupo solo determina si la estación debe recibir o no los datagramas.

Direcciones IP Multicast

Las direcciones multicast en IP se dividen en dos tipos: aquellas que se asignan de forma permanente y otras que están disponibles para uso temporal. De la misma manera que el multicast por hardware, el multicast IP utiliza la dirección destino del datagrama para especificar que el datagrama debe ser entregado via multicast. Como ya hemos visto, IP reserva las direcciones de clase D para los grupos multicast.

Los primeros 4 bits contienen 1110 e identifican la dirección como multicast. Los restantes 28 bits especifican un grupo multicast en particular. No hay mas estructura en los bits del grupo, es decir el mismo no se particiona en campos que especifiquen otro tipo de información.

Cuando las expresamos en notación decimal puntuada, las direcciones multicast tienen el siguiente rango:

224.0.0.0 a 239.255.255.255

Sin embargo, muchas partes de este espacio tienen asignado un significado especial. Por ejemplo, la dirección mas baja, 224.0.0.0, está reservada y no puede ser asignada a ningún grupo. Además, el resto de las direcciones hasta 224.0.0.255 se utilizan para el ruteo multicast y para los protocolos de mantenimiento de los grupos; los routers tienen prohibido reenviar datagramas enviados a cualquier dirección de ese rango.

Mapeo de multicast IP a multicast Ethernet

A pesar de que el estándar multicast IP no especifica todos los tipos de hardware de red, si especifica como mapear una dirección IP multicast a una dirección multicast Ethernet. El mapeo es eficiente y fácil de entender: para mapear una dirección IP multicast a su correspondiente dirección multicast Ethernet, ubicar los 23 bits menos significativos de la dirección multicast IP dentro de los 23 bits menos significativos de la dirección multicast Ethernet especial $01.00.5E.00.00.00_{16}$.

Este mapeo no es único ya que las direcciones multicast IP tienen 28 bits que identifican el grupo multicast; mas de un grupo multicast pueden ser mapeados a la misma dirección multicast Ethernet. Los diseñadores crearon este esquema ya que no tuvieron opción, fue el espacio de direcciones que les asignaron cuando se adquirieron al IEEE. Era un grupo de investigadores y no se tenía seguridad del resultado del desarrollo que iban a realizar. El esquema fue un compromiso entre usar la mayor parte posible de la dirección multicast (23 bits) y permitir que IP utilice una parte fija del espacio de direccionamiento de multicast en Ethernet. Así, el debugging sería mucho mas sencillo y se elimina la interferencia entre IP y otros protocolos que pueden compartir una red Ethernet.

La consecuencia de este diseño es, que algunos datagramas multicast pueden ser recibidos en una estación en la cual no se debían recibir. Por esto, el software de la capa de red IP debe chequear cuidadosamente las direcciones destino multicast para todos los datagramas recibidos y descartar los datagramas multicast no deseados.

Multicast en IPv6

En IPv6, el tráfico multicast opera de la misma forma que en IPv4. Nodos arbitrariamente ubicados pueden esperar por tráfico multicast en direcciones multicast IPv6 arbitrarias. Los nodos IPv6 pueden “escuchar” en múltiples direcciones multicast simultáneamente. Los nodos pueden unirse o dejar un grupo multicast en cualquier momento.

Las direcciones multicast en IPv6 tienen el FP 1111 1111. Por lo tanto una dirección multicast IPv6 siempre comienza con FF. Las direcciones multicast no pueden ser utilizadas como direcciones fuente ni tampoco como direcciones intermedias en un encabezado de ruteo. Además del FP, las direcciones multicast incluyen una estructura adicional que identifica “flags”, el ambiente y el grupo multicast.

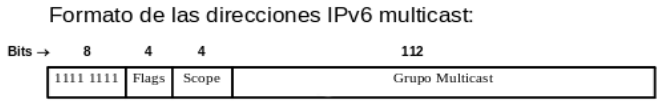


Ilustración 2: Formato de direcciones multicast en IPv6

- Flags: 000T, donde:
  - ➔ T=0: dirección asignada de forma global y permanente (IANA[4])
  - ➔ T=1: dirección asignada de forma local y temporal.
- Scope (0-F): valor que indica el ámbito o el alcance de la emisión. Puede haber 16 ámbitos diferentes.
- El grupo multicast puede ser cualquiera

[4] IANA: Internet Assigned Number Authority

Equivalencia de ámbitos multicast en IPv4-IPv6

El campo “Scope” permite 16 ámbitos diferentes, pero, por el momento, sólo se ha asignado significado a unos pocos valores de este campo, como se indica en la siguiente tabla:

Scope IPv6	Ambito	Direcciones IPv4 (RFC 2365)
0	Reservado	
1	Nodo	
2	Nivel de enlace (LAN)	224.0.0.0/24 239.255.0.0/16
3	(sin asignar)	
4	(sin asignar)	
5	Ubicación (ej. Campus)	
6	(sin asignar)	
7	(sin asignar)	
8	Organización	239.192.0.0/14
9	(sin asignar)	
A	(sin asignar)	
B	(sin asignar)	
C	(sin asignar)	
D	(sin asignar)	
E	Global	224.0.1.0-238.255.255.255
F	(sin asignar)	

Tabla 1: Valores del campo scope (IPv6)

Capítulo 2: Multicast confiable

Transmisión confiable

El término ***"confiable"*** en la transmisión de datos significa que el protocolo en cuestión debe poder cumplir con los siguientes puntos:

- Recuperación de pérdidas
- Entrega ordenada
- Sin duplicados
- Aislamiento de fallas independientes

Los paquetes que se transmiten en una red del tipo "mejor esfuerzo" (como lo son las redes IP) pueden sufrir pérdidas por una variedad de razones: sobrecarga de colas en los switches ocasionadas por congestión, colisiones repetidas en medios compartidos, fallos en el ruteo, etc. Adicionalmente, los paquetes pueden arribar fuera de orden en el destinatario porque algunos paquetes de la secuencia pueden haber seguido una ruta diferente o porque algún elemento de red intermedio ha reordenado los paquetes por alguna razón. También se experimentan demoras muy variables, especialmente al encontrar colas de transmisión y, en algunos casos, la red puede llegar a duplicarlos.

Muchas aplicaciones necesitan semánticas confiables de transmisión de datos para funcionar correctamente y recibir una secuencia de paquetes sin pérdidas, ni duplicados y en el orden exacto en el que fueron transmitidos. Un ***protocolo de transporte confiable*** resuelve estas cuestiones y oculta dicha complejidad proporcionando una abstracción confiable a las aplicaciones.

Se han desarrollado una gran cantidad de protocolos de transporte confiables. Todos utilizan las mismas ideas para cumplir con los requisitos de confiabilidad: redundancia para resolver pérdidas de paquetes, almacenamiento (buffers) en los receptores para poder reordenar y el uso de temporizadores y retransmisiones.

Una de las ideas básicas que se desarrollaron para lograr la confiabilidad es el diseño de protocolos "stop-and-wait". La idea es sencilla, el transmisor agrega un encabezado a cada paquete (distinto del encabezado de capa de red) que incluye un identificador unívoco para cada paquete (número de secuencia). El receptor, al recibir el paquete con identificador de secuencia x, deberá enviar una confirmación ("acknowledgement"[5]) al transmisor con el número de secuencia x. De esta manera el receptor le comunica al transmisor que ha recibido correctamente el paquete con número de secuencia x y el transmisor enviará el siguiente paquete de la secuencia (x+1), si y solo si, ha recibido el ACK para el paquete x. Si el transmisor, luego de un período de tiempo (llamado "timeout") no recibe la confirmación ACK del paquete correspondiente, entonces retransmitirá el paquete.

[5] En adelante, usaremos "acks" como forma corta para referenciar las confirmaciones ("acknowledgements").

Obviamente, las retransmisiones pueden generar paquetes duplicados en los receptores, ya que por ejemplo, al perderse un ACK, el transmisor asume que debe retransmitir un paquete que en realidad el receptor recibió correctamente. Debido a esto, es necesario que los receptores lleven un registro de cuál fue el último número de secuencia de los paquetes que entregaron a la capa de aplicación.

Multicast Confiable

El término “multicast confiable” se refiere a cualquier sistema que utiliza entrega multicast, pero además garantiza que todos los miembros del grupo reciban los datos sin pérdidas, duplicación o corrupción de los mismos. En teoría, el multicast confiable combina la ventaja de un esquema de reenvío que mucho mas eficiente que el broadcast con la ventaja de que todos los datos lleguen intactos. Por esto, el multicast confiable tiene muchos beneficios potenciales y un gran espectro para su aplicación.

La comunicación multicast confiable se está volviendo cada vez mas importante, especialmente para aplicaciones como conferencias multimedia, sistemas de archivos replicados, simulación interactiva distribuída y muchas otras. Debido a que las aplicaciones grupales de comunicación tienen una amplia variedad de requerimientos de confiabilidad, se han desarrollado múltiples protocolos multicast confiables, ninguno de los cuáles es un estándar dominante asi como lo es TCP para las transmisiones unicast confiables. [SPEAK00]

En la práctica, el multicast confiable no es tan general y sencillo como parece. Primero, si un grupo multicast tiene múltiples estaciones que pueden enviar, la noción de entrega de los datagramas en orden carece de sentido. Segundo, los mecanismos de reenvío multicast (p. ej. RPF) pueden producir duplicados, incluso en pequeñas redes del tipo internet.

Existen varios mecanismos para poder obtener la confiabilidad en los protocolos Multicast:

- Transmisión repetitiva: (ciclo abierto)
- FEC[6] a nivel paquete (proactivo)
- FEC a nivel paquete (reactivo)
- Redundancia provista a nivel aplicación
- Retransmisión (a partir de NACK o falta de ACK)
    1. ACK: hacer que todos los receptores confirmen la recepción de los paquetes
    2. NACK: hacer que los receptores solo informen cuando detectan pérdida de paquetes

Si es necesario conocer que absolutamente todos los receptores recibieron todos los datos, entonces es probable que se necesite un protocolo basado en ACKs. Por otro lado, si para la aplicación en cuestión no es importante si alguno de los receptores no recibió la totalidad de los datos, entonces los protocolos

[6] FEC: “Forward Error Correction”

basados en NACKs pueden ser mas adecuados ya que escalan mucho mejor.

Normalmente, estas técnicas también pueden ser combinadas para lograr esquemas confiables.

Mecanismos basados en ACKs

Debido a que un grupo multicast puede tener un arbitrario número de miembros y a que los protocolos tradicionales utilizan acks para garantizar confiabilidad, esto requeriría que el transmisor pueda manejar un número arbitrario de acks. Desafortunadamente, esto deriva en un problema conocido como “ACK implosion”. Este problema ha sido el foco de múltiples investigaciones en los últimos años y existen varias estrategias para solucionarlo o, al menos, intentar disminuirlo.

Para superar el problema de la implosión de las confirmaciones, los protocolos de multicast confiable aplican una aproximación jerárquica en donde el multicast se restringe a una única fuente. Antes de enviar los datos, se establece un árbol de reenvío (“forwarding tree”) desde la fuente hasta todos los miembros del grupo y se determinan puntos de confirmación (“acknowledgement points”).

Un punto de confirmación consiste de un router en el árbol de reenvío que acepta realizar copias de caché de los datos y procesar las confirmaciones de los routers o las estaciones que estén por debajo en el árbol. Si se requiere una retransmisión, el punto de confirmación obtiene una copia desde su caché.

Otra estrategia para mejorar los protocolos basados en ACK es la de combinar varios ACKs en un solo paquete.

Mecanismos basados en NACKs

Muchos esquemas de multicast confiable, en vez de utilizar confirmaciones positivas (ACKs), utilizan confirmaciones negativas (NAKs); esto quiere decir que las estaciones receptoras no responden salvo que algún datagrama se pierda o no sea recibido. Este esquema permite una mejor escalabilidad para el ambiente en el que usamos multicast ya que se pasa la carga de control de errores del transmisor a los destinatarios. La estación que transmite envía los datagramas sin esperar las confirmaciones. La idea es evitar el envío de mensajes de estado (ACKs) cuando todo está normal y así mejorar la eficiencia del protocolo [DIOT00].

Para permitir a una estación detectar que un datagrama no ha sido recibido, se debe asignar a cada uno un número único de secuencia. Cuando la estación detecta una pérdida envía un NACK para solicitar la retransmisión del datagrama perdido. El NACK debe ser propagado hacia arriba en el árbol (en dirección a la fuente) hasta que el mismo llega a un punto de confirmación. En este punto se procesa el NACK y se retransmite el datagrama hacia abajo en el árbol.

Cada punto de confirmación utiliza el mismo esquema para asegurarse de que tiene una copia de todos los datagramas de la secuencia. Eventualmente, al seguir cada uno de los puntos de confirmación, llegaremos a la fuente de la transmisión.

En este esquema, el diseño de la topología y los puntos de confirmación son cruciales para el éxito del esquema de multicast confiable. Sin los suficientes puntos de confirmación, un datagrama perdido

puede causar una implosión de NACKs. En particular, si un router tiene demasiados descendientes, un datagrama perdido puede causar que el mismo sea sobrecargado con pedidos de retransmisión. Desafortunadamente, la selección automática de los puntos de confirmación es un problema complejo y muchos protocolos multicast confiables requieren configuración manual. Por esto, la transmisión multicast se aplica mejor a servicios que tienden a existir por largos períodos de tiempo, topologías que no cambian rápidamente y situaciones donde los routers intermedios acuerdan servir como puntos de confirmación [COMER00].

Como vemos, los mecanismos basados en NACKs pueden sufrir también problemas de implosión, los cuales son resueltos con técnicas de supresión de NACKs y/o soluciones basadas en temporizadores, como hace por ejemplo el protocolo SRM.

Grupo de trabajo RMT

La IETF[7] formó el grupo RMT (Reliable Multicast Transport Working Group) para establecer un conjunto estándar de protocolos de transporte multicast confiables. Este grupo creó un conjunto de directivas que los desarrolladores de los protocolos propuestos deberían satisfacer. Una vez implementados, se espera que sean de uso general para una amplia variedad de aplicaciones.

Es claro que diferentes aplicaciones van a requerir diferentes aspectos de confiabilidad. A diferencia del modelo de transporte unicast, donde TCP es prácticamente universal, es poco probable que un único protocolo de transporte multicast confiable sea aceptable para todo tipo de aplicaciones. Por esto, el grupo de trabajo se ha concentrado en crear módulos de protocolo que pueden ser integrados entre sí para proveer diferentes formas de confiabilidad.

El foco del grupo RMT apunta a tres tipos de módulos:

- “Nack Oriented Reliable Multicast Protocol” (NORM) que utiliza confirmaciones negativas (NAKs) para garantizar confiabilidad. Este protocolo evita el problema de la implosión de los NAKs haciendo que todos los receptores también escuchen y procesen los NAKs de otras estaciones y/o utilizando demoras probabilísticas para evitar NAKs redundantes. NORM puede trabajar en modelos multicast con un único transmisor y también con múltiples transmisores. Este protocolo seguramente será mas útil en aplicaciones de transferencia de archivos.
- “TRee ACKnowledgement based protocol” (TRACK) que utiliza un árbol para controlar la retro-alimentación y las “reparaciones”. Este protocolo tiene la característica de devolver al transmisor una confirmación de cada receptor. Esto puede ser importante, por ejemplo, en aplicaciones financieras donde el receptor “debe pagar” por los datos. En este modelo el transmisor necesita una confirmación de que puede “cobrar” a los receptores. TRACK puede no ser una buena alternativa en redes donde el camino de retorno del receptor al transmisor no sea confiable, como por ejemplo enlaces satelitales.

[7] Internet Engineering Task Force: http://www.ietf.org/html.charters/rmt-charter.html

- “Asynchronous Layered Coding” (ALC) que utiliza técnicas de forward error-correction (FEC) y no requiere de retro-alimentación. FEC es una técnica donde el transmisor codifica un mensaje de N símbolos como N+P símbolos que luego son transmitidos a los receptores. El receptor solo necesita recuperar con éxito N símbolos para reconstruir el mensaje original; hasta P símbolos pueden ser perdidos o pueden corromperse en tránsito. En la práctica, la mayoría de los esquemas que usan FEC transmiten los N paquetes originales como están y crean P paquetes de paridad adicionales que pueden ser utilizados para recuperar alguno de los originales que puedan haberse perdido.

Alternativas de transporte multicast

Multicast IP

- Pros: estándar y altamente disponible. Respeta la filosofía “extremo a extremo”.
- Contras: no tiene garantías de confiabilidad, no es “amigable” con los routers.

“Scalable Reliable Multicast” (SRM)

Scalable Reliable Multicast (SRM) es un marco de trabajo de multicast confiable para usar a nivel aplicación y orientado a sesiones. En este marco de trabajo **todo** el tráfico es multicast. Este marco de trabajo puede ser resumido describiendo sus 3 tipos de paquetes:

1. “latidos” (heartbeats): cada miembro envía períodicamente un heartbeat que incluye el número de secuencia del último paquete enviado. Estos paquetes son utilizados por los demás miembros para poder detectar la pérdida de paquetes, comparando el número de secuencia del heartbeat con el número de secuencia del último paquete recibido.
2. “NACKs”: cuando se detecta un paquete perdido, se envía un paquete NACK a todos los miembros usando el mismo método de transporte que los datos originales. Como todos los miembros ven este NACK, todos pueden participar en la reparación de los datos. Esto es importante ya que descarga al transmisor original del tráfico y procesamiento de las retransmisiones y hace que el proceso de reparación sea mas rápido cuando la distancia entre el transmisor y el generador del NACK es grande.
3. “reparación” (repair): cada miembro de la sesión, mantiene un caché con los últimos paquetes de datos recibidos (y enviados) y si reciben un NACK de un paquete que tienen en el caché, retransmiten el paquete a todo el grupo como una reparación.

Para minimizar el número de NACKs y reparaciones, estas dos operaciones se preceden por un “exponential back-off”. Esto significa que en vez de enviar el paquete directamente, el cliente debe esperar un tiempo aleatorio (basado en la distancia al nodo en cuestión) y si en este tiempo otro nodo envía el paquete correspondiente, este cancela su propia transmisión.

SRM está diseñado para cumplir con la definición minimal de multicast confiable, esto es, entrega eventual de todos los datos a todos los miembros del grupo, sin asegurar un determinado orden de entrega. Los autores creen que si fuera necesario asegurar un determinado orden de entrega, sería fácil de agregar una capa sobre SRM que de esto se ocupe.
SRM está fuertemente basado en el modelo de entrega de grupo que es la pieza central del protocolo multicast IP. En multicast IP, las fuentes de datos simplemente envían a la dirección multicast del grupo sin necesitar conocimiento previo de los miembros del grupo. Para poder recibir datos enviados al grupo, los receptores anuncian que están interesados a través de un mensaje multicast "join" en la subred local y no necesitan conocimiento previo de los miembros del grupo y/o transmisores activos. Cada receptor se une o deja el grupo de manera individual, sin afectar la transmisión de los datos a cualquier otro miembro. SRM extiende el concepto de grupo multicast maximizando la información y los datos compartidos entre todos los miembros, y fortalece la individualidad de la pertenencia haciendo a cada miembro responsable por la correcta recepción de sus propios datos.
Finalmente, SRM intenta seguir los principios centrales del diseño de TCP/IP. Primero, SRM requiere sólo el modelo básico de entrega del protocolo IP – mejor esfuerzo con posible duplicación y/o re-ordenamiento de paquetes – y construye la confiabilidad en una base punto-a-punto. No se requiere ningún cambio o soporte especial de la red IP que existe por debajo. En segundo lugar, de una forma similar a TCP que setea temporizadores y ventanas de control de congestión, los algoritmos de SRM dinámicamente ajustan sus parámetros de control basándose en la performance observada dentro de una sesión. Esto permite a las aplicaciones que utilizan el marco de trabajo de SRM a adaptarse a un rango amplio de tamaños de grupo, topologías y anchos de banda de los enlaces mientras mantiene una performance alta y robusta. [FLOYD00]

- Pros: útil para replicar datos con poca probabilidad de cambios. No sufre de implosiones de ACK o NAK.
- Contras: limitado a escalabilidad de mejor esfuerzo; no puede llevar cuenta de la pertenencia a los grupos; el throughput es inestable si la tasa de fallos de la red no es desestimable.

"Reliable Multicast Transport Protocol" (RMTP)

RMTP provee una entrega ordenada y completa de una secuencia de datos desde una fuente a un grupo de receptores. RMTP está basado en una jerarquía de múltiples niveles en la cual los receptores se agrupan en jerarquías de regiones locales, con un receptor designado (DR) en cada región.
RMTP define DRs que reúnen los mensajes de estado de los nodos de la región RMTP local y proveen las reparaciones (retransmisiones de datos perdidos), si estos están disponibles. Los receptores dirigen los mensajes administrativos usando unicast hacia el DR. De esta manera, el DR provee recuperación local y consolidación del tráfico de control al siguiente DR de la jerarquía si los datos solicitados no estuvieran disponibles. Ver ilustración 3.

RMTP provee entrega confiable de una única fuente a un grupo de receptores sin saber exactamente la identidad de los mismos. Los receptores envían sus mensajes de estado periódicamente al transmisor/DR que retransmite los paquetes que se puedan haber perdido. Dado que los ACKs se envían periódicamente desde los receptores, aunque uno de estos ACKs se pierda, el transmisor/DR no debe realizar nada especial, porque otro ACK reflejando el estado actualizado será generado por el mismo receptor. Por lo tanto, la generación periódica de mensajes de estado provee al protocolo RMTP de una característica inherente de tolerancia a fallos y no debe ser considerado como una sobrecarga ya que simplifica la recuperación de errores.
El enfoque jerárquico utilizado en RMTP junto con la decisión de diseño de no llevar cuenta explícitamente de los receptores, dan un alto grado de escalabilidad al protocolo. Si algunos receptores están muy lejos del transmisor, este no debe preocuparse por estos, ya que el DR correspondiente será responsable de manejar los ACKs y de retransmitir los paquetes perdidos. Además, una característica es clave para la escalabilidad de RMTP: la información de estado que se mantiene en el transmisor es independiente del número de receptores. El precio que RMTP paga por esta escalabilidad es el caché adicional que tienen que mantener tanto el transmisor como los DRs.

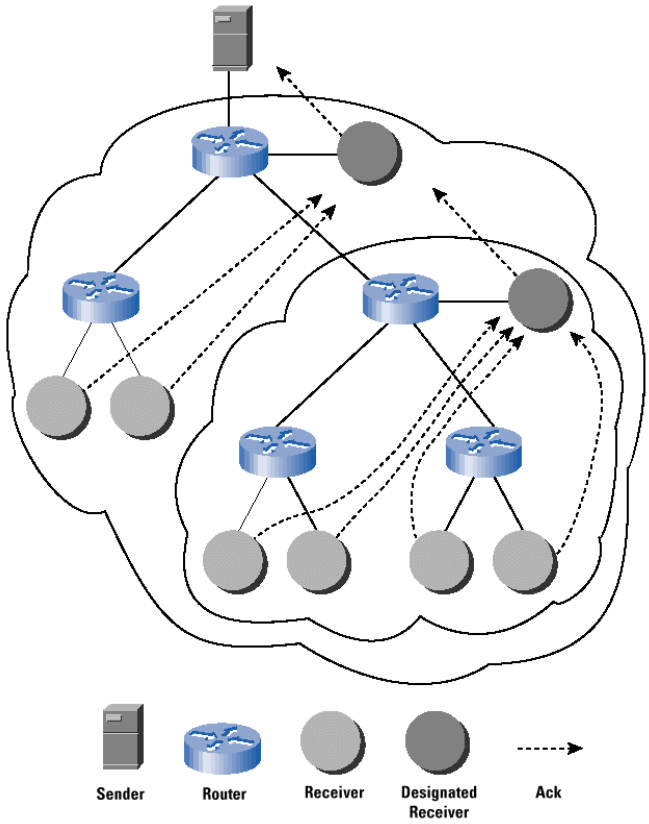


Ilustración 3: http://www.cisco.com/web/about/ac123/ac147/images/ipj/ipj_1-2/figure_rmpa5.gif

- Pros: su estructura jerárquica brinda un comportamiento predecible; se adapta automáticamente a fallos en los servidores o a servidores que no pueden alcanzarse. Util para transmisión de archivos multicast. Escalabilidad.
- Contras: comparte las mismas características que SRM. Requiere que los DRs se definan estáticamente.

"NACK-Oriented Reliable Multicast" (NORM)

Este protocolo puede proveer transporte confiable punto-a-punto de objetos o secuencias de datos sobre mecanismos genéricos de ruteo y/o reenvío de IP multicast. NORM utiliza un mecanismo selectivo de confirmaciones negativas para brindar confiabilidad en el transporte y ofrece mecanismos adicionales para permitir la operación con mínima coordinación previa entre transmisores y receptores. El

protocolo especifica un esquema de control de congestión para permitir compartir el ancho de banda disponible en las redes con otros protocolos de transporte, como TCP por ejemplo. Es capaz de operar en esquemas de ruteo multicast recíproco entre transmisor y receptores y también con conectividad asimétrica (posiblemente con un camino de retorno unicast). El protocolo ofrece una serie de características que permiten que diferentes tipos de aplicaciones u otros protocolos de transporte de nivel superior utilicen sus servicios de diferentes formas. El protocolo permite usar FEC (forward error correction) para las reparaciones y otros conceptos de “Reliable Multicast Transport” (RTM) en su diseño. En síntesis, el protocolo NORM está diseñado para proveer transporte confiable, eficiente, escalable y robusto de grandes cantidades de datos sobre una red IP multicast. El protocolo utiliza:

- una estrategia de reparación basada en NACKs, donde los receptores solicitan la reparación de datos perdidos enviando una confirmación negativa (NACK) cuando descubren pérdida de paquetes,
- mecanismos de supresión de feedback de los receptores, que permiten a los receptores cancelar el envío de mensajes de feedback superfluos al transmisor, y
- control de congestión, de forma que el transmisor puede ajustar su velocidad de transmisión de acuerdo a las condiciones de la red.

Las ventajas y desventajas de “NORM” son:

- Pros: simplicidad; el emisor no necesita conocer la cantidad de receptores; el emisor no necesita llevar cuenta del estado de los receptores; escalabilidad para aplicaciones y topologías de red donde es prohibitivo construir una infraestructura de entrega multicast sobre la capa básica IP multicast. [ADAMS00]
- Contras: como varios de los protocolos basados en NAKs, es difícil para el emisor saber cuando puede liberar los buffers de transmisión; se necesitan mecanismos adicionales a nivel de sesión si el emisor necesita saber si un determinado receptor ha recibido todos los datos.

“Pragmatic General Multicast” (PGM)

PGM, al igual que los protocolos que ya hemos descripto, es un protocolo de transporte multicast confiable que requiere entrega de datos multicast de una única fuente a múltiples receptores. PGM funciona sobre un servicio “mejor esfuerzo” de entrega de datagramas, como puede ser IP Multicast. PGM garantiza que cada receptor del grupo recibe todos los paquetes de datos de las transmisiones y/o las reparaciones, o es capaz de detectar la pérdida no recuperable de paquetes de datos.

Obtiene su escalabilidad utilizando un esquema híbrido de jerarquías, el uso de FEC, y las técnicas de eliminación y supresión de NAKs. La jerarquía se construye usando elementos de red (NEs) capaces de manejar PGM – típicamente son routers mejorados para dar soporte a PGM además de IP Multicast. De todas maneras, PGM está diseñado para operar, sacrificando parte de su eficiencia, aunque algunos o todos los NEs no estén preparados para manejar PGM.

PGM no requiere que los receptores envíen sus paquetes con transmisión multicast, lo que lo hace aplicable en redes en las que solo es posible enviar tráfico multicast desde la fuente a los receptores. PGM además hace un uso eficiente del canal de retorno en redes asimétricas que tienen un canal de alto ancho de banda en direccion fuente-receptores pero tienen un canal de ancho de banda limitado en dirección receptores-fuente.

PGM no está diseñado para ser usado en aplicaciones que dependen en la entrega confirmada a un grupo conocido de receptores o un orden total entre múltiples fuentes y no soporta estas características. PGM permite unirse y dejar los grupos en cualquier momento, proporcionando confiabilidad solo dentro de la ventana actual de transmisión. Por lo tanto, es mas adecuado para aplicaciones que soportan algun tipo de recuperación a nivel aplicación si ocurrieran eventos de pérdida irrecuperable.

PGM es hoy un RFC experimental de la IETF. En el siguiente capítulo veremos en detalle la arquitectura del protocolo PGM.

Elección del protocolo a utilizar en el presente trabajo

Como se verá en los próximos capítulos, el escenario particular al cual se orienta el presente trabajo impone ciertas restricciones y/o condiciones al protocolo a utilizar para poder implementar el esquema de control de congestión. Es necesario que el protocolo seleccionado posea al menos una implementación sobre sistemas operativos Linux, mas precisamente debería ser posible compilarla y ejecutarla sobre una plataforma que hoy ya es considerada muy vieja y se ha discontinuado: RedHat 7.x - distribución del año 2001 -.

Además, por supuesto, dicha implementación deberá ser de código abierto, ya que se necesitará disponer del código fuente para realizar las modificaciones necesarias.

Para cada protocolo analizado, las observaciones son las siguientes:

- NORM: la única implementación que se encontró disponible es una opción de protocolo de transporte para el sistema ZeroMQ (sistema de mensajería distribuída). Su documentación es prácticamente nula.

- SRM: existen en Internet algunas referencias a una librería llamada “libsrm” pero no se pudo encontrar el código fuente de la misma. Lamentablemente se tuvo que descartar esta alternativa.

- RMTP: no se encontró disponible ninguna implementación del protocolo RMTP para la plataforma requerida

- FLUTE: este protocolo podría haber sido utilizado ya que es orientado exclusivamente a archivos. Existe una implementación llamada MAD-FCL, la cual se intentó compilar para las

plataformas requeridas, pero se encontraron dificultades con algunas dependencias, ya que el mismo se ha desarrollado orientado a distribuciones basadas en Debian.

- PGM: debido a la solidez y madurez de su implementación para Linux (OpenPGM), claramente es la opción mas adecuada. Esta implementación está muy bien documentada, su desarrollo se encuentra activo, los autores dan soporte via listas de correo, el código fuente es muy claro y no tiene dependencias complicadas de compilar sobre la plataforma requerida. La librería se distribuye a partir de un paquete llamado “libpgm” cuya página de Internet es http://openpgm.googlecode.com/

En base a lo descripto en los puntos anteriores, se ha elegido como plataforma de investigación del presente trabajo al protocolo PGM y, mas precisamente a la implementación de OpenPGM para Linux como herramienta básica para realizar las modificaciones necesarias, implementar la librería resultante y, finalmente, experimentar con la misma en los escenarios reales seleccionados.

En los capitulos siguientes, se describe la arquitectura del protocolo PGM, el escenario de aplicación, la librería OpenPGM, sus particularidades y el esquema de la solución utilizada.

Capítulo 3: Arquitectura PGM

Es este capítulo describiremos la arquitectura del protocolo PGM. Esta sección no intenta ser una descripción exhaustiva, sino que intenta cubrir las características esenciales y mas interesantes del mismo. Para una descripción completa citamos [SPEAK00].

Pragmatic General Multicast (PGM) es un protocolo de transporte multicast confiable diseñado principalmente para aplicaciones que requieran transmisión de datos multicast ordenada y libre de duplicados desde múltiples transmisores hacia múltiples receptores.

La principal ventaja de PGM sobre los protocolos multicast tradicionales es que para cada receptor en el grupo, PGM garantiza que el mismo recibe todos los paquetes a partir de las transmisiones y posibles retransmisiones, o es capaz de detectar una pérdida de paquetes de datos que es irrecuperable.

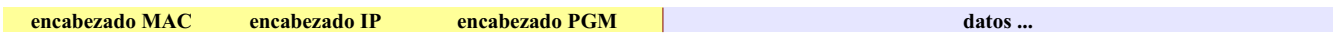
PGM se diseñó específicamente como una solución para aplicaciones multicast cuyos requerimientos de confiabilidad sean básicos. Su meta central de diseño es la simplicidad de operación siempre teniendo en cuenta la escalabilidad del protocolo y la eficiencia en el uso de la red.

Descripción general del protocolo

En PGM, no existe la noción de pertenencia a un grupo; se provee una entrega multicast confiable de datos dentro de una ventana de transmisión. El árbol de PGM se construye utilizando elementos de red (NEs) que manejan PGM sobre el árbol multicast existente. Una fuente PGM envía a los receptores paquetes de datos multicast (ODATA) en una secuencia. Cuando los receptores detectan paquetes que faltan en la secuencia, envían un paquete unicast a su padre en el árbol PGM. Los padres confirman la recepción del NAK a todos sus hijos con una confirmación multicast del NAK (paquete conocido como NCF). Los paquetes de reparación (RDATA) son generados por la fuente o por un reparador local designado (DLR) en respuesta a un NAK; esto significa que los NEs nunca guardan ODATA o proporcionan paquetes de reparación. Un paquete de reparación es el paquete perdido reenviado o un paquete FEC, dependiendo de los parámetros de la sesión. Antes de enviar un NAK, los receptores ejecutan un back-off aleatorio y suprimen el NAK si reciben el correspondiente NCF, los datos o los datos de reparación.

Paquetes PGM y encabezado

Los paquetes PGM van encapsulados dentro de datagramas IP, por lo que el esquema de encabezados y datos es el siguiente:



A PGM le corresponde, dentro de la suite IP, el identificador de protocolo número 113. La descripción

detallada del encabezado PGM es la siguiente:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Puerto fuente (source port)																Puerto destino (destination port)														
Flags								Opciones (options)								Suma de chequeo (checksum)														
Identificador global de transmisión (GSI)																														
Identificador global de transmisión (GSI)																Longitud de TSDU														
datos ...																														

Tabla 2: Detalle del encabezado PGM

- Puerto fuente: 16 bits
- Puerto destino: 16 bits
- Flags: 8 bits
  - 00..01 - Version: 2 bits – Versión de protocolo PGM
  - 02..03 – sin utilizar
  - 04..07 - Tipo: 4 bits
- Opciones: 8 bits
  - 00 – bit E – Opción de extensiones
  - 01 – bit N – Las opciones están en formato network order
  - 02..05 – sin utilizar
  - 06 – bit T – El paquete es un paquete de paridad para un grupo de paquetes de tamaño variable
  - 07 – bit P – El paquete es un paquete de paridad
- Suma de chequeo: 16 bits
  Se computa como la suma de chequeo del paquete PGM entero (incluyendo el encabezado). Si la suma computada es 0 se transmite como todos 1's. Un valor de 0 en este campo significa que el transmisor no generó una suma de chequeo. Cualquier elemento intermedio entre el transmisor y los receptores que modifique el encabezado PGM, por cualquier motivo, debe recomputar la suma de chequeo (o limpiarla). Esta suma de chequeo es mandatoria en los paquetes de datos.
- Identificador global de transmisión: 48 bits
  Un identificador global único de fuente de transmisión. Este identificador no debe cambiar durante toda la duración de la sesión de transporte. La forma recomendada de crear este identificador es usar los 48 bits bajos de la firma MD5 del nombre DNS del transmisor.
- Longitud de TSDU: 16 bits
  Es la longitud de la unidad de datos de transporte, excluyendo el encabezado de transporte. La longitud total del TPDU, se calcula sumando la longitud del encabezado de transporte (TH) mas la longitud del TSDU. La longitud del encabezado (TH) es la longitud del encabezado del paquete específico de PGM mas la longitud de las opciones que puedan estar presentes.
- Datos: longitud variable

Jerarquía

Para establecer la jerarquía PGM, los emisores emiten periódicamente mensajes de camino a la fuente (SPMs)[8]. Cada SPM contiene la dirección del nodo padre PGM desde el cual proviene. Los NEs

[8] SPM: Source Path Message

reemplazan esta dirección por la propia cuando reenvian estos mensajes para que de esta manera los hijos conozcan a sus padres en la jerarquía. Esta técnica permite que los NEs que no implementen PGM puedan trabajar en forma transparente entre nodos que si implementan PGM. Si el ruteo multicast cambia o si un nodo que implementa PGM se cambia por uno que no lo implementa, los mensajes SPMs hacen que se actualice el árbol PGM.

De esta manera, el árbol PGM se situa sobre el árbol multicast normal. Si todos los elementos implementan PGM, entonces los árboles serán idénticos. Todos los paquetes multicast son enviados con la dirección fuente del emisor PGM, incluso si provienen de algún NE. Esto asegura que los paquetes viajen por el mismo camino que los paquetes que efectivamente vienen del emisor.

Para minimizar la pérdida de NAK, PGM define un procedimiento nodo a nodo de capa de red para reenvío de los NAKs. Luego de detectar un paquete perdido, un receptor envia por unicast repetidamente un NAK a su padre en la jerarquía PGM hasta que reciba un NCF para ese paquete. Al recibir un NAK, los NEs no lo eliminan, sino que inmediatamente envían via unicast el NAK a su padre respectivo hasta que reciban un NCF. El NAK será repetido unas pocas veces en un intervalo corto de tiempo hasta que se reciba el NCF o los intentos terminen fallando. Sea o no reenviado con éxito a su padre, el NAK luego será descartado por el NE. De esta forma, el reenvío de los NAKs que realizan los NEs mejora pero no garantiza la confiabilidad de los mismos. La responsabilidad final de regenerar un NAK no confirmado recae sobre el receptor para asi mantener el principio de confiabilidad punto-a-punto.

Otra función de los NEs en PGM es realizar “anticipación de NAKs”. Esto se debe a que puede ser que reciban un paquete NCF que no corresponda a ningún NAK pendiente. Si el NCF viene del padre del NE, este registrará el NCF de manera anticipada ya que sus hijos pueden reclamar el mismo paquete. Si esto sucede, el NAK puede ser confirmado sin ninguna acción adicional, porque ya ha sido confirmado hacia arriba en la jerarquía. La anticipación de NAKs, junto con la eliminación de NAKs, colaboran en mejorar las probabilidades de que solo un NAK se reenvie hacia arriba en el árbol, cumpliendo con el objetivo de reducir la cantidad de NAKs.

Los NCFs son multicast. Sin embargo, no son propagados por los NEs de PGM, ya que actúan como confirmaciones nodo a nodo. Los nodos padres en PGM generan un NCF por cada NAK, incluso si el NAK ya ha sido confirmado previamente. Pero, la recepción de un NAK duplicado no genera que el NE emita un NAK hacia arriba si el mismo ya ha sido confirmado por su padre.

Los NEs no intentan asegurarse de que se reciba un paquete RDATA en respuesta a un NAK. Es función de los receptores, emitir nuevamente el NAK si fuera necesario. Los NEs descartan el estado del NAK luego de expirado un tiempo; esto tiene el beneficio adicional de mantener el estado de reparación relativamente adaptable a cambios en el ruteo de la red.

Cuando se envían los paquetes de reparación, los NE de PGM solo los reenviarán en las interfaces sobre las cuales han recibido algún NAK que se corresponda. Esto se conoce como “reenvío limitado

de reparaciones". De esta forma, as reparaciones solo se reenvian a los sub-árboles que contengan receptores que necesiten la reparación.

Los mensajes SPM, NCF y RDATA requieren tratamiento especial por los elementos de red en PGM. La manera obvia de descubrir estos paquetes es examinar el tipo de paquete para cada paquete del protocolo de transporte PGM. Sin embargo, la sobrecarga que esto genera en los NEs lo tornan prohibitivo. En cambio, estos mensajes se transmiten con la opción IP "router alert". Esta opción indica a los NE con una marca de capa de red que el paquete se debe extraer y analizar para un procesamiento mas detallado. Los paquetes originales de datos (ODATA) se reenvian como cualquier otro datagrama IP multicast sin ningún procesamiento especial por parte de los routers.

Supresión de NAKs

Los receptores detectan paquetes de datos perdidos a partir de "huecos" en los números de secuencia de los paquetes que reciben y transmiten de forma unicast un NAK por cada paquete faltante al NE que está en el punto superior de la jerarquía dentro del árbol de distribución del TSI[9] correspondiente. Ese elemento PGM inmediatamente transmite, usando multicast, un mensaje NCF (confirmación de NAK) por la interfaz en la que recibió el NAK. A medida que los receptores reciben el NCF, detienen el envío unicast del NAK correspondiente. Notesé que los NCFs no son propagados por los elementos de red PGM, estos confirman la recepción de un NAK dentro de un solo "salto" PGM.

Sin embargo, los receptores no envían un NAK inmediatamente al detectar una pérdida, sino que lo demoran intencionalmente por un intervalo de tiempo aleatorio. Si se recibe su correspondiente NCF, RDATA o ODATA en este intervalo de tiempo, el receptor suprime el NAK y no lo envía. Luego de enviar (o suprimir) un NAK, el receptor espera por un NCF correspondiente hasta un determinado tiempo de expiración. Luego de recibir el NCF, el receptor espera por un paquete RDATA o ODATA hasta un determinado tiempo de expiración y asi reparar la pérdida. Si expira el tiempo esperando por el NCDF o esperando por la reparación luego del NCF confirmado, vuelve a comenzar y envía otro NAK luego de otro intervalo de tiempo aleatorio con supresión.

El intervalo de espera base es especificado en los SPMs, esto permite que los padres PGM ajusten las demoras basandosé en su experiencia previa en la sesión, y en un estimado de su número de hijos. El intervalo de espera actual usado por el receptor se calcula a partir del intervalo de espera base, pero puede ser aumentado por el número de timeouts que hayan ocurrido sobre NCFs o reparaciones. El intervalo también se incrementa si el NAK generado es para mas de un paquete de paridad.

Eliminación de NAKs

Los elementos de red PGM crean un estado de retransmisión por cada NAK que reciben. Este estado de retransmisión se asocia con la interfaz de red en la cual el NAK se transmite y almacena el TSI y SQN del NAK con una lista de las interfases en las cuáles se ha recibido al menos una instancia de ese NAK.

[9] TSI: Transport Session Identifier (identificador de sesión)

Una vez que existe un estado de retransmisión almacenado para un TSI/SQN, el elemento de red PGM confirma, pero no reenvia, instancias adicionales de dicho NAK.

Debido a este mecanismo, cada elemento de red PGM, solo reenvía una instancia de cada NAK recibido.

Forward error correction

El protocolo PGM soporta código de corrección basado en Reed-Solomon a nivel paquete. Se utilizan códigos de bloque lineales para generar h paquetes de paridad a partir de k paquetes originales de datos, de forma que tomando k paquetes cualquiera, del total de (h+k) paquetes, se pueden decodificar los k paquetes originales. Los k paquetes originales se conocen como el grupo de transmisión.

FEC puede trabajar de dos formas:

- FEC proactivo: se envían paquetes FEC de manera anticipada de posibles pérdidas sin esperar por retro-alimentación de los receptores
- FEC a demanda: se generan los paquetes FEC por pedido de los receptores. Para obtener paquetes FEC a demanda, un receptor genera un NAK de paridad, que indica el número de paquetes de paridad para un grupo de transmisión

Ambos métodos de FEC son opcionales, y pueden ser combinados por el transmisor a discreción. Un NAK de paridad suprime otro NAK para el mismo grupo de transmisión si solicita un número mayor o igual de paquetes de paridad.

El uso de FEC supone una serie de ventajas:

- retransmisión eficiente: un mensaje de paridad puede reparar varios mensajes diferentes perdidos;
- supresión mas eficiente: un NAK que resulte de la pérdida de un paquete puede suprimir el NAK de algún otro paquete, si están en el mismo grupo de transmisión;
- uso mas eficiente del ancho de banda para NAKs

Ventana de transmisión

En una transferencia unicast, es obvio que el borde final de la ventana de transmisión del emisor debe avanzarse al punto en el que todos los paquetes hayan sido confirmados por el receptor. En PGM, al no tener confirmaciones de tipo ACK, esto no se puede hacer.

PGM permite al emisor avanzar arbitrariamente el borde final de la ventana de transmisión. Las implementaciones pueden soportar ajustes automáticos, como por ejemplo mantener la ventana en un tamaño fijo (en bytes o paquetes), o una duración fija en tiempo real. Adicionalmente, se puede demorar opcionalmente el avance de la ventana al existir ausencia de NAKs.

Esto permite a la aplicación determinar que ciertos datos ya no se necesitan y se pueden eliminar de la ventana, o, a partir de un período de silencio de NAKs, tomar la decisión de avanzar la ventana.

Cuando se avanza la ventana de transmisión, una fracción de la parte trasera de la ventana (llamada ventana de incremento) queda fuera del segmento disponible para retransmisiones.

Los NAKs de paquetes cuyos números de secuencia están fuera de la ventana de transmisión son confirmados con sus NCFs respectivos, pero se trasmitirán los correspondientes paquetes RDATA para reparar esas pérdidas.

Reparaciones locales (DLRs)

Además de que la fuente provea las reparaciones, PGM soporta reparadores locales designados (DLRs). Esta característica del protocolo es opcional. Los DLRs son estaciones (no NEs) que anuncian su presencia con un mensaje multicast, haciendo que los subsiguientes NAKs sean redirigidos al DLR y no al emisor. Los DLRs reciben y guardan en un caché todos los paquetes en la ventana de transmisión. Luego cumplen con los NCFs y las reparaciones de la misma forma que lo hace el emisor.

Los DLRs pueden insertarse en el esquema de PGM “hacia arriba” en el árbol y cubren al emisor o pueden estar fuera del árbol (“off-tree”) que son DLRs que están un salto PGM por debajo de un NE. Se llaman fuera del árbol, a pesar de que están por debajo del punto de la pérdida, porque pueden pertenecer a un subárbol que no ha sido afectado por la misma. Un DLR fuera del árbol responde a un NAK emitiendo un NCF via unicast y los datos de reparación a su padre. El padre luego reenvía la reparación hacia abajo del árbol.

Los DLRs “off-tree” se descubren via el mecanismo de “polling”. Los NEs de PGM luego transmiten los “polls” de multicast a la dirección de grupo de la sesión y las estaciones capaces de cumplir el rol de DLR responden con un mensaje que contiene la información de redirección. Estos mensajes de redirección son almacenados por los NEs y usados para redirigir los NAKs.

Polling

Las demoras aleatorias que preceden a los NAKs deben ser proporcionales al número de hermanos en PGM. Esto previene la implosión de los NAKs cuando la escala crece. Tan importante como escalar hacia arriba es escalar hacia abajo; esto es, reducir las demoras cuando el número de hermanos disminuye de forma que PGM no tenga mala performance con pequeños grupos.
PGM soporta sondeo (“polling”) para que los padres PGM puedan estimar el número de hijos. Basándose en los resultados del sondeo y en la experiencia de la sesión, el padre utiliza un campo del mensaje SPM para indicar a sus hijos el intervalo de demora aleatoria a utilizar.

Los sondeos son realizados al enviar un paquete multicast POLL. Estos paquetes no son reenviados por los NEs PGM ya que solo aplican a los hijos del padre PGM. Igual que los NCFs, siempre tienen la

dirección IP fuente del emisor PGM para que así utilicen el mismo árbol multicast. Cada paquete POLL indica una probabilidad de que cada hijo responda al sondeo con un mensaje de respuesta unicast. Además contiene un intervalo de tiempo aleatorio para back-off que los hijos deben cumplir antes de responder. Finalmente, cada sondeo se realiza en un número de rondas. Solo los hijos que fueron parte de la primera ronda del sondeo pueden responder a las siguientes rondas.
La probabilidad de respuesta combinada con las rondas sirve para evitar la imposión en el caso de un aumento masivo y súbito en la cantidad de hijos.

Control de congestión

En el modo básico de operación de PGM simplemente se realiza un límite en la tasa de trasferencia del emisor. Para realizar control de congestión el emisor debe recibir información acerca del estado de la transmisión. Para este fin, PGM soporta 3 tipos de feedback al emisor:

1. carga del peor enlace (medido por los NEs)
2. carga del peor camino punto-a-punto (medido por los NEs)
3. carga del peor camino punto-a-punto (medido por los receptores)

El emisor utiliza este feedback para ajustar su tasa de trasferencia. Sin embargo, PGM no especifica como se ajustará esta tasa; el esquema de control de congestión se deja a criterio de la implementación.

PGMCC

Como trabajos relacionados, se han propuesto varios esquemas de control de congestión complementarios a PGM. Uno de éstos es llamado “pgmcc”; esquema de control de congestión de tasa única (“single rate”) cuyos objetivos se centran en la escalabilidad, estabilidad y rápida respuesta a la variación de las condiciones de la red. Este esquema, en base a los resultados experimentales obtenidos en su desarrollo, es considerado amigable a TCP (“TCP friendly”) [RIZZO00].

En el esquema de tasa única que usa pgmcc, todos los receptores obtienen la misma tasa de datos y el emisor se adapta al receptor mas lento del grupo. Se conoce que estos esquemas tienen la limitación de que un único receptor lento puede tirar abajo la tasa de envío para todo el grupo.

El control de congestión que implementa PGMCC tiene un comportamiento muy similar al control de congestión de TCP, ya que utiliza un ciclo de control basado en ventana que se ejecuta entre el emisor y un receptor seleccionado, llamado “ACKER”. El rol del ACKER es proveer al emisor de una rápida retroalimentación de la misma forma que lo hace un receptor en TCP.

Capítulo 4: Escenario de Aplicación

El presente trabajo y su respectiva investigación surgen de la necesidad de mejorar la capa de comunicaciones de un sistema de punto de venta, que fué realizado con desarrollo propio y a medida, por la Gerencia de Sistemas de la Cooperativa Obrera Ltda. En este capítulo se describe dicha necesidad, el diseño e implementación de una capa de comunicaciones multicast básica que formó parte de ese desarrollo y los motivos por los cuáles es necesario mejorar la misma.

Hasta el año 2004, la Cooperativa Obrera utilizó en los puntos de venta de sus sucursales un sistema propietario y comercial, lo cual históricamente trajo una serie de dificultades y limitaciones a su operación. Debido a esto, ya en el año 2003, dentro de la Gerencia de Sistemas se decidió desarrollar un sistema propio para reemplazarlo. De acuerdo a los requerimientos del mismo, en el análisis y diseño se define la necesidad de contar con un módulo de comunicaciones. Este módulo es necesario para el envío de comandos de control a los puntos de venta, para el envío de datos y para monitoreo del estado de cada uno de los puntos de venta.

Estructura del sistema

El sistema de punto de venta que se desarrolló como parte de ese proyecto, está formado por varios componentes de hardware y software:

Componentes de hardware:

- 1 Servidor dedicado local a la sucursal (SERVER) para centralizar datos y actuar de controlador de los puntos de venta
- N Puntos de venta (POS): estas son las cajas registradoras que se encuentran en cada pasillo (checkout) de la sucursal. Internamente tienen arquitectura de PC y se conectan con una serie de dispositivos periféricos: scanner de códigos de barra, controlador/impresor fiscal, pinpad, teclado especial con lector de banda magnética, balanza, etc.
- X Terminales de consulta: PC normales de escritorio para operar el sistema
- Red local de comunicaciones (LAN): una red local Ethernet para intercomunicar estos componentes
- Router[10]: un router que conecta la sucursal con el resto de la red WAN de la organización

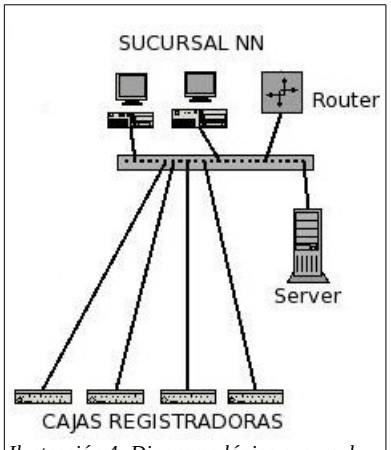


Ilustración 4: Diagrama lógico sucursal

[10] En realidad no es un componente del sistema, pero se agrega por completitud

Componentes de software:

- POSEIDON: la aplicación de punto de venta que se ejecuta en cada POS para realizar los tickets y facturas, tarea central de la operatoria comercial de la sucursal;
- ZEUS: un “demonio” que se ejecuta en el SERVER para implementar tareas de control y en especial una capa de comunicación con los puntos de venta;
- DB: una base de datos local a la sucursal almacenada dentro del SERVER;
- APOLLO: backoffice web para consultas, reportes y administración general de la sucursal que también ejecuta dentro del SERVER;

Equipamiento

En el año 2003, cuando se comenzó el desarrollo del sistema, la Cooperativa Obrera contaba con 58 sucursales con hardware de distintas generaciones y muy variado:

- Se contaba con servidores con procesadores Pentium / Pentium Pro / Pentium II y III. El procesador más lento era un Pentium 75 Mhz y el más potente un Pentium III de 700 Mhz. La cantidad de memoria RAM variaba de 64 a 256 MB y los discos rígidos de 2 a 9 GB.
- Las cajas registradoras iban de 486 de 66Mhz a K6-2 de 300 Mhz con 4 a 16 MB de RAM.
- El equipamiento de red era muy variado con muchos Hubs viejos de 10Mb/s, algunos Hubs de 10 o 100 Mb/s sin auto-negociación de sus bocas y unos pocos switches de 10/100 Mb/s.

Esta heterogeneidad de equipamiento hizo difícil el diseño del sistema ya que se contaba, en muchos casos, con equipos de recursos muy limitados. A pesar de esto, se pudo definir que, como piso, se iba a poder utilizar una distribución de Linux (basada en RedHat 7.3 pero muy personalizada) con kernels de la línea 2.4.x. Esto nos permitiría utilizar la excelente implementación del protocolo TCP/IP del kernel de Linux y realizar aplicaciones con múltiples hilos de ejecución como lo soporta la librería nativa “glibc” del sistema operativo.

Comunicaciones

En el análisis del sistema se definieron una serie de requerimientos de comunicaciones que el sistema debería cumplir. Estos requerimientos, para la mayoría de las funciones, mapeaban naturalmente a un diseño de arquitectura cliente / servidor donde cada POS se conectaría a un servicio desarrollado para tal fin que se ejecutaría en el SERVER. Estas conexiones serían TCP/IP. Estas funciones son: reporte de estado y monitoreo de cada POS y envío de comandos de control. La conexión cliente/servidor contra el SERVER, se usaría para que las POS envíen periódicamente un paquete con su estado al SERVER y recibiría comandos de control para realizar funciones específicas inherentes a la aplicación (emisión de cierre al final del día, actualización centralizada de parámetros de configuración, etc.).

Otro requerimiento de comunicación es el de la autorización “online” de transacciones. Para esto cada POS debe conectarse, también en una arquitectura cliente/servidor, con un servidor central a toda la organización que se ocupa de conectarse con las distintas empresas tarjeteras para autorizar las compras con tarjetas de crédito y/o débito (VISA, Posnet, Diners Club, American Express, etc.). La naturaleza de este requerimiento también llevó a implementar dicha función con una conexión TCP/IP desde la POS al servidor autorizador de tarjetas. El protocolo utilizado es ISO8583 con estructura request-reply.

El último, pero no menos importante, requerimiento de comunicaciones que debía implementar el sistema, era un mecanismo para poder enviar tablas de datos desde el SERVER a todas las POS y además el envío de nuevas versiones de la aplicación (archivos de varios MBs). El sistema anterior tenía esta función, pero su implementación utilizaba N conexiones unicast para el envío por lo que en sucursales con un número elevado de puntos de venta las redes colapsaban en el momento de enviar grandes tablas de datos; las transmisiones fallaban e incluso el sistema no verificaba la integridad de estas tablas al momento de ponerlas en funcionamiento, lo que generaba un verdadero caos ya que las tablas quedaban “incompletas” o directamente “corruptas”. La forma de evitar estos problemas era hacer envios manualmente a grupos de cajas de acuerdo a experiencias empíricas previas de acuerdo a la velocidad de la red y/o el tipo de cajas registradoras que tenía cada sucursal y asi estimar un número de cajas a las que se podía enviar de forma relativamente confiable. De todas maneras nunca se podía garantizar la integridad de los datos que se transmitían. Este punto, entonces, era fundamental que el nuevo desarrollo pudiera mejorarlo y solucionar los problemas que tenía el sistema anterior.

A la luz de estos inconvenientes, se decidió usar multicast para la transmisión de estas tablas ya que esto nos daría la ventaja de utilizar mas eficientemente el ancho de banda de la red. Enviar n veces el mismo contenido de las tablas con transmisiones unicast ocupa por mas tiempo el ancho de banda de la red y si no existe un control de simultaneidad de estas transmisiones las redes eran fácilmente congestionadas. Utilizar multicast en una red LAN nos daría la ventaja de realizar solo una transmisión total y, con un bajo número de reparaciones, ocupar muchos menos tiempo el ancho de banda de la red de la sucursal (la cual en muchos casos está limitada a equipamiento no switcheado y con bajas velocidades ~ 10Mb/s); esta ventaja sería mas evidente sobre la solución unicast a medida que creciera el nro n de cajas registradoras y se reduciría considerablemente el tiempo total de transmisión.

Como podemos ver en la ilustración 5, realizamos una estimación teórica de las ventajas de la transmisión multicast. Para esto tomamos una red de ancho de banda teórico de 10Mb/s, y comparamos 3 transmisiones unicast: una totalmente lineal como si se transmitiera de forma serializada un archivo detrás de otro para cada uno de los receptores asumiendo una eficiencia de red de 1MB/s; otra con un factor de retransmisión debido a congestión del 5% por receptor y otra con un factor de congestión del 15% por receptor. En los casos de multicast gráficamos 2 escenarios, uno donde existiera un 5% de reparaciones y otro donde exista un 15% de reparaciones. Evidentemente multicast tiene mucha mejor escalabilidad y suponía una ventaja indiscutible sobre la alternativa unicast; es por esto que se optó por desarrollar dicha función utilizando IP multicast.

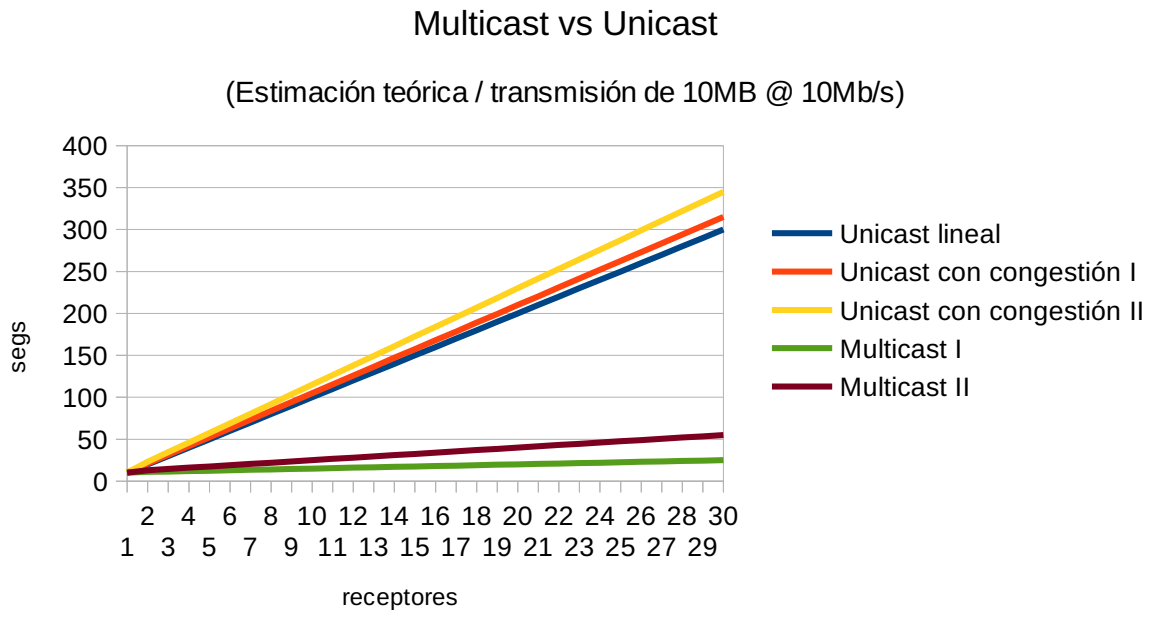


Ilustración 5: Multicast vs Unicast

Transmisión Multicast en POSEIDON / ZEUS

La transmisión multicast en el sistema se implementó de forma que un thread (*tnetcomm*) del proceso "demonio" ZEUS que se ejecuta en el servidor y que es el encargado de procesar los comandos de control y notificarlos a las POS, sería la fuente de las transmisiones multicast y un thread específico (*tmulti*) de la aplicación POSEIDON sería el receptor en cada uno de los puntos de venta. Además la aplicación POSEIDON tendría un thread adicional (*talive*) que sería el cliente de la estructura cliente/servidor y se conectaría por TCP/IP contra un pool de threads (*tserver*) del SERVER para comunicar su estado y recibir los comandos de control.

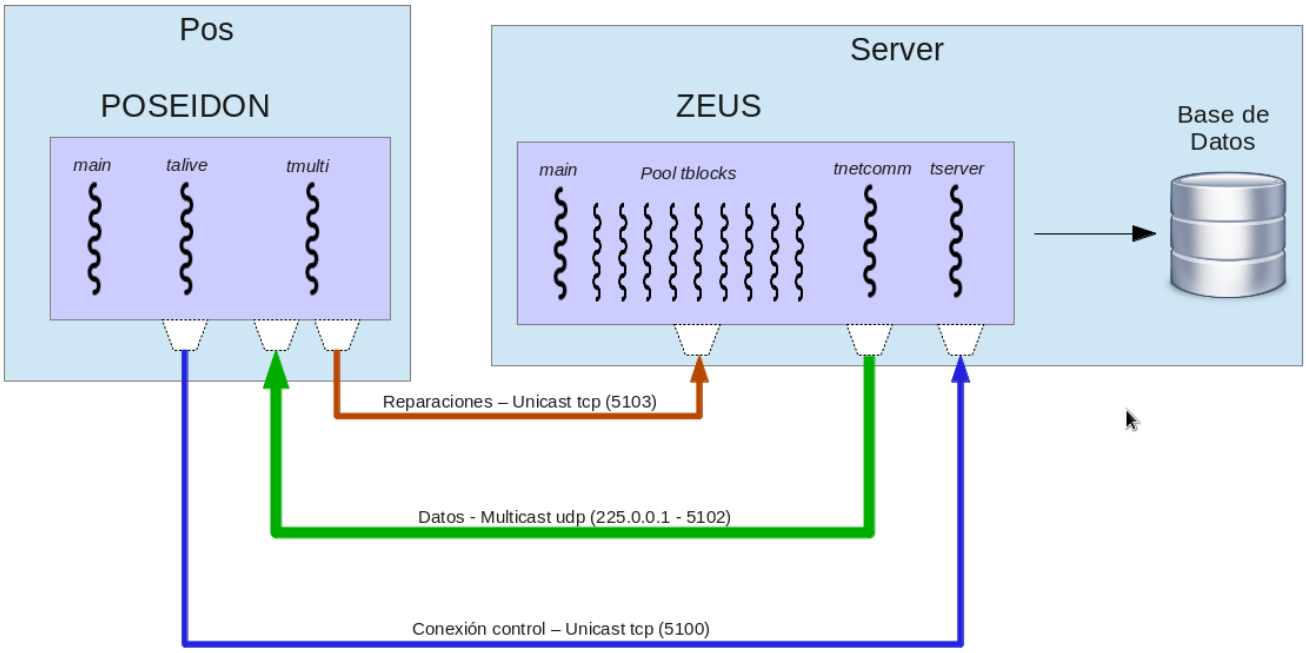


Ilustración 6: Esquema original de transmisión multicast

Las transmisiones multicast se dividirían en “sesiones” y, por simplicidad, solo podría haber activa una sesión por vez entre el SERVER y las POS. Cada sesión sería para transmitir una tabla de datos o un archivo.
Una sesión multicast en el sistema se compone de 4 etapas: preparación, transmisión, reparación y chequeo de consistencia. A continuación se decribe cada una de estas etapas en detalle.

Preparación de la transmisión

En la etapa de preparación se realizan algunas tareas previas a la transmisión y que simplificarán los pasos posteriores. Como primer medida se diferencia la sesión multicast para envío de tablas de datos del envío de archivos.

Si es una sesión para envío de tabla de datos, el demonio ZEUS se ocupa de realizar una bajada de los datos desde la base de datos a un archivo[11] temporal con el formato interno de datos[12] que utiliza el sistema POSEIDON. Luego se empaqueta el/los archivos en un archivo que será el input de la rutina de preparación. Si la sesión, en cambio, es para envío de un archivo ya determinado, el archivo es directamente el input de la rutina de preparación.
A partir de ese punto la rutina es común para ambos tipos de sesión y los pasos son:

[11] También pueden ser varios archivos si dicha tabla contiene índices
[12] Berkeley DB

- Si la sesión tiene configurada compresión, entonces se comprime el archivo input de la misma para reducir la transmisión efectiva que se hará por la red.
- Una vez que se tiene el archivo final a transmitir (comprimido o no), se calcula el tamaño total del mismo (*file_size*)
- La transmisión se realiza en bloques. Estos bloques permiten la reparación posterior, ya que los receptores pedirán a la fuente los bloques no recibidos/perdidos haciendo referencia por número de bloque. En esta etapa, se calcula el número total de bloques (*blocks*) en función de *file_size* y el tamaño de bloque (*MAX_DATA_SIZE*).
- Como último paso de esta etapa, se calcula un valor de hash (*md5sum*) de los contenidos del archivo utilizando el algoritmo MD5. Este hash se utilizará para chequear la consistencia del archivo recibido en los receptores y decidir si se utilizará luego en capa de aplicación o se debe descartar por errores en algún bloque.

Transmisión

Antes de comenzar la transmisión multicast, el demonio ZEUS envía por las conexiones de control para cada punto de venta conectado un paquete que indica el inminente comienzo de una sesión multicast. Este paquete contiene la información que se calculó en la etapa de preparación (file_size, blocks, md5sum y tabla de datos en caso de ser una sesión de envío de tabla). Una vez recibido un paquete de inicio de sesión multicast, el thread talive de la aplicación POSEIDON comunica al thread tmulti que debe iniciar una sesión de recepción multicast y le pasa los parámetros de dicha sesión.

El emisor, luego realiza un delay de inicio (delay_start) programado para incrementar las posibilidades de que la mayor cantidad posible de puntos de venta hayan recibido el paquete de inicio de sesión via la conexión de control. Una vez expirado este timer de inicio, se da comienzo a la transmisión multicast pura.

En principio la transmisión era un ciclo simple de lectura de los bloques del archivo de input y de transmisión de los mismos por el socket multicast. Pero luego de la primer implementación del esquema de comunicación notamos que algunos receptores (dependiendo de los recursos de hardware de los mismos) eran sobrepasados por la capacidad de envío del SERVER y se perdían una gran cantidad de bloques porque se llenaban sus buffers y/o no daban a basto volcando la información recibida al disco y debían descartar paquetes.

Para evitar abrumar a los receptores transmitiendo demasiado rápido, se modifico la rutina de transmisión para lograr el efecto de una tasa efectiva de transmisión y asi limitar la velocidad de envío del SERVER sobre el socket multicast. La implementación del límite en la tasa de trasferencia se realizó mediante el siguiente parámetro:

- *bwlimit*: límite a la tasa de transferencia en KB/s

Luego en el ciclo de envío, se invoca la función “throttle” para controlar que el envío no esté

sobrepasando el límite especificado por “bwlimit”.

Función Throttle

```
void throttle(u_int32_t count) {
      struct timeval tv;
      if (_bwlimit <= 0 || count <= 0) {
            return;
      }
     _byte_count += count;
     gettimeofday(&tv, NULL);
     long elapsed_milliseconds = (tv.tv_usec + 1000000 * tv.tv_sec) - _start;
     if (elapsed_milliseconds > 0) {
            // bps actuales
            unsigned long bps = _byte_count * 1000L / elapsed_milliseconds;
            // Si los bps actuales son mayores a bwlimit, aplicar throttle
            if (bps > _bwlimit) {
                  // Calcular tiempo a dormir
                  long wake_elapsed = _byte_count * 1000L / _bwlimit;
                  int to_sleep = (int)(wake_elapsed - elapsed_milliseconds);
                  if (to_sleep > 1) {
                        usleep(to_sleep);
                        _reset();
                  }
            }
      }
}
```

Reparaciones

Una vez terminada la etapa de transmisión, el receptor calcula los bloques que no pudo recibir a partir del complemento de los que si recibió y el número total de bloques, e inicia la etapa de reparación. Las reparaciones las solicita via una conexión tcp unicast al servidor ZEUS donde habrá un pool de threads que se ocuparán de proveer los bloques que los puntos de venta vayan solicitando a partir de los id de bloques. El receptor solicita y recibe uno por uno los bloques que le faltan. De existir errores de conexión, el receptor reintenta una cantidad predeterminada de veces y luego aborta con un error que reporta a la capa de aplicación.

Consistencia

Una vez recibido el archivo completo (a partir de los bloques recibidos por multicast y los bloques que se solicitaron como reparación) se chequea la consistencia del mismo calculando nuevamente el hash MD5 del contenido del archivo y comparando contra el que se envío en el paquete de inicio de sesión. Si los hashes difieren entonces se descarta el archivo recibido y la transmisión se aborta con un error que se reporta a la capa de aplicación.

Parametrización

Los parámetros a configurar por el administrador del sistema son:

- Compresión (compress): se define como un trade off entre la potencia de los cpus de las POS y la velocidad de la red. Si los cpus son muy lentos (caso 486) o la red es rápida es conveniente transmitir el archivo sin comprimir. Si los cpus son rápidos y/o la red es muy lenta, entonces conviene comprimir para ocupar menos tiempo la red.
- Tamaño de bloque (MAX_DATA_SIZE): default 4KB
- Delay de inicio (delay_start): default 1.5 secs
- Limitación de tasa de envío (bwlimit): empíricamente se determinó que con un límite de 400 KB/s se lograba una performance aceptable. En redes de 10Mb/s este es el valor utilizado.

Para facilitar la parametrización del sistema, los receptores llevan una estadística completa de tiempos invertidos en cada una de las etapas, cantidad de bloques recibidos vía multicast y cantidad vía reparaciones. En base a esto en cada sucursal se experimentaba con valores de limitación de tasa de envío para intentar encontrar el punto óptimo donde se pueda enviar multicast lo más rápido posible sin que esto redunde en que los receptores luego soliciten demasiadas reparaciones. Si la tasa de transferencia era demasiado rápida, se ajustaba para mas delay y se volvía a intentar hasta encontrar un punto satisfactorio.

Inconvenientes y limitaciones

- Parametrización estática: la parametrización inicial requería varias pruebas para ajustar las tasas de envío de acuerdo al tipo de servidor/cajas/red. Una vez parametrizado para un escenario, el protocolo no se adaptaba a los posibles cambios de la red y/o equipos.
- Falta de un mecanismo de control de congestión: la parametrización inicial asumía un 100% de disponibilidad de la red para la transmisión multicast. Esta premisa no siempre se cumplía en los momentos donde se transmitía y la utilización de la red por otras aplicaciones puede generar congestión y/o ser afectada por la transmisión multicast.
- Reparación unicast genera duplicaciones: este esquema de reparación tiene la desventaja que un mismo bloque perdido puede ser transmitido hasta N veces en el peor caso si ese que los N puntos de venta no lo pudieron recibir
- Validez de las reparaciones: las reparaciones que proveen los threads de ZEUS que están para esa función son válidas hasta el comienzo de una nueva sesión. Esto tiene como desventaja que

si comenzamos una nueva sesión demasiado rápido un receptor puede recibir como reparaciones bloques que no corresponden al archivo original. Por supuesto que la integridad del archivo completo será adulterada y detectada en el control de consistencia, pero a esta altura la transmisión fue en vano.

A pesar de que se cumplió el objetivo de implementar estos envíos con transmisión multicast y los requerimientos fueron satisfechos, estos inconvenientes hacen que la eficiencia de transmisión del protocolo no sea óptima y su rendimiento sea muy dependiente de la configuración inicial y de las características de cada escenario, que por supuesto eran totalmente distintas de sucursal en sucursal.

Por consiguiente el objetivo del presente trabajo es utilizar PGM para reformar la estructura de transmisión multicast de este sistema e intentar solucionar o reducir estos inconvenientes.

Capítulo 5: OpenPGM

Introducción

OpenPGM es una implementación de código abierto de PGM en base a la especificación RFC 3208 disponible en www.ietf.org. PGM es un protocolo multicast confiable y escalable que permite a los receptores detectar pérdidas, solicitar retransmisiones de datos perdidos o notificar a la aplicación de pérdidas irrecuperables. PGM corre sobre un servicio de transmisión de datagramas con semántica de entrega mejor-esfuerzo; actualmente OpenPGM utiliza IP, pero puede ser implementado sobre un modelo conmutado como por ejemplo: InfiniBand.

Plataformas Soportadas

Actualmente, las plataformas preferidas son Solaris 10 sobre SPARC64, Ubuntu 8.04 a 12.10 y Debian 7 sobre AMD64, OS X 10.6 y 10.7, FreeBSD 8.0, OpenSolaris 2009.06 y Windows Server 2008R2. La plataforma mínima en x86 es la micro arquitectura P5 con timers de alta resolución y 80486 se puede utilizar con timers de baja resolución y operaciones atómicas.

La operación de PGM/IP requiere privilegios especiales (administrador) en Unix y Microsoft Windows Vista o superior. El encapsulamiento sobre UDP, etiquetado como PGM/UDP, no requiere privilegios especiales y es soportado en todas las plataformas, excepto los routers PGM.

Las siguientes plataformas no han sido testeadas: OS X 10.8, Solaris 11, AIX, HP/UX, Windows Server 8, Windows RT, iOS, Android.

Conceptos

OpenPGM facilita el desarrollo de aplicaciones distribuídas escalables que intercambian datos a través de la red y con asistencia de la misma. OpenPGM es un transporte de datos, no impone ningún formato de mensaje o lógica de comunicación de alto nivel. OpenPGM puede ser ejecutado en varias plataformas de hardware y de software permitiendo así una plataforma de red heterogénea.

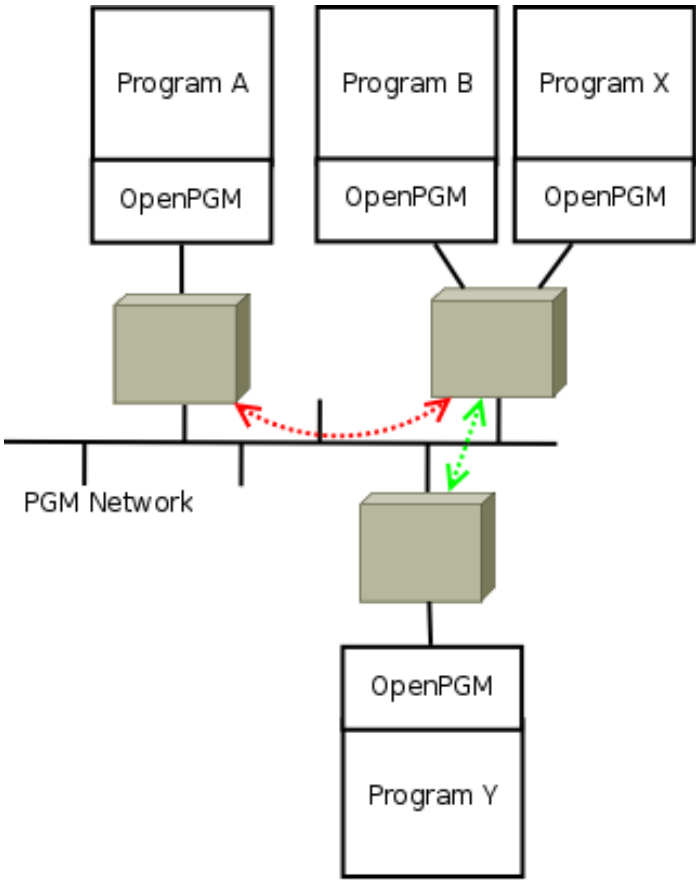


Ilustración 7: http://miru.hk/wiki/OpenPGM_operating_environment.png

Desde el punto de vista del programador, OpenPGM provee una API que permite integrar el sistema de comunicaciones PGM dentro de la aplicación. OpenPGM no interfiere, ni reemplaza, los sistemas existentes de comunicación entre procesos y no existe ningún proceso externo tipo “daemon” que tenga que arbitrar múltiples aplicaciones en un nodo que se comunique por el mismo transporte de red.

OpenPGM, actualmente, define una interfaz de lenguaje basada en la librería de bajo nivel “Glib[13]”.

Cada nodo puede ejecutar varios programas OpenPGM, sin embargo ciertas restricciones del protocolo PGM hacen que un transporte no pueda compartirse entre varios programas/procesos.

Transporte

En OpenPGM el mecanismo de software para enviar y entregar mensajes es llamado “transporte”. Un transporte define el ambiente de entrega – esto es el conjunto de destinos posibles para un mensaje que se envía.

[13] GNOME Library. Esta librería provee objetos básicos para construir aplicaciones y librerías escritas en lenguaje C.

Parámetros de transporte de red

Las llamadas para crear los transportes esperan tres parámetros que gobiernan el comportamiento del transporte: puerto de datos destino (data destination port), red de recepción (receive network), y red de envío (send network). En escenarios simples, es suficiente un grupo multicast para enviar y recibir, pero otros escenarios pueden requerir configuraciones mas complejas:

- Varias aplicaciones distribuídas independientes comparten la misma red y deben ser aisladas entre sí (data-destination port)
- Un programa corre en un nodo con mas de una interfaz de red y se debe seleccionar una red específica para la comunicación multicast saliente (send network)
- Los nodos de la red utilizan direccionamiento multicast asimétrico para lograr mayor eficiencia, y se debe especificar a que grupos multicast se debe unir (parámetros de red)

Puerto de datos destino

Los puertos de datos destino dividen la red en particiones lógicas. Cada transporte se comunica en un par de puertos fuente y destino; un transporte solo puede comunicarse con otros transportes en el mismo puerto de destino. Un programa que necesite comunicarse en mas de un puerto, debe crear un objeto de transporte separado para cada puerto. Un número de puerto es un entero decimal que tiene las mismas restricciones de rango que en los sockets IP (1-65535).

Puerto de datos fuente

Los puertos de datos fuente son el extremo receptor de PGM y necesitan ser únicos para cada aplicación dentro de un nodo. De manera predeterminada es un número generado aleatoriamente dentro del rango 1-65535, sin embargo puede ser especificado manualmente si queremos que la aplicación se una nuevamente a sesiones previas o asegurar que no exista conflicto entre varias aplicaciones simultáneas en el mismo nodo.

Parámetros de red

El transporte de OpenPGM acepta una lista de interfaces de recepción y direcciones de grupo multicast y un solo par de interfaz de envío y dirección multicast.

Para simplificar la configuración del usuario final/desarrollador, existe una función de ayuda para interpretar las definiciones del parámetro de red en el formato interno utilizado por OpenPGM.

La red predeterminada es la dirección multicast con ambiente administrativo 239.192.0.1 para IPv4 y ff08::1 para IPv6.

Construyendo el parámetro de red

El formato del parámetro de red consiste de tres partes separadas por “;” - interfaz, grupos multicast y

dirección de envío. Ejemplos:

```
eth0                                    interfaz
eth0;225.0.0.1                          un grupo multicast
eth0;225.0.0.1,225.0.0.5;225.0.0.6      dos grupos multicast, una dirección de envío
```

Parte uno – interfaz

La primer parte identifica la interfaz de red que se puede especificar de varias maneras:

- Nombre de interfaz: por ejemplo eth0, eth1, em0, etc.
- Dirección IP: una dirección estándar IPv4/IPv6, por ejemplo: 4.2.2.2, “abcd::1”, “[b00c::2]”
- Red IP: un prefijo de red IPv4/IPv6 multi parte con con tamaño de bloque CIDR opcional e identificador de ámbito; por ejemplo: “10.4”, “127.1/8”, “beef::”, “fe80::/64%eth0”
- Nombre de host: un nombre de host que se pueda resolver mediante NSS[14], por ejemplo “localhost”
- Nombre de red: un nombre de red que se pueda resolver mediante NSS, por ejemplo “localnet”
- Default: la interfaz predeterminada según las tablas de ruteo multicast.

Parte dos – grupos multicast de recepción

La segunda parte es una lista de uno o mas grupos multicast a los que el transporte se debe unir, especificados como direcciones IP o nombres de DNS separados por comas. Cada una de estas direcciones debe especificar un grupo multicast válido. Unirse al grupo multicast permite a los receptores recibir datos enviados a dicho grupo.

Parte tres – grupo multicast de envío

La tercera parte es una única dirección de envío. Cuando un programa envía datos multicast a través del transporte resultante es enviada a esta dirección. La dirección de envío no puede ser alguna de las direcciones de la lista de la parte dos. Si el transporte se une a uno o mas grupos multicast en la parte dos, pero no especifica la tercera parte, la dirección de envío se setea de forma predeterminada a la primera dirección de la lista de la parte dos.

Sesiones PGM

Identificadores globales de envío (GSI)[15]

Un identificador de envío globalmente único (GSI) es requerido por PGM para llevar el estado de los participantes de las transmisiones. La notación utilizada para representar los GSI es sobre el siguiente formato: "000.000.000.000.000.000”. Por ejemplo: 94.249.49.74.47.133

[14] NSS: Name Service Switch. Facilidad utilizada en sistemas operativos de la familia UNIX para resolver nombres

[15] *GSI: globally unique source identifier*

Identificadores de sesión de transporte (TSI)[16]

Los TSI son identificadores de transporte globalmente únicos y se componen de un GSI y un puerto de datos origen asignado por el emisor. El formato es el mismo que el del GSI mas el puerto. Por ejemplo: 94.249.49.74.47.133.60655

Sesión

Una sesión multicast confiable PGM se define a partir de una combinación de un GSI y un puerto fuente. Cuando se configuran múltiples sesiones PGM es necesario seleccionar las direcciones de grupos multicast y los puertos adecuados.

El puerto de datos destino (dport) y el grupo multicast tiene que ser el mismo para cada nodo que quiera unirse a una sesión particular, el puerto de datos origen (sport) tiene que ser único por GSI (es decir para el nodo). Por ejemplo los siguientes parámetros son todos únicos:

- Sesión 1: red “;239.192.0.1”, sport 0, dport 7500
- Sesión 2: red “;239.192.0.2”, sport 0, dport 7500
- Sesión 3: red “;239.192.0.2”, sport 0, dport 7501
- Sesión 4: red “;239.192.0.2”, sport 0, dport 7502

El filtrado multicast en la red ocurre en el nivel de la dirección de grupo multicast, el filtrado a nivel puerto ocurre al nivel del sistema operativo.

El estado de un transporte está limitado a la sesión PGM entre un emisor y uno o mas receptores. El ambiente de la sesión se define explícitamente por opciones PGM o auto configuradas en tiempo de ejecución.

Comienzo de sesión

La presencia de un emisor PGM es notada cuando se recibe el primer paquete con un nuevo TSI. Este paquete puede ser un paquete SPM (Source Path Message) que describe el árbol de distribución o un paquete de datos que indica una unión tardía (“late join”).

Fin de sesión

Cada receptor PGM mantiene una lista activa de información de sesiones. Se dispone de dos métodos para marcar sesiones como no válidas y liberar sus recursos.

- *OPT_FIN*: un paquete con la opción OPT_FIN indica que un emisor ha finalizado la sesión. Por ejemplo, esto puede indicar el final de la transmisión de un archivo
- *Vencimiento*[17]: la entrega de un paquete con la opción OPT_FIN no está garantizada. Por esto,

[16] *TSI: transport session identifier*
[17] “Peer expiry”

si la aplicación es cerrada abruptamente, los receptores que están suscriptos a un trasporte que ha sido terminado no van a recibir mas paquetes. En los dominios de aplicación donde las sesiones son temporales, estas deben ser descartadas cuando se cierra el transporte. Si una aplicación receptora no recibe la notificación de cierre de sesión se debe asumir luego de un período de tiempo en el cual no se reciben paquetes que el emisor ha finalizado.

Encapsulado sobre UDP

Cuando se utiliza encapsulado sobre UDP se debe especificar un par de puertos para la capa de transporte UDP además del conjunto de parámetros/puertos de PGM.

Si se implementa un sistema con múltiples canales PGM, puede ser conveniente fijar los mismos parámetros PGM y alterar los puertos UDP para cada sesión, por ejemplo:

- Sesión 1: red “;239.192.0.1”, sport 0, dport 7500, udp-port 3055
- Sesión 2: red “;239.192.0.2”, sport 0, dport 7500, udp-port 3056

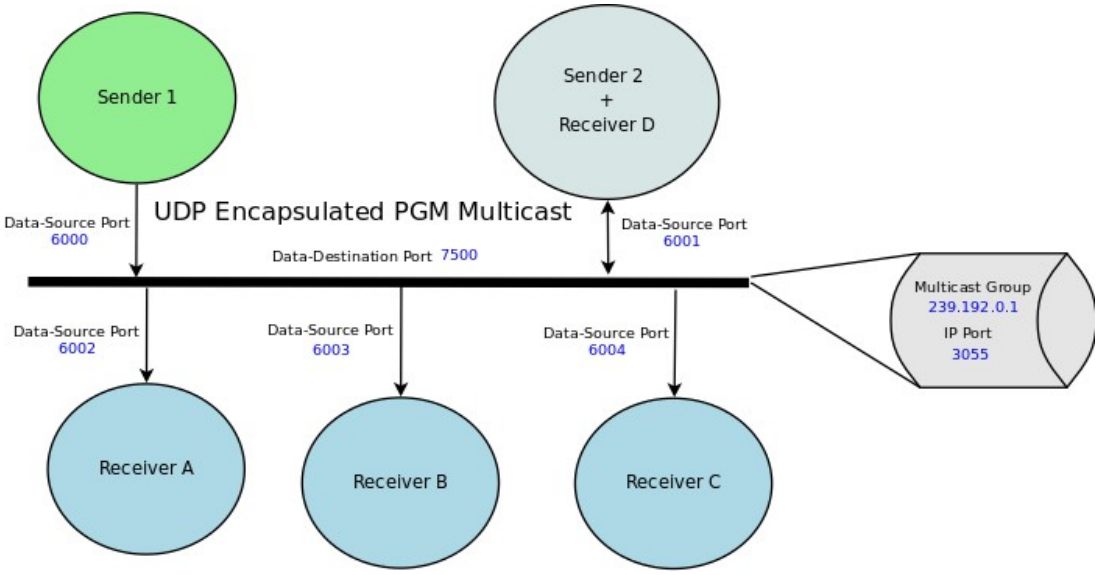


Ilustración 8: http://miru.hk/wiki/Udp_encapsulated_ports.png

Implementación de PGMCC

Las versiones actuales (5.2.x) de OpenPGM han implementado parcialmente el esquema de control de congestión “PGMCC”. De todas maneras, dicha implementación es considerada totalmente experimental, no está soportada por los autores, e incluso hay que modificar la configuración de la librería en tiempo de compilación para que la misma sea habilitada.

El esquema de control de congestión PGMCC no se ha tenido en cuenta para el presente trabajo ya que, como se verá en el capítulo siguiente, se prefiere utilizar una estrategia diferente realizando modificaciones mínimas a PGM para poder cambiar la tasa de transferencia en forma dinámica y de esta forma intentar mejorar su comportamiento.

Como vimos en la descripcion de PGM del capítulo 3, PGMCC se basa en un esquema de ventana deslizante similar al utilizado en TCP, tomando como referencia un “ACKER” del grupo de receptores. La estrategia utilizada es diferente a la que presentaremos, ya que quizás en el caso de PGMCC, los objetivos específicos de escalabilidad, estabilidad y rápida respuesta a los cambios de condiciones de la red, son mas ambiciosos que los del presente trabajo.

Interfaz de programación

La interfaz de programación de OpenPGM es similar a la de los sockets BSD.

Se crea un “socket/transporte” PGM llamando a la función `pgm_socket()` de la siguiente forma:

```
pgm_socket(pgm_socket, AF_INET, SOCK_SEQPACKET, protocol, NULL)
```

Los únicos valores válidos para `protocol` son `IPPROTO_PGM` para sockets PGM/IP y `IPPROTO_UDP` para sockets PGM/UDP.

En el caso común el protocolo PGM es bi-direccional y en preparación para recibir paquetes un receptor debe asociar (“bind”) un socket a una dirección de interfaz local usando la función `pgm_bind()`. Solo un socket PGM debe ser asociado a un par local (dirección, puerto). Cuando se especifica el parámetro `INADDR_ANY` en la llamada a `pgm_bind()`, el socket se asociará de acuerdo a las reglas locales de multicast. La función `pgm_connect()` no puede ser invocada en un socket sin asociar, el proceso debe definir previamente todos los parámetros del protocolo, no hay configuración automática.

La función pgm_bind() construye un transporte asociado a los dispositivos de red especificados. La función pgm_connect() inicia la conexión del socket PGM.

Formato de direcciones

Un punto de acceso PGM se define como una combinación de un número de puerto de datos destino de 16 bits y un identificador de sesión de transporte PGM (TSI).

```
struct pgm_sockaddr_t {
       uint16_t      sa_port;    /* data-destination port */
       pgm_gsi_t     sa_addr;
};

/* Transport Session Identifier */
struct pgm_tsi_t {
       pgm_gsi_t     gsi;
       uint16_t      sport;      /* data-source port */
};

/* Global Source Identifier */
struct pgm_gsi_t {
       char identifier[6];
};
```

El campo `sa_port` contiene el número de puerto asignado a la aplicación PGM. `sa_addr` es el TSI. El campo `gsi` de la estructura `pgm_tsi_t` es un GSI que puede ser creado con las funciones de librería `pgm_gsi_create_from_hostname(), pgm_gsi_create_from_addr(), pgm_gsi_create_from_string(), pgm_gsi_create_from_data()`. Finalmente `sport` es un número de puerto aleatorio generado por el proceso, este número de puerto debería ser único en el ámbito del emisor.

Unidades de datos

Las implementaciones de PGM deben entregar a las aplicaciones, unidades completas de datos (APDUs[18]). Esto significa que las APDUs que han sido fragmentadas en varias unidades de datos de transporte (TPDUs[19]) deben ser re-ensambladas antes de entregarlas a la aplicación.

Inicialización de la librería

Antes de poder crear, asociar y conectar los sockets PGM, es necesario inicializar la librería OpenPGM. Para esto existe una función que crea e inicializa todas las estructuras internas necesarias para que la librería pueda funcionar.

Esta función tiene un valor de retorno booleano que retorna el resultado de dicha inicialización. El prototipo de esta función es:

```
bool pgm_init* (pgm_error_t** error);
```

La primer llamada a esta función creará todo lo necesario para la operación interna de la librería de los siguientes ítems:

- soporte para mensajería
- sincronización de threads
- manejo de memoria
- generador de números pseudo-aleatorios
- soporte para temporizadores (timers)
- número de protocolo PGM

Toda llamada a `pgm_init()` debe ser correspondida por una llamada final a `pgm_shutdown()`.

Envío y recepción

La función `pgm_send()` envía una APDU por el transporte multicast representado por el socket. El transporte puede manejar paquetes de hasta el tamaño máximo de protocolo (TPDU) y cuando una aplicación provee a la función una APDU de mayor tamaño, el transporte fragmentará el mensaje en

[18] *APDU: Application Protocol Data Unit*
[19] *TPDU: Transmission Protocol Data Unit*

varios paquetes y luego los receptores lo reconstruirán y lo presentarán a la aplicación como una única unidad.

Esta función cuando termina con éxito, devuelve el valor `PGM_IO_STATUS_NORMAL` y cuando existe un error `PGM_IO_STATUS_ERROR`. Si el transporte es configurado como no bloqueante, entonces se retorna `PGM_IO_STATUS_WOULD_BLOCK` si la operación de envío causará que el transporte se bloquee y si el bloqueo será causado por el motor de limitación de tasa de transferencia, entonces se retorna el valor `PGM_IO_STATUS_RATE_LIMITED`.

Ejemplo de un envío básico no bloqueante:

```
char* s = "Hola mundo!";
size_t len = strlen (s) + 1;
int status;
do {
  status = pgm_send (sock, s, len, NULL);
} while (PGM_IO_STATUS_WOULD_BLOCK == status || PGM_IO_STATUS_RATE_LIMITED == status);
```

Las funciones `pgm_recv(), pgm_recvfrom() y pgm_recvmsg()` se utilizan para recibir datos de un transporte en OpenPGM.

El protocolo PGM es bi-direccional, incluso las aplicaciones que solo envían datos necesitan procesar requerimientos entrantes para responder a las solicitudes de retransmisión (NAKs) y a otros tipos de paquetes. Las dos primeras funciones llenan el buffer provisto en la llamada con hasta len bytes contiguos, por lo tanto dicho buffer debe ser lo suficientemente grande para alojar la APDU mas grande posible. Si el parámetro `from` no es nulo, entonces esa estructura se llena con el TSI fuente de la transmisión. La función `pgm_recv()` es idéntica a `pgm_recvfrom()` con el parámetro `from` anulado.

La pérdida no recuperable de información en la transmisión causará que la función retorne inmediatamente con el valor `PGM_IO_STATUS_RESET`. Si el parámetro `PGM_ABORT_ON_RESET` no está seteado (valor predeterminado) se puede continuar recibiendo datos con llamadas subsiguientes a la función `pgm_recv()`, si estuviera seteado, en cambio, el transporte seguirá devolviendo `PGM_IO_STATUS_RESET` hasta que sea destruído.

Las funciones pgm_recv son la entrada en el núcleo de la máquina de estados de OpenPGM. Incluso en sockets cuto modo sea solo de envío, se deben hacer llamadas frecuentes a estas funciones para procesar los paquetes NAKs entrantes, para enviar NCFs y reparaciones RDATA, además de enviar SPM para definir el árbol PGM.

Si la recepción tuvo éxito, se retorna `PGM_IO_STATUS_NORMAL`; se retorna `PGM_IO_STATUS_ERROR` cuando ocurra un error. Si el transporte es marcado como no bloqueante y no se han descubierto emisores, entonces si la operación se bloqueará se retorna

`PGM_IO_STATUS_WOULD_BLOCK`. Si no hay datos disponibles, pero hay un timer en estado pendiente ejecutando se retornará `PGM_IO_STATUS_TIMER_PENDING` y si el motor de estado está intentando enviar datos de reparación pero en el momento no puede debido al control de tasa de transferencia, este retornará `PGM_IO_STATUS_RATE_LIMITED`.

Ejemplo de recibir una APDU con datos del TSI fuente:

```
char buf[4096];
struct pgm_sockaddr_t from;
size_t bytes_read;
pgm_recvfrom (transport, buf, sizeof(buf), 0, &from, &bytes_read, NULL);
printf ("se recibieron %zu bytes desde %s\n", pgm_tsi_print (&from.sa_addr));
```

Temporizadores y selección de fuente de reloj

Debido a diferencias en los distintos tipos de hardware sobre los que se puede ejecutar la librería OpenPGM y, en particular en LINUX, las diferentes implementaciones de temporizadores (“timers”) que posee cada variante de kernel, es posible seleccionar como la librería obtendrá del sistema operativo su fuente de reloj (“clock source”).

La fuente de reloj es muy importante para la librería OpenPGM al ser utilizada en aplicaciones que necesitan alta resolución de los temporizadores para obtener mensajería PGM de alta velocidad.

Un sistema operativo calcula la fecha y hora actual en base a varias fuentes para así proveer a las aplicaciones con un origen de tiempo preciso, estable y monotónico. Históricamente, calcular la fecha y hora del sistema de forma estable no ha sido una tarea asequible y las aplicaciones que buscan alta performance, típicamente, intentan minimizar los llamados a la función gettimeofday().

Funciones estándar en orden de resolución de los relojes:

- `gettimeofday(tv)`: devuelve el número de segundos y microsegundos desde el valor de Epoch[20] en una estructura de tipo “timeval”
- `ftime(tp)`: retorna el número de segundos y milisegundos desde el valor de Epoch en una estructura de tipo “timeb”.
- `clock_gettime(clk_id, tp)`: retorna el reloj con una resolución de nanosegundos en una estructura de tipo “timespec”

Actualmente, la resolución de los temporizadores en el kernel de LINUX se encuentra en constante modificación; la mayoría de los kernels tienen resoluciones estándar de 1-4 ms, mientras que los temporizadores de alta velocidad pueden proveer resoluciones cuyas unidades son los microsegundos o hasta incluso los nanosegundos.

[20] Epoch: 00:00:00 UTC, January 1, 1970

Los procesadores x86 (desde el Pentium) cuentan con un registro de 64 bits llamado TSC[21] en el que se almacena el número de ciclos desde el último “reset”. La instrucción RDTSC, que permite obtener el valor de dicho registro, es una forma sencilla y poco costosa de obtener el reloj actual con alta resolución, sin embargo, los sistemas de múltiples núcleos pueden sufrir de deriva (“drift”) entre los distintos núcleos; particularmente los procesadores Intel con “hyperthreading” son conocidos por sus falencias en este sentido, por lo que se recomienda deshabilitar esta característica en los sistemas en los que se quiera ejecutar mensajería PGM de alta velocidad. Una alternativa en éstos sistemas es utilizar afinidad de procesador para las llamadas a RDTSC y así garantizar que la fuente de reloj es siempre el mismo procesador. Otra opción es utilizar un concepto más moderno conocido como HPET.

El temporizador de eventos de alta precisión (HPET[22]) fue creado para proveer un temporizador estable de alta resolución global a todo el sistema en equipos con procesadores multi-núcleo y asi resolver las irregularidades de los registros TSC. Las lecturas del dispositivo HPET usualmente tienen un costo de unos ~500 nanosegundos.

En LINUX, otra posible fuente de reloj es el dispositivo RTC (/dev/rtc) que ofrece una fuente de reloj estable y externa al registro TSC, pero que no puede ser compartido entre múltiples aplicaciones. Este dispositivo tiene la ventaja de estar disponible en, prácticamente, todas las plataformas, a diferencia de la limitada disponibilidad de HPET.

En la práctica la selección de la fuente de reloj se hace preseteando la variable de entorno PGM_TIMER. Los posibles valores de esta variable son:

Valor PGM_TIMER	Descripción
CLOCK_GETTIME	Utilizar función clock_gettime()
FTIME	Utilizar función ftime(tp)
RTC	Leer reloj desde /dev/rtc. Timer de 8192 Hz que provee resolución de 122us
TSC	Utilizar llamadas a RDTSC y normalizar usando una cantidad calibrada de “core ticks”
HPET	Leer reloj desde el dispositivo HPET
GETTIMEOFDAY	Utilizar llamadas a gettimeofday(tv, NULL). Valor predeterminado en LINUX
MMTIME	Utilizar temporizadores de Windows multimedia, baja resolución pero estables
QPC	Utilizar WPC (Windows Performance Counters) dependientes de drivers y soporte de BIOS

Tabla 3: Posibles valores para la variable PGM_TIMER

[21] TSC: Time Stamp Counter
[22] HPET: High Precision Event Timer

Capítulo 6: Diseño de la solución

Estrategia de envío con control de congestión

Debido a que este protocolo será utilizado en múltiples escenarios que poseen distintas características de equipamiento de red, y servidores (emisores) y cajas registradoras (receptores) con caracterísricas totalmente heterogéneas, es necesario que el mismo pueda adaptarse a cada red regulando la tasa de transmisión para no sobrecargar a los receptores, evitando así que se produzcan demasiados NAKs y se deba realizar un alto número de retransmisiones.

Para poder cumplir con este objetivo, la idea es utilizar una estrategia para evitar la congestión (congestion avoidance). Se utilizará en la solución un esquema basado en el algoritmo de incremento aditivo y reducción multiplicativa (AIMD)[23]. Esta estrategia tiene un algoritmo de control con retro-alimentación para evitar la congestión combinando el crecimiento lineal de la ventana de congestión con una reducción exponencial de la tasa de transferencia cuando ésta es detectada.

La estrategia de este algoritmo es incrementar la tasa de transmisión hasta que ocurra pérdida de paquetes. En el caso de PGM, se utilizarán los NAKs como indicadores de que al menos un receptor detectó una pérdida. Cuando se detecta la congestión, el emisor reducirá la tasa de transferencia por un factor multiplicativo; por ejemplo dividir a la mitad la ventana de transmisión. El resultado de esto es un gráfico con forma de “dientes de sierra” ya que este algoritmo va “probando” la red para detectar el ancho de banda disponible, de manera similar a lo que ocurre con el algoritmo para evitar congestión en TCP. El modelo de este comportamiento se puede observar en la ilustración número 9.

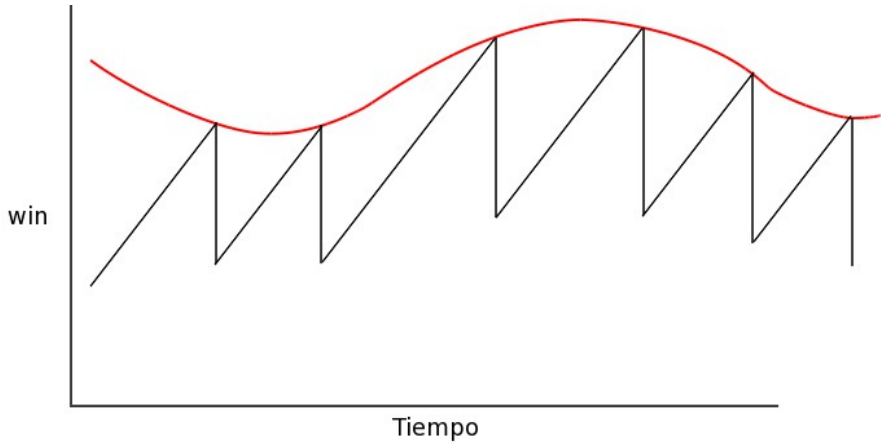


Ilustración 9: Curva de utilización de capacidad de red (curva roja)

[23] AIMD: Additive increase/multiplicative decrease

Fórmula matemática AIMD

Sea w(t) la tasa de transferencia (ventana de congestión) durante el intervalo de tiempo t; sea a (a > 0) el factor de incremento aditivo y sea b (0 < b < 1) el factor de decremento multiplicativo:

$$w(t+1)=\begin{cases} w(t)+a & si\ no\ se\ detecta\ congestión \\ w(t)*b & si\ se\ detecta\ congestión \end{cases}$$

Diseño de la función de envío

Cada sesión de envío del emisor trabajará enviando datos a los receptores via paquetes de datos (ODATA) con PGM, con una ventana de transmisión c*wnd*, que le permitirá enviar sin restricciones hasta agotada dicha ventana. Luego del envío de cada bloque, el emisor deberá chequear cual es la tasa actual de recepción de paquetes NAK. La librería OpenPGM lleva un registro interno de la cantidad de NAKs recibidos para cada objeto pgm_sock desde que este es creado y consultando este dato podremos conocer cual es la cantidad de paquetes NAK recibidos.
Si la tasa de NAKs es menor al umbral *nak_th* durante toda la duración de la ventana *cwnd*, entonces se ampliará la ventana de transmisión utilizando el factor lineal aditivo (a). Por el contrario, si se detecta una tasa de NAKs mayor al umbral *nak_th* en ese período, reduciremos la ventana de transmisión multiplicando la misma por el factor de decremento (b) y el emisor entrará en un estado de control de congestión (CC).

En el estado de control de congestión, el emisor podrá seguir enviando datos, pero sin incrementar la ventana de transmisión *cwnd* y deberá controlar luego de la transmisión de cada paquete la tasa actual de NAKs; es decir que dentro del estado de control de congestión el límite c*wnd* no se utilizará; el emisor, sin embargo, utilizará una ventana de control de congestión *cc_cwnd*. Esta ventana se usa para "limpiar" el estado de congestión ya que el emisor deberá controlar que no haya recibido ningún paquete NAK durante esta ventana para poder salir del estado de control de congestión. Una vez que el emisor no haya recibido ningún NAK durante una ventana del tamaño definido para el control de congestión se podrá volver al estado normal de envío.

Esta estrategia tiene la ventaja de no requerir ninguna modificación en los receptores. Es decir, que los receptores solo implementarán la función normal de recepción con la notificación de los NAKs respectivos a los paquetes faltantes que hayan detectado como cualquier receptor normal PGM. Este diseño también tiene la ventaja de que no es necesario modificar la lógica interna del protocolo PGM o sus funciones de bajo nivel. El esquema de control de congestión se puede construir, por ejemplo, sobre la implementación OpenPGM del protocolo PGM, utilizando sus primitivas. Solo será necesario, como se verá en el capítulo de implementación, una mínima modificación a la librería OpenPGM para poder consultar el contador de número de NAKs recibidos ya que este contador es de uso interno de la misma y no existe ninguna primitiva en su API[24] para poder leer el valor del mismo.

[24] API: application programming interface

Parámetros configurables

Los siguientes parámetros que controlarán la transmisión serán configurables por parte del usuario/desarrollador que utilice las funciones del protocolo:

- *cwnd*: tamaño inicial de ventana de transmisión
- *cc_cwnd*: ventana de transmisión en la que no se deben recibir NAKs para salir de estado CC
- *nak_th*: umbral límite de la tasa de NAKs
- *a*: factor de incremento lineal ($a > 0$)
- *b*: factor de decremento exponencial ($0 < b < 1$)
- *r*: tasa de transferencia inicial de transmisión (valor predeterminado: 100000bps)

Algoritmo de envío

A continuación se presenta, en lenguaje de pseudo-código, el algoritmo que se utilizará en la solución:

```
 1:  mientras existan datos para enviar
 2:       pgm_send(sock, buffer)
 3:       si estado==CC
 4:            si nak_rate < nak_threshold
 5:                 si se completó cc_cwnd
 6:                      estado=NORMAL
 7:                 sino
 8:                      avanzar ventana cc_cwnd
 9:            sino
10:                 reiniciar ventana cc_cwnd
11:       sino
12:            si nak_rate > nak_threshold
13:                 estado=CC
14:                 cwnd = cwnd * b
15:                 cc_cwnd = cc_cwnd * b
16:                 continuar
17:       si se completó cwnd
18:            si estado==NORMAL y nak_rate < nak_threshold
19:                 cwnd = cwnd + a
20:                 cc_cwnd = cc_cwnd + a
21:       sino
22:            avanzar ventana cwnd
23:  fin bucle
```

De esta manera, el emisor podrá ir incrementando la tasa de transferencia linealmente, cada vez que complete una ventana *cwnd* sin que la tasa de NAKs haya superado el umbral. Cuando la tasa de NAKs sea mayor al umbral configurable, deberá reducir la tasa de transferencia y luego entrará en la fase de

control de congestión. Cuando haya pasado una ventana *cc_cwnd* completa donde la tasa de NAKs sea menor a la mitad del umbral definido, entonces podrá salir del estado de control de congestión y volver a trabajar con la ventana *cwnd*. Este proceso se repetirá sucesivamente y la tasa de transferencia debería converger al ancho de banda disponible en la red para la transmisión multicast.

Diseño de la librería compartida

Para encapsular la funcionalidad desarrollada en este proyecto, se diseñará e implementará una librería compartida en lenguaje C++ llamada “pgmcoop”, que podrá utilizarse desde cualquier programa/sistema como librería en formato estático o dinámico (shared).

Esta librería tendrá una API muy simple, formada por una estructura de datos y varias funciones. La estructura de datos encapsulará los parámetros que el usuario de la librería deberá seleccionar para configurar las transmisiones y recepciones PGM (`pgmcoop_options_t`), como por ejemplo: max_tpdu, max_rte, port, loglevel, etc.

Las funciones mas importantes de la librería serán:

- `pgmcoop_send(filename, options)`: envía el archivo referenciado por el nombre “`filename`” con los parámetros y opciones especificados en la estructura “`options`”.
- `pgmcoop_recv(filename, options)`: recibe un archivo y lo almacena con el nombre “`filename`” a partir de los parámetros y opciones especificados en la estructura “`options`”.

Ambas funciones retornarán un código de estado (entero) que indicará el éxito (0) o un fallo en la ejecución de la función con un número mayor que 0. En el caso de indicar un error, se llenará la variable pública `pgmcoop_error` con el mensaje de error descriptivo correspondiente para informar al usuario el detalle del fallo.

En este esquema, la aplicación utilizará la API de la librería “pgmcoop”, que luego internamente utilizará la librería OpenPGM, con su API levemente modificada, como se verá en la siguiente sección. La librería OpenPGM utiliza como primitivas las llamadas al sistema para utilizar los sockets IP de la implementación IP multicast del kernel. Este diseño se ve resumido en la ilustración 10.

Para mantener la librería lo mas simple posible, la misma se diseña con algunas limitaciones ya que la aplicación solo podrá mantener activa una sesión, sea esta de transmisión o recepción y, en principio, la implementación de la librería no será “thread-safe”, por lo que las estructuras internas no se podrán compartir desde distintos hilos de ejecución.

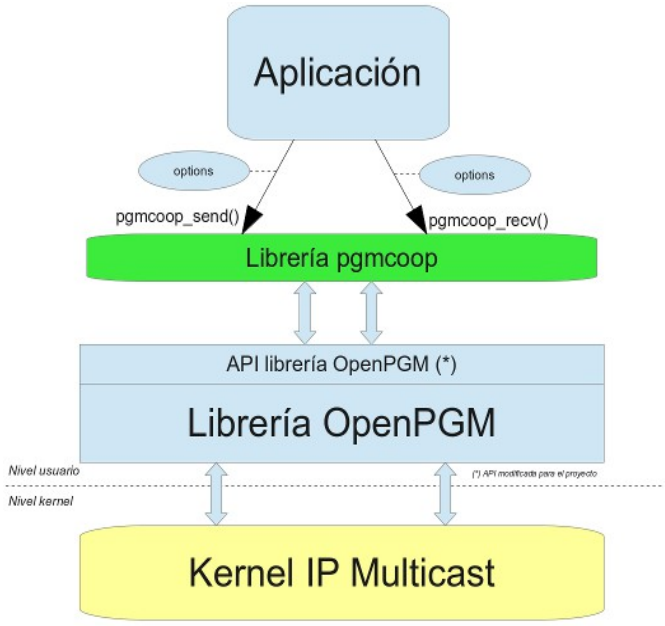


Ilustración 10: Diseño de librería pgmcoop

Modificaciones necesarias a la librería OpenPGM

- Consulta de contadores de estadísticas internas: la librería OpenPGM internamente lleva contadores para una gran cantidad de estadísticas. De todas maneras, estos contadores en la librería original no pueden ser consultados desde la aplicación que usa la misma. Se modificó la API de la librería para poder consultar estos contadores, en especial el valor de los NAKs recibidos en el emisor para servir de dato de entrada del algoritmo de control de congestión. Las funciones que se implementaron para obtener las estadísticas internas son 2 y se creó un nuevo archivo fuente (stats.c) para encapsular las mismas:
  - `pgm_dump_sender_stats`: para mostrar como salida en un descriptor de archivo las estadísticas de envío de un transmisor (paquetes enviados, bytes enviados, NAKs recibidos, paquetes descartados, paquetes y bytes retransmitidos, etc.)
  - `pgm_dump_receiver_stats`: para mostrar como salida en un descriptor de archivo las estadísticas de un receptor (paquetes y bytes recibidos, cantidad de paquetes ODATA, RDATA y NAKs, paquetes duplicados, etc.)

```
void pgm_dump_sender_stats(pgm_sock_t* const restrict sock, FILE *fd)
{
      const pgm_txw_t* window = sock->window;
      fprintf (fd, "Performance information:\n"
           "Data bytes sent %u\n"
           "Data packets sent %u\n"
           "Bytes buffered %u\n"
           "Packets buffered %u\n"
```

```
           "Bytes sent %u\n"
           "Raw NAKs received %u\n"
           "Checksum errors %u\n"
           "Malformed NAKs %u\n"
           "Packets discarded %u\n"
           "Bytes retransmitted %u\n"
           "Packets retransmitted %u\n"
           "NAKs received %u\n"
           "NAKs ignored %u\n"
           "Transmission rate %u bps\n"
           "NNAK packets received %u\n"
           "NNAKs received %u\n"
           "Malformed NNAKs %u\n",
           sock->cumulative_stats[PGM_PC_SOURCE_DATA_BYTES_SENT],
           sock->cumulative_stats[PGM_PC_SOURCE_DATA_MSGS_SENT],
           window ? (uint32_t)pgm_txw_size (window) : 0,   /* minus IP & any UDP header */
           window ? (uint32_t)pgm_txw_length (window) : 0,
           sock->cumulative_stats[PGM_PC_SOURCE_BYTES_SENT],
           sock->cumulative_stats[PGM_PC_SOURCE_SELECTIVE_NAKS_RECEIVED],
           sock->cumulative_stats[PGM_PC_SOURCE_CKSUM_ERRORS],
           sock->cumulative_stats[PGM_PC_SOURCE_MALFORMED_NAKS],
           sock->cumulative_stats[PGM_PC_SOURCE_PACKETS_DISCARDED],
           sock->cumulative_stats[PGM_PC_SOURCE_SELECTIVE_BYTES_RETRANSMITTED],
           sock->cumulative_stats[PGM_PC_SOURCE_SELECTIVE_MSGS_RETRANSMITTED],
           sock->cumulative_stats[PGM_PC_SOURCE_SELECTIVE_NAKS_RECEIVED],
           sock->cumulative_stats[PGM_PC_SOURCE_SELECTIVE_NAKS_IGNORED],
           sock->cumulative_stats[PGM_PC_SOURCE_TRANSMISSION_CURRENT_RATE],
           sock->cumulative_stats[PGM_PC_SOURCE_SELECTIVE_NNAK_PACKETS_RECEIVED],
           sock->cumulative_stats[PGM_PC_SOURCE_SELECTIVE_NNAKS_RECEIVED],
           sock->cumulative_stats[PGM_PC_SOURCE_NNAK_ERRORS]);
}

void pgm_dump_receiver_stats(pgm_sock_t* const restrict sock, FILE *fd)
{
    if (!sock->peers_list) {
        return;
    }

    pgm_rwlock_reader_lock (&sock->peers_lock);
    pgm_list_t* peers_list = sock->peers_list;
    while (peers_list) {
        pgm_list_t* next = peers_list->next;
        pgm_peer_t* peer = peers_list->data;
        const pgm_rxw_t* window = peer->window;
        const uint32_t outstanding_naks = window->nak_backoff_queue.length +
                                          window ->wait_ncf_queue.length +
                                          window->wait_data_queue.length;
        fprintf (fd, "Performance information:\n"
             "Data bytes received %u\n"
             "Data packets received %u\n"
             "NAK failures %u\n"
             "Bytes received %u\n"
             "Checksum errors %u\n"
             "Malformed SPMs %u\n"
             "Malformed ODATA %u\n"
             "Malformed RDATA %u\n"
             "Malformed NCFs %u\n"
             "Packets discarded %u\n"
             "Losses %u\n"      /* detected missed packets */
             "Bytes delivered to app %u\n"
             "Packets delivered to app %u\n"
             "Duplicate SPMs %u\n"
             "Duplicate ODATA/RDATA %u\n"
             "NAK packets sent %u\n"
             "NAKs sent %u\n"
```

```
            "NAKs retransmitted %u\n"
            "NAKs failed %u\n"
            "NAKs failed due to RXW advance %u\n"
            "NAKs failed due to NCF retries %u\n"
            "NAKs failed due to DATA retries %u\n"
            "NAK failures delivered to app %u\n"
            "NAKs suppressed %u\n"
            "Malformed NAKs %u\n"
            "Outstanding NAKs %u\n"
            "NAK repair min time %u µs\n"
            "NAK repair mean time %u µs\n"
            "NAK repair max time %u µs\n"
            "NAK fail min time %u µs\n"
            "NAK fail mean time %u µs\n"
            "NAK fail max time %u µs\n"
            "NAK min retransmit count %u\n"
            "NAK mean retransmit count %u\n"
            "NAK max retransmit count %u\n",
            peer->cumulative_stats[PGM_PC_RECEIVER_DATA_BYTES_RECEIVED],
            peer->cumulative_stats[PGM_PC_RECEIVER_DATA_MSGS_RECEIVED],
            peer->cumulative_stats[PGM_PC_RECEIVER_NAK_FAILURES],
            peer->cumulative_stats[PGM_PC_RECEIVER_BYTES_RECEIVED],
            sock->cumulative_stats[PGM_PC_SOURCE_CKSUM_ERRORS],
            peer->cumulative_stats[PGM_PC_RECEIVER_MALFORMED_SPMS],
            peer->cumulative_stats[PGM_PC_RECEIVER_MALFORMED_ODATA],
            peer->cumulative_stats[PGM_PC_RECEIVER_MALFORMED_RDATA],
            peer->cumulative_stats[PGM_PC_RECEIVER_MALFORMED_NCFS],
            peer->cumulative_stats[PGM_PC_RECEIVER_PACKETS_DISCARDED],
            window->cumulative_losses,
            window->bytes_delivered,
            window->msgs_delivered,
            peer->cumulative_stats[PGM_PC_RECEIVER_DUP_SPMS],
            peer->cumulative_stats[PGM_PC_RECEIVER_DUP_DATAS],
            peer->cumulative_stats[PGM_PC_RECEIVER_SELECTIVE_NAK_PACKETS_SENT],
            peer->cumulative_stats[PGM_PC_RECEIVER_SELECTIVE_NAKS_SENT],
            peer->cumulative_stats[PGM_PC_RECEIVER_SELECTIVE_NAKS_RETRANSMITTED],
            peer->cumulative_stats[PGM_PC_RECEIVER_SELECTIVE_NAKS_FAILED],
            peer->cumulative_stats[PGM_PC_RECEIVER_NAKS_FAILED_RXW_ADVANCED],
            peer->cumulative_stats[PGM_PC_RECEIVER_NAKS_FAILED_NCF_RETRIES_EXCEEDED],
            peer->cumulative_stats[PGM_PC_RECEIVER_NAKS_FAILED_DATA_RETRIES_EXCEEDED],
            peer->cumulative_stats[PGM_PC_RECEIVER_NAK_FAILURES_DELIVERED],
            peer->cumulative_stats[PGM_PC_RECEIVER_SELECTIVE_NAKS_SUPPRESSED],
            peer->cumulative_stats[PGM_PC_RECEIVER_NAK_ERRORS],
            outstanding_naks,
            window->min_fill_time,
            peer->cumulative_stats[PGM_PC_RECEIVER_NAK_SVC_TIME_MEAN],
            window->max_fill_time,
            peer->min_fail_time,
            peer->cumulative_stats[PGM_PC_RECEIVER_NAK_FAIL_TIME_MEAN],
            peer->max_fail_time,
            window->min_nak_transmit_count,
            peer->cumulative_stats[PGM_PC_RECEIVER_TRANSMIT_MEAN],
            window->max_nak_transmit_count);
            peers_list = next;
    }

    pgm_rwlock_reader_unlock (&sock->peers_lock);
}
```

- Modificación de tasa máxima de transferencia: cada sesión en la librería OpenPGM lleva un valor de máxima tasa de transferencia (max_rte) que regula en el emisor el throughput del envío

de datos a través del socket PGM. Este valor en la librería original no puede ser modificado durante una emisión en curso. Para poder implementar el algoritmo de control de congestión propuesto, es necesario poder modificar este valor una o mas veces durante una emisión PGM. Por lo tanto, se modificará la librería agregando una función de API para poder modificar el valor de max_rte y reiniciar el algoritmo interno de token-bucket que utiliza la librería OpenPGM.
Para implementar esto, se agregó la función pgm_rate_modify en el archivo fuente socket.c. Los argumentos deben ser expresados en bytes por segundo.

```
bool pgm_rate_modify (pgm_sock_t*const sock, const unsigned max_rte, const unsigned
                      odata_max_rte, const unsigned rdata_max_rte)
{
    if (max_rte > 0) {
        pgm_rate_destroy(&sock->rate_control);
        sock->txw_max_rte = max_rte;
        pgm_trace(PGM_LOG_ROLE_RATE_CONTROL,_("Setting TX rate to %" PRIzd " bytes per second."), max_rte);
        pgm_rate_create (&sock->rate_control, sock->txw_max_rte, sock->iphdr_len, sock->max_tpdu);
      }
    if (odata_max_rte > 0) {
        pgm_rate_destroy(&sock->odata_rate_control);
        sock->odata_max_rte = odata_max_rte;
        pgm_trace(PGM_LOG_ROLE_RATE_CONTROL,_("Setting ODATA rate to %" PRIzd " bytes per second."),
                  odata_max_rte);
        pgm_rate_create (&sock->odata_rate_control, sock->odata_max_rte, sock->iphdr_len, sock->max_tpdu);
        sock->is_controlled_odata = TRUE;
    }
    if (rdata_max_rte > 0) {
        pgm_rate_destroy(&sock->rdata_rate_control);
        sock->rdata_max_rte = rdata_max_rte;
        pgm_trace(PGM_LOG_ROLE_RATE_CONTROL,_("Setting RDATA rate to %" PRIzd " bytes per second."),
                  rdata_max_rte);
        pgm_rate_create (&sock->rdata_rate_control, sock->rdata_max_rte, sock->iphdr_len, sock->max_tpdu);
        sock->is_controlled_rdata = TRUE;
    }
}
```

Capítulo 7: Implementación

Estructura de la librería

El código fuente de la librería se compone de un archivo de encabezado (“header”) de C++ que contiene las definiciones de tipos y estructuras de datos, constantes y variables que serán utilizados en la misma; además un conjunto de archivos fuente (“source”) que contienen la definición de las funciones principales (`pgmcoop_send` y `pgmcoop_recv`) y las funciones auxiliares que son necesarias para su implementación.

```
pgmcoop
|-- include
|   `-- pgmcoop.hh
`-- src
    |-- Makefile
    |-- init.cpp
    |-- nakthread.cpp
    |-- recv.cpp
    |-- send.cpp
    |-- signal.cpp
    |-- sock.cpp
    `-- static.cpp

2 directories, 9 files
```

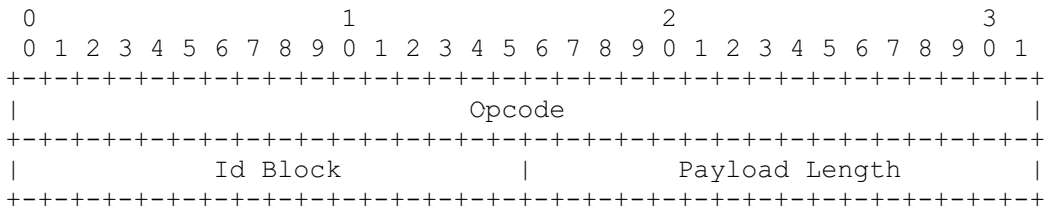
El código fuente contiene un archivo “Makefile” para facilitar la compilación de la librería. Hay dos formas de compilar la misma:

1. Librería estática (.a): es la manera predeterminada y se realiza ejecutando “make” desde el directorio de trabajo “src”. Este modo generará un archivo libpgmcoop.a que luego deberá ser vinculado en forma estática con el resto de los archivos objeto (.o) al momento de generar los binarios ejecutables.
2. Librería compartida (.so): de esta manera se puede generar una librería compartida que no es necesario incluir en el binario ejecutable. La librería generada libpgmcoop.so deberá estar instalada en el directorio de librerías del sistema al momento de querer ejecutar cualquier binario que haya sido vinculado dinámicamente contra la misma. Este modo se invoca ejecutando “make shared_lib” desde el directorio de trabajo “src”.

De manera adicional se define una clase para encapsular algunas constantes y funciones que se utilizarán para definir un protocolo muy simple, llamado “MDP”, que determina la estructura de los datos que se enviarán en la secuencia de paquetes multicast y darán forma a la transmisión de archivos. Este protocolo define un encabezado en los paquetes donde se empaqueta un código de operación (“opcode”) para comunicar a los destinatarios el tipo de paquete, un código de bloque (“id_block”) para

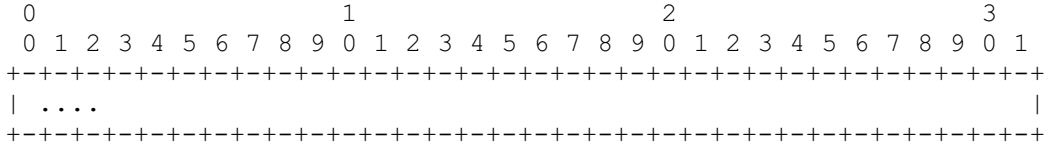
los paquetes de datos y una longitud de paquete (“payload length”) de 16 bits; a continuación sigue el campo de datos (“payload”) de longitud máxima “MAX_DATA_SIZE” configurable en tiempo de compilación.

MDP v1 HEADER



- Opcode (32 bits): Identificador de tipo de paquete
- Id Block (16 bits): Identificador de bloque de datos (en opcode 0 indica la cantidad de bloques)
- Payload Length (16 bits): Tamaño del campo de datos

MDP v1 DATA PAYLOAD



- Longitud indicada por el campo Payload Length del header.

Descripción del código fuente

El código fuente se organizó en varios archivos (.cpp) que contienen las funciones que componen la librería y se agruparon de acuerdo a características comunes de las mismas. A continuación se brinda una breve descripción de cada uno de los archivos y sus funciones. Las funciones cuyo nombre comienza con “_” son consideradas internas y auxiliares a la implementación de la librería y no forman parte de la interfaz (API) de la misma.

`init.cpp`

`pgmcoop_init_options(void)`: crea una estructura de datos pgmcoop_options_t, y la inicializa con los valores predeterminados de las opciones, por ejemplo: max_tpdu 1500, max_rte 100000, etc. Devuelve el puntero (dirección de memoria) a la misma.

`nakthread.cpp`

`_pgmc_create_nak_thread(pgm_sock_t *sock)`: función interna a la librería para crear un thread que deberá procesar los paquetes NAK. Este thread es necesario de acuerdo al modo de funcionamiento de la librería OpenPGM.

_pgmc_nak_routine(void *arg): método “run” del thread de procesamiento de NAKs. Interno a la implementación de la librería.

recv.cpp

`pgmcoop_recv(const char* file_name, pgmcoop_options_t* options)`:

Implementación de la función para recibir un archivo via transmisión multicast PGM. El destino será un archivo que se creará con el nombre especificado en el parámetro file_name. La lógica de la función es primero inicializar el motor de la librería OpenPGM invocando la función pgm_init, luego crear un buffer para la recepción de datos y un socket PGM en la variable pgmc_sock invocando a la función interna _pgmc_create_recv_sock a la cual se le pasa un objeto options con las opciones que se van a utilizar en la sesión.

Luego, la función ingresa en un bucle para recibir los datos. En el bucle se realizan sucesivas llamadas a la función pgm_recvfrom. La recepción de datos es no bloqueante y el resultado de la invocación a esta función puede tener como resultado: 1. una recepción normal, 2. la indicación de que no se pueden recibir datos debido a que se debe esperar a que un timer expire o a cumplir con una limitación impuesta en la tasa de transferencia o a que simplemente todavía no se recibieron datos. En el primer caso, se procede a analizar el contenido del buffer y se interpreta el paquete con el protocolo MDP para ver si se trata de un paquete de comienzo de archivo, paquete de datos o fin de archivo. En el segundo caso es necesario utilizar alguno de los mecanismos de espera no bloqueante que posea la plataforma sobre la que se compile el programa. Estos mecanismos se describen en una sección posterior de este mismo capitulo.

Al finalizar el bucle, se cierran los descriptores de archivo que se utilizaron, opcionalmente se puede hacer un dump de estadísticas a salida de errores, se cierra el socket pgmc_sock, se invoca la función de shutdown de la librería OpenPGM y se elimina el buffer utilizado para la recepción de datos. La función retorna el valor 0 si la transmisión del archivo fue exitosa o -1 si no lo fue.

send.cpp

`pgmcoop_send(const char* file_name, pgmcoop_options_t* options)`:

Implementación de la función para enviar el archivo especificado en el parámetro file_name via transmisión multicast PGM. Como parte de la inicialización, se invoca la función pgm_init de la librería OpenPGM, se crea el socket que se utilizará para enviar, se crea el thread que procesará los frames NAK, se reserva el espacio de memoria para el buffer de envío y se abre para lectura el archivo que se enviará.

El comienzo de la transmisión se marca con el envío de un paquete MDP con opcode 0 (start of file). Luego se ingresa en un bucle para transmitir todos los bloques de datos del archivo en paquetes MDP con opcode 1 (data packet) utilizando la función pgm_send().

De forma posterior al envío, se chequean las condiciones del mecanismo de control de congestión implementado. Esto se describe en una sección posterior de este capitulo.

Para finalizar, al salir del bucle de envío, se transmite un paquete MDP para marcar el fin de archivo; el opcode es 2 (end of file packet).

signal.cpp

`_pgmc_on_signal (int signum)`: se define una función auxiliar para manejo de las señales. Las funciones de envio y recepcion definen la captura de las señales SIGINT y SIGTERM para invocar esta función, que luego notifica a las funciones que deberán finalizar a través de la variable global de la librería _pgmc_is_terminated.

sock.cpp

`_pgmc_create_send_sock (pgmcoop_options_t *options)`: función para crear un socket pgm_sock_t e inicializarlo con las opciones especificadas en la estructura options. Finaliza con la invocación de la función pgm_connect de la API de OpenPGM. Esta función esta orientada a sockets para envío de datos.

`_pgmc_create_recv_sock (pgmcoop_options_t *options)`: función análoga a la anterior pero orientada a sockets para recepción de datos.

static.cpp

Declaración de variables estáticas internas y globales a la librería pgmcoop.

Mecanismos de recepción no bloqueante

En plataformas GNU/Linux se disponen de tres mecanismos de recepción de datos en esquemas no bloqueantes. Cual de ellos encontramos, es dependiente de la versión de librería base del sistema (glibc) y/o la versión de kernel que tiene nuestra plataforma. Los mecanismos son:

- select: mecanismo tradicional de llamadas a sistema en UNIX para hacer entrada/salida asincrónica. Su interfaz de programación es complicada y la implementación de la misma suele ser mediocre. Impone un límite en la cantidad máxima de descriptores de archivo que un proceso puede monitorear lo cual puede ser un inconveniente para las aplicaciones. Por otro lado, la performance de esta llamada a sistema se degrada linealmente al aumentar el número de descriptores. El timeout se especifica con una estructura timeval que permite resolución de microsegundos.
- poll: esta llamada a sistema realiza una tarea similar a select, espera hasta que uno de los descriptores de archivo de un conjunto se encuentre listo para realizar I/O. Su interfaz de programación permite especificar tipos de eventos por los cuales se desea esperar. Además el timeout se especifica en cantidad de milisegundos. Sin embargo en velocidad se considera que ambas llamadas al sistema se desempeñan de la misma forma.
- epoll: se considera un avance enorme con respecto a las otras dos alternativas en términos de interfaz de programación y de performance, pero solo está disponible en versiones 2.6.x del kernel de Linux. Está diseñado para escalar mejor a grandes números de descriptores de archivo/sockets.

Implementación del control de congestión

La rutina de transmisión va recorriendo el archivo original en un ciclo y envía los bloques (de tamaño configurable en tiempo de ejecución) utilizando la primitiva pgm_send de la librería OpenPGM.

Lo primero que se hace en esta implementación, luego de enviar un bloque, es chequear si nos encontramos en modo control de congestión (variable booleana cc) por haber detectado, previamente, una alta tasa de NAKs. Luego para manejar la ventana de control de congestión se lleva un contador cc_wnd y un límite de ventana cc_cwnd. Cuando recorrimos toda la ventana de control de congestión teniendo una tasa de NAKs siempre menor a la mital del umbral nak_th, entonces la implementación vuelve al modo normal, reseteando el contador cc_wnd. Si durante el modo de control de congestión se sigue detectando una tasa de NAKs superior al umbral, entonces la tasa de transmisión se reduce por el factor b.

El mecanismo de control de congestión implementado se basa en chequear la cantidad de tramas NAK generadas por los receptores y en base a esto ajustar la tasa de transferencia. Para esto se utiliza una función adicional que agregamos como parte de este proyecto a la librería OpenPGM para consultar los contadores internos de la misma. (Ver capítulo 5).

En este chequeo se calcula la tasa de NAKs como un porcentaje de los NAKs recibidos contra los paquetes envíados. Si la tasa actual de NAKs (nak_rate) es mayor al umbral (nak_th), entonces se ingresa en modo control de congestión y se decrementa la tasa de transferencia en base al factor b. Para esto se utiliza la otra función adicional que agregamos a la librería OpenPGM para modificar en tiempo de ejecución la tasa de transferencia del socket pgm.

Para realizar el incremento de la tasa de transferencia se utiliza la ventana wnd con límite de ventana cwnd (configurable), para controlar luego de cuantos bloques transmitidos - durante los cuales la tasa de NAKs no supera nak_th - se realiza el incremento de la tasa de transferencia. Este incremento también se hace utilizando la misma función de modificación de la tasa de transferencia que para el decremento, pero utilizando el factor a.

Implementación de utilidades para envío y recepción

Como parte de la implementación, testeo y optimización de la librería realizada en este trabajo, se realizaron dos utilidades para poder testear el comportamiento de la misma. Estas utilidades de línea de comandos para sistemas GNU/Linux son programas escritos en C++ y poseen una interfaz simple para que el usuario pueda enviar y recibir archivos usando la librería pgmcoop.

Las utilidades se llaman filerecv y filesend y a continuación describimos su implementación y uso.

Utilidad filesend

Esta utilidad utiliza la función de API pgmcoop_send de la librería pgmcoop para realizar el envío de un archivo por protocolo multicast PGM. Soporta una serie de opciones de línea de comandos para controlar los parámetros del envío. Al finalizar el envío, requiere una confirmación interactiva para terminar la aplicación ya que en segundo plano queda ejecutando el thread de procesamiento de NAKs, que, por supuesto, pueden llegar en forma posterior a la finalización del envío. Queda a consideración del usuario cuanto tiempo esperar para terminar la aplicación. Para terminar por completo la ejecución, se invoca la función de API pgmcoop_send_finish().

Opciones filesend:

Opción corta	Opción larga	Descripción
`-V`	`--version`	Muestra la versión de la utilidad
`-h`	`--help`	Muestra una pantalla de ayuda con la descripción de los parámetros y las opciones
`-t`	`--stats`	Activa el muestreo de estadísticas luego de la transmisión
`-H`	`--http`	Activa consola http de librería OpenPGM para ver estadísticas en tiempo real
`-f <argumento>`	`--filename <argumento>`	Especifica el nombre del archivo a enviar. Parámetro mandatorio
`-n <argumento>`	`--network <argumento>`	Dirección de red a utilizar. Valor predeterminado: 225.0.0.1
`-s <argumento>`	`--source <argumento>`	Puerto fuente a utilizar en el envío
`-r <argumento>`	`--rate <argumento>`	Límite para la tasa de transferencia a usar en el envío. Default: 100000 bps
`-l <argumento>`	`--loglevel <arugmento>`	Nivel de mensajes de log a mostrar

Tabla 4: Opciones posibles para utilidad filesend

Utilidad filerecv

Se inicializa una estructura pgm_options usando la función pgmcoop_init_options() y de forma predeterminada se utiliza la dirección de grupo multicast IPv4 225.0.0.1 para recibir. Luego se invoca la función pgmcoop_recv() con el nombre de archivo para utilizar como almacenamiento de los datos recibidos y las opciones que contiene la estructura pgm_options. La utilidad soporta varias opciones de línea de comandos.

Opciones filerecv:

Opción corta	Opción larga	Descripción
`-V`	`--version`	Muestra la versión de la utilidad
`-h`	`--help`	Muestra una pantalla de ayuda con la descripción de los parámetros y las opciones
`-t`	`--stats`	Activa el muestreo de estadísticas luego de la transmisión
`-H`	`--http`	Activa consola http de librería OpenPGM para ver estadísticas en tiempo real
`-d`	`--debug`	Activa modo debug con mensajes detallados para depuración de código
`-f <argumento>`	`--filename <argumento>`	Especifica el nombre del archivo sobre el cual recibir. Parámetro mandatorio
`-n <argumento>`	`--network <argumento>`	Dirección de red a utilizar. Valor predeterminado: 225.0.0.1
`-l <argumento>`	`--loglevel <argumento>`	Nivel de mensajes de log a mostrar

Tabla 5: Opciones posibles para utilidad filerecv

Utilidades para pruebas de rendimiento

Para facilitar la tarea de realizar las pruebas de rendimiento y medición de performance de los 3 protocolos se desarrollaron, como parte de este trabajo de investigación, dos utilidades (**`msend`** y **`mrecv`**) para ejecutar en el transmisor y los receptores. Estas utilidades implementan tanto el protocolo ad-hoc MDP, como también PGM y PGMCOOP usando respectivamente las librerías openpgm y pgmcoop.

La utilidad **`mrecv`** está implementada con dos hilos de ejecución donde cada uno abre un puerto UDP y queda a la espera de la transmisión del padrón de prueba. Uno de los hilos/puertos es para las transmisiones MDP y el otro para PGM/PGMCOOP ya que ambas variantes tienen el mismo código a nivel receptores. Los puertos utilizados en las pruebas fueron: 7500 para MDP y 7600 para PGM/PGMCOOP.

Ambos hilos de ejecución registran estampillas de tiempo para el comienzo y el fin de la transmisión, esto permite registrar luego el tiempo que tardó cada receptor en recibir el archivo y la tasa de transferencia lograda.

Para el caso de MDP, además se registran la cantidad de bloques recibidos en la fase multicast, los bloques perdidos (que tuvieron que retransmitirse) y el tiempo que llevó dicha etapa de retransmisión.

Para PGM/PGMCOOP se hace uso de una función de la librería openpgm que permite listar una serie de estadísticas importantes acerca del rendimiento del protocolo. Estas estadísticas se envían al mismo archivo de registro del receptor para luego analizar. Un valor importante que se encuentra dentro de esas estadísticas es la cantidad de NAKs generados, la cantidad de NAKs realmente enviados y cuantos fueron suprimidos por el protocolo, etc.

La utilidad **`msend`** tiene implementados los 3 modos de transmisión (mdp, pgm, pgmcoop) y el mismo se puede seleccionar mediante un parámetro. Otro parámetro es el nombre del archivo a transmitir y por último se debe especificar la tasa de transmisión inicial para el envío.

Antes de comenzar a enviar los bloques del archivo, la utilidad envía el nombre del archivo, el tamaño (en bytes), la cantidad de bloques y por último un checksum en MD5 del contenido del archivo en un encabezado. Esto es utilizado por los receptores para verificar, luego de terminada la transmisión, que los contenidos del archivo recibido son exactamente idénticos al original. Luego comienza a enviar los bloques del archivo usando la tasa de transmisión inicial especificada en los parámetros.

Al finalizar la transmisión, se detienen los receptores y cada uno guarda un archivo de registro con sus respectivos

resultados. Estos registros, junto al que deja el transmisor, se recuperan y los valores se guardan en planillas de cálculo para análisis posterior.

Ejemplos de ejecución:

```
# msend -f asociado.tar -m mdp -b 200 -v
```

Envía el archivo asociado.tar con el protocolo mdp usando 200 KB/s de tasa inicial y activa el modo “verbose” para mostrar las estadísticas de la transmisión al finalizar

```
# msend -f asociado.tar -m pgmcoop -b 4000
```

Envía el archivo asociado.tar con el protocolo pgmcoop usando 4000 KB/s como tasa inicial.

Capítulo 8: Mediciones y performance

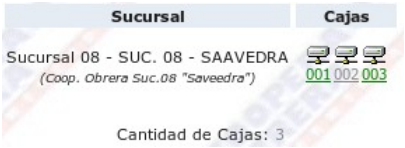
Escenarios de testing

Para realizar la experimentación de los protocolos y sus respectivas mediciones de performance, en este trabajo se utilizaron 6 escenarios distintos que intentan representar una variedad de tecnologías de equipos y segmentos de red. Los escenarios son sucursales reales de la Cooperativa Obrera, ya que de esta forma se aprovecha la estructura ya creada en cada caso, con sus equipamientos de red reales y de distintas generaciones con características de hardware particulares a cada sucursal.
Para garantizar la veracidad de los resultados de las pruebas al máximo posible, en todos los casos las mismas se realizaron disponiendo de la red a voluntad y deteniendo completamente el tráfico normal de las aplicaciones.

Los 3 primeros escenarios utilizados (A, B y C) tienen redes de 10 Mb/s implementadas con hubs. Los restantes 3 escenarios (D, E y F) son sucursales que poseen red switcheada de 100 Mb/s con equipos full duplex.

El detalle de los escenarios utilizados para experimentación es el siguiente:

<u>Escenario A: Sucursal 08 (Saavedra)</u>



- Resumen: red “lenta”, pocos receptores, hw limitado
- Equipamiento de red:
  - Tipo: Hub
  - Velocidad: 10 Mb/s
- Servidor: IBM x3200
  - Procesador: Intel Xeon CPU 3040 @ 1.86GHz
  - Memoria RAM: 1024 MB
  - Interfaz de red: Broadcom NetXtreme BCM5721 Gigabit Ethernet PCI Express @ 10 Mb/s
- Receptores: 3 (tres)
  - Procesador: AMD-K6 @ 332 Mhz
  - Memoria RAM: 16 MB (14 MB utilizables, ya que 2 MB son necesarios para el kernel)
  - Interfaz de red: PCnet, 10 Mb/s

<u>Escenario B: Sucursal 21 (Pigüe)</u>

- Resumen: red “lenta”, nro medio de receptores, hw limitado
- Equipamiento de red:
  - Tipo: Hub
  - Velocidad: 10 Mb/s

- Servidor: IBM x3400
  - Procesador: Intel Xeon CPU 5110 @ 1.60GHz
  - Memoria RAM: 1024 MB
  - Interfaz de red: Broadcom NetXtreme BCM5721 Gigabit Ethernet PCI Express @ 10 Mb/s
- Receptores: 3 (tres)
  - Procesador: AMD-K6 @ 332 Mhz
  - Memoria RAM: 16 MB (14 MB utilizables, ya que 2 MB son necesarios para el kernel)
  - Interfaz de red: PCnet, 10 Mb/s

Escenario C: Sucursal 02 (Villa Mitre – B. Blanca)

- Resumen: red “lenta”, nro medio de receptores, hw moderno
- Equipamiento de red:
  - Tipo: Hub
  - Velocidad: 10 Mb/s
- Servidor: IBM x3400
  - Procesador: Intel Xeon CPU 5110 @ 1.60GHz
  - Memoria RAM: 1024 MB
  - Interfaz de red: Broadcom NetXtreme BCM5721 Gigabit Ethernet PCI Express @ 10 Mb/s
- Receptores: 8 (ocho)
  - Procesador: Intel Atom N270 @ 1.60 GHz
  - Memoria RAM: 896 MB (883 MB utilizables)
  - Interfaz de red: Realtek RTL8101E 10/100 Fast Ethernet @ 10Mb/s

Escenario D: Sucursal 96 (Centenario - Neuquén)

- Resumen: red “normal”, servidor medio, pocos receptores high-end
- Equipamiento de red:
  - Tipo: Switch
  - Velocidad: 100 Mb/s
- Servidor: IBM x3400
  - Procesador: Intel Xeon CPU 5110 @ 1.60GHz
  - Memoria RAM: 1024 MB
  - Interfaz de red: Broadcom NetXtreme BCM5721 Gigabit Ethernet PCI Express @ 100 Mb/s
- Receptores: 4 (cuatro)
  - Procesador: Intel Atom N270 @ 1.60GHz
  - Memoria RAM: 900 MB utilizables
  - Interfaz de red: Realtek RTL8101E 10/100 Fast Ethernet @ 100Mb/s

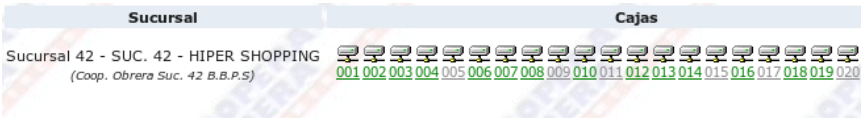
Escenario E: Sucursal 36 (Olavarría – Bs. As.)

- Resumen: red “normal”, nro medio de receptores, servidor high-end, hw moderno
- Equipamiento de red:

    - Tipo: Switch
    - Velocidad: 100 Mb/s
- Servidor: Dell PowerEdge T320
    - Procesador: Intel Xeon E5-2403 0 @ 1.80GHz
    - Memoria RAM: 4 GB
    - Interfaz de red: Broadcom Corporation NetXtreme BCM5720 Gigabit Ethernet PCIe @ 100Mb/s
- Receptores: 9 (nueve)
    - Procesador: Intel Celeron CPU @ 2.00GHz
    - Memoria RAM: 256 MB (250 MB utilizables)
    - Interfaz de red: Realtek RTL8139 10/100 Fast Ethernet @ 100Mb/s

Escenario F: Sucursal 42 (Hipershopping – B. Blanca)

- Resumen: red “normal”, hw Cisco, servidor high-end, muchos receptores
- Equipamiento de red:
    - Tipo: Switches Cisco línea 2960
    - Velocidad: 100 Mb/s
- Servidor: Dell PowerEdge T320
    - Procesador: Intel Xeon E5-2403 0 @ 1.80GHz
    - Memoria RAM: 4 GB
    - Interfaz de red: Broadcom Corporation NetXtreme BCM5720 Gigabit Ethernet PCIe @ 100Mb/s
- Receptores: 20 (veinte)
    - Procesador: Intel Atom N270 @ 1.60GHz
    - Memoria RAM: 900 MB utilizables
    - Interfaz de red: Realtek RTL8101E 10/100 Fast Ethernet @ 100Mb/s



Pruebas básicas de latencia

Para probar de manera básica la latencia existente en los distintos escenarios, se realizaron pruebas de envío y recepción de paquetes ICMP con la conocida herramienta de diagnóstico “ping”. Se trabajó con distintos tamaños de paquete, haciendo una prueba básica con datagramas de 56 bytes y luego pruebas con 1024, 2048, 4096, 8192 y 16384 bytes. Las pruebas se realizaron tomando uno de los receptores como modelo y se registró el round-trip-time promedio (avg) de cada prueba. Los resultados los volcamos en la siguiente tabla (en todos los casos la unidad de medición es milisegundos):

	A – suc08	B – suc21	C – suc02	D – suc96	E – suc36	F – suc42
ping -c 100 server (avg)	0.403	0.408	0.526	0.143	0.205	0.210
ping -c 10 -s 1024 server (avg)	1.997	2.001	2.315	0.469	0.556	0.557
ping -c 10 -s 2048 server (avg)	3.713	3.721	4.428	0.722	0.812	0.821
ping -c 10 -s 4096 server (avg)	7.110	7.160	7.540	1.081	1.153	1.150
ping -c 10 -s 8192 server (avg)	14.273	14.463	14.394	1.752	2.201	1.837
ping -c 10 -s 16384 server (avg)	27.620	27.789	28.269	3.127	3.161	3.164

Tabla 6: Resultados de pruebas de latencia

De los números que arrojaron éstas pruebas, en los escenarios A, B y C observamos un elevado incremento en los tiempos a medida que aumentamos el tamaño del datagrama. Este incremento es debido a la velocidad del segmento de red que en estos dos casos es de 10Mb/s.

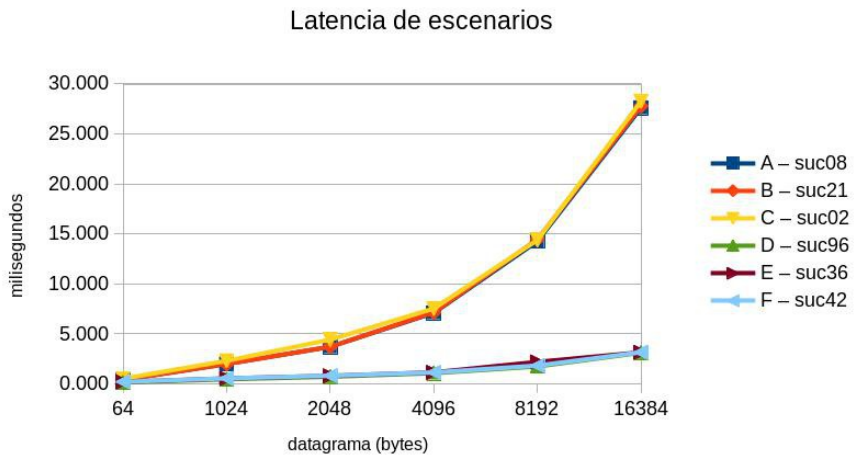


Ilustración 11: Medición de latencia con PING

En los escenarios D, E y F los tiempos son también muy similares y se incrementan prácticamente en forma lineal a medida que aumentamos el tamaño del datagrama.

Pruebas de ancho de banda multicast disponible a nivel red (IP)

Para establecer una “cota superior” en el rendimiento de los protocolos que luego vamos a medir, utilizamos la herramienta

de medición “iperf”[25] para crear un flujo de datos TCP/UDP y medir el rendimiento en cada uno de los escenarios. Esta métrica nos va a dar una pauta de cuanto puede ser el rendimiento máximo de los protocolos multicast que probaremos en cada uno de los escenarios. Este rendimiento máximo estará dado, por supuesto, por la combinación de: tipo de servidor, equipamiento de red (hub/switch y velocidad/ancho de banda), tipo de receptores (hw) y cantidad de los mismos. [SCHROD00]

Para las pruebas con iperf utilizaremos uno de los receptores en modo servidor y el servidor de la sucursal será utilizado en modo cliente para generar el tráfico UDP multicast. Se harán dos grupos de pruebas, un grupo de pruebas con buffers de tamaño 100 bytes y otro grupo de pruebas con buffers de tamaño 1500 bytes. En ambos grupos se probará con distintos valores de ancho de banda del tráfico UDP a generar (1M, 10M, 20M, 50M y 100M).

Pruebas con buffer de 100 bytes

Comando lado servidor:

```
# iperf -s -u -B 239.192.0.1 -l 100  -w 1M
```

Comando lado cliente, donde <B> va variando con los valores 1M, 10M, 20M, 50M y 100M

```
# iperf -c 239.192.0.1 -b <B> -T 16 -l 100 -w 10M
```

buffers 100 bytes **bandwidth to send**	A	B	C	D	E	F
			(kb/s)			
1M	990	994	1000	1000	1000	1000
10M	5870	5930	5960	10000	10000	10000
20M	5980	5900	5980	20000	20000	20000
50M	5890	5790	5980	50000	49900	49900
100M	5310	5960	5980	55200	55300	54500

Tabla 7: Resultados de pruebas iperf con buffers de 100 bytes

La tabla 7 muestra los resultados en kb/s de las pruebas con iperf (buffers de 100 bytes). Es evidente que en estas pruebas se manifiesta la diferencia de ancho de banda entre las redes de 10 y 100 Mb/s.

Como se puede observar en el gráfico, en todos los escenarios, a medida que vamos incrementando el ancho de banda transmitido, el nivel de la utilización de la red sube hasta casi cubrir un 60% del ancho de banda disponible. Esta es una medida de cuanto tráfico multicast puede manejar el equipamiento de red utilizado y se puede tomar como máximo posible utilizando buffers pequeños; por lo tanto en estos escenarios el cuello de botella es la red.

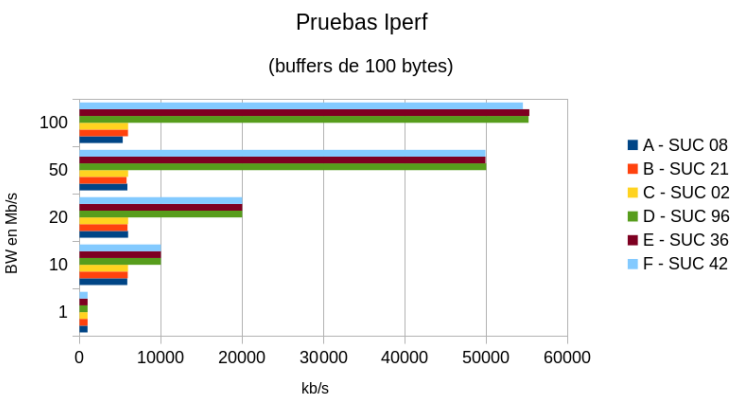


Ilustración 12: Pruebas iperf con buffers de 100 bytes

[25] IPERF: https://code.google.com/p/iperf/ - http://iperf.sourceforge.net/

Por otro lado, en los escenarios D, E y F, con redes conmutadas de 100 Mb/s, las tasas de transferencia corresponden exactamente al ancho de banda utilizado hasta llegar a la prueba de 100 Mb/s en la que no se puede sostener ese ancho de banda y los tres casos se llega a 55 Mb/s.

Pruebas con buffer de 1500 bytes

Comando lado servidor:

```
# iperf -s -u -B 239.192.0.1 -l 1500  -w 1M
```

Comando lado cliente, donde <B> va variando con los valores 1M, 10M, 20M, 50M y 100M

```
# iperf -c 239.192.0.1 -b <B> -T 16 -l 1500 -w 10M
```

buffers 1500 bytes **bandwidth to send**	A	B	C	D (kb/s)	E	F
1M	994	999	1000	1000	1000	1000
10M	9040	8910	9130	10000	10000	10000
20M	9130	9040	9160	20000	20000	20000
50M	9180	9160	9140	50000	50000	49300
100M	9140	9190	9140	89300	92400	89300

Tabla 8: Resultados de pruebas iperf con buffers de 1500 bytes

La tabla 8 muestra los resultados en kb/s de las pruebas con iperf (buffers de 1500 bytes).

En este caso, el comportamiento es similar, sólo que el hecho de usar buffers mas grandes permite lograr máximos mas altos en todos los escenarios. Esto se debe a que en cada datagrama colocamos mas datos (“payload”) y de esta forma se aprovecha mejor el ancho de banda de la red.
En todos los escenarios se logran mejores resultados utilizando casi el máximo ancho de banda limitado por el equipamiento de red, y, evidentemente, los receptores se ven muy beneficiados por el incremento de buffer ya que el rendimiento es significativamente mayor al caso con buffers de 100 bytes. En todos los escenarios se logran tasas de utilización de la red de mas del 90%.

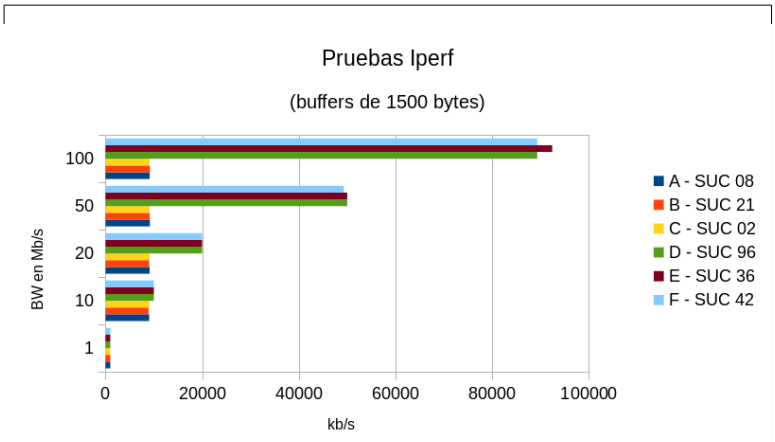


Ilustración 13: Pruebas iperf con buffers de 1500 bytes

Pruebas de ancho de banda multicast a nivel aplicación (PGM)

Luego de hacer pruebas para determinar el ancho de banda disponible a nivel IP multicast, nos interesa determinar, mas específicamente a nivel aplicación y para el protocolo PGM cual es el rendimiento de la red en cada uno de los escenarios. Para poder determinar este rendimiento, utilizamos la herramienta “pgmping” provista por la distribución de la librería

OpenPGM. Esta herramienta permite medir entre dos estaciones, una en modo “transmisor” generando tráfico y otra en modo “receptor” recibiendo y repitiendo el tráfico recibido en la red para ser recibido nuevamente por el transmisor. La herramienta va generando mensajes con las estadísticas observadas.

Comando lado servidor:

```
# pgmping -n "eth0;239.192.0.1" -r 800000000 -p 3065 -e
```

Comando lado cliente, donde <B> va variando con los valores 500, 1000, 5000 y 10000 y representa la cantidad de mensajes por segundo.

```
# pgmping -n "eth0;239.192.0.1" -r 800000000 -p 3065 -m <B>
```

Resultados de las pruebas con la herramienta "pgmping" en kb/s:

pkts/seg	A	B	C	D	E	F
500	1730	2140	2240	4340	4350	4350
1000	790	980	1130	8690	8690	8690
5000	loss	500	730	37100	35400	43770
10000	loss	loss	loss	loss	loss	loss

Tabla 9: Resultados de pruebas con pgmping

En éstas pruebas vamos aumentando la cantidad de mensajes a enviar por segundo. En los escenarios A, B y C con redes de 10Mb/s, el comportamiento de cada caso es bastante diferente. El escenario C muestra una performance un poco mejor que el B, que a su vez es levemente superior al escenario A. El limitado hardware de red de estos tres escenarios hace que a medida que elevamos la cantidad de paquetes por segundo que se envían el segmento de red se empieza a saturar muy rápidamente; observandose, en los casos de 5000 o mas paquetes por segundo, un alto número de paquetes perdidos.

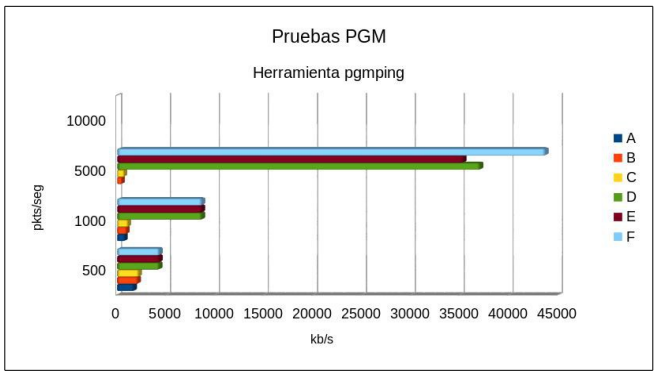


Ilustración 14: Gráfico comparativo de rendimiento con pgmping

Los escenarios D, E y F en éstas pruebas se comportan de manera muy similar, lograndosé tasas mucho mas altas debido, por supuesto, a un segmento de red con mas ancho de banda.
Asimismo, no es de sorprender que el escenario F que tiene un segmento de red implementado con switches Cisco, logre las mejores cifras de todas las pruebas.

Debe apreciarse, de todos modos, que con tráfico multicast PGM no es posible lograr los porcentajes de utilización de red de mas del 90% como se lograban con tráfico multicast puro en las pruebas de iperf del punto anterior.

Performance MDP, PGM y PGMCOOP

Para las pruebas de envío multicast, tomamos como ejemplo el envío diario del padrón de asociados que hacemos desde el servidor de cada sucursal a todas las cajas registradoras. El tamaño de este padrón puede variar entre sucursales ya que cada sucursal tiene en su padrón los asociados que han sido utilizados en la misma. De todas maneras para comparar la performance de los protocolos vamos a utilizar la tasa de transferencia y la cantidad/porcentaje de bloques perdidos/retransmitidos en el caso de MDP.

Esquema genérico de las pruebas

Para realizar las pruebas de rendimiento se utilizaron las utilidades de simulación msend y mrecv descriptas en el capítulo anterior. Estas utilidades registran una serie de estadísticas muy útiles durante su ejecución y se muestran como parte de la salida estándar al finalizar la misma.

Ejemplo de registros de un envío en MDP con 600 KB/s:

```
TRANSMISOR:

MSEND PARAMETERS:
* PROTOCOL: mdp
* FILE: asociado.tar
* SIZE: 9420800
* BLOCK SIZE: 2048 [4600]
* MD5SUM: 'fb2d15e9aff723b5f8ab745241a4536c'
* MDP BWLIMIT: 614400 bytes/sec
SENDING METADATA...
STARTING TRANSMISSION...
[0%.............25%............50%.............75%............100%] 599kB/sec

TOTAL BLOCKS: 4600
TRANSMISSION TIME: 15.39 (597 kB/sec)

RECEPTOR SIN BLOQUES PERDIDOS:

MDP [RECEIVED/MISSING/TOTAL BLOCKS] : Count (4600/0/4600)
MDP [END MULTICAST] : Time (15.38 - 598 kB/sec)
MDP [TRANSMISSION TIME] : Time (15.38 - 598 kB/sec)
MDP [MD5SUM CHECK BEGIN]
MDP [MD5SUM CHECK END] : Time (3.19)
MDP [CONSISTENCY CHECK] : MD5SUM (fb2d15e9aff723b5f8ab745241a4536c) [   OK   ]

RECEPTOR CON BLOQUES PERDIDOS:

MDP [RECEIVED/MISSING/TOTAL BLOCKS] : Count (4029/571/4600)
MDP [END MULTICAST] : Time (15.38 - 524 kB/sec)
MDP [BEGIN MISSING BLOCKS]
MDP [END MISSING BLOCKS] : Time (6.02 - 189 kB/sec)
MDP [TRANSMISSION TIME] : Time (21.40 - 429 kB/sec)
MDP [MD5SUM CHECK BEGIN]
MDP [MD5SUM CHECK END] : Time (2.97)
MDP [CONSISTENCY CHECK] : MD5SUM (fb2d15e9aff723b5f8ab745241a4536c) [   OK   ]
```

Ejemplo de registros de un envío en PGMCOOP con 600 KB/s:

TRANSMISOR:

```
STARTING TRANSMISSION...
PGMCOOP using: cwnd=40,cc_wnd=5,a=1024,b=0.96,th=1250000
+ Adjusting max_rte to 657408 [avg: 613708 bps] cwnd=43, cc_cwnd=6 (wnd)
* NAK total: 0 - NAK rate: 0.00
+ Adjusting max_rte to 703427 [avg: 639938 bps] cwnd=47, cc_cwnd=7 (wnd)
* NAK total: 1 - NAK rate: 1.20
+ Adjusting max_rte to 752667 [avg: 667947 bps] cwnd=51, cc_cwnd=8 (wnd)
* NAK total: 3 - NAK rate: 2.31
+ Adjusting max_rte to 805354 [avg: 696949 bps] cwnd=55, cc_cwnd=9 (wnd)
* NAK total: 4 - NAK rate: 2.21
* NAK rate threshold triggered: 4.71
* NAK total: 9 - NAK rate: 4.71
=> Congestion control started
- Adjusting max_rte to 773140 [avg: 702993 bps] cwnd=53, cc_cwnd=9 (cc)
# Usleeping for 26315. => Congestion control finished
* NAK rate threshold triggered: 4.52
* NAK total: 10 - NAK rate: 4.52
=> Congestion control started
- Adjusting max_rte to 742215 [avg: 709846 bps] cwnd=51, cc_cwnd=9 (cc)
# Usleeping for 27027.... => Congestion control finished
* NAK rate threshold triggered: 4.74
* NAK total: 12 - NAK rate: 4.74
=> Congestion control started
- Adjusting max_rte to 712527 [avg: 713145 bps] cwnd=49, cc_cwnd=9 (cc)
# Usleeping for 28571.... => Congestion control finished
* NAK rate threshold triggered: 8.30
* NAK total: 22 - NAK rate: 8.30
=> Congestion control started
- Adjusting max_rte to 684026 [avg: 713022 bps] cwnd=48, cc_cwnd=9 (cc)
# Usleeping for 29411. => Congestion control finished
* NAK rate threshold triggered: 5.05
* NAK total: 14 - NAK rate: 5.05
=> Congestion control started
- Adjusting max_rte to 656665 [avg: 710435 bps] cwnd=47, cc_cwnd=9 (cc)
# Usleeping for 31250. => Congestion control finished
+ Adjusting max_rte to 702632 [avg: 699593 bps] cwnd=51, cc_cwnd=10 (wnd)
* NAK total: 8 - NAK rate: 2.47
* NAK rate threshold triggered: 4.66
* NAK total: 17 - NAK rate: 4.66
=> Congestion control started
- Adjusting max_rte to 674527 [avg: 700508 bps] cwnd=49, cc_cwnd=10 (cc)
# Usleeping for 30303. => Congestion control finished
* NAK rate threshold triggered: 4.21
* NAK total: 16 - NAK rate: 4.21
=> Congestion control started
- Adjusting max_rte to 647546 [avg: 698730 bps] cwnd=48, cc_cwnd=10 (cc)
# Usleeping for 31250. => Congestion control finished
* NAK rate threshold triggered: 4.09
* NAK total: 16 - NAK rate: 4.09
=> Congestion control started
- Adjusting max_rte to 621645 [avg: 695249 bps] cwnd=47, cc_cwnd=10 (cc)
# Usleeping for 32258. => Congestion control finished
* NAK rate threshold triggered: 4.05
* NAK total: 17 - NAK rate: 4.05
=> Congestion control started
- Adjusting max_rte to 596780 [avg: 689350 bps] cwnd=46, cc_cwnd=10 (cc)
# Usleeping for 34482. => Congestion control finished
* NAK rate threshold triggered: 4.98
* NAK total: 23 - NAK rate: 4.98
=> Congestion control started
- Adjusting max_rte to 572909 [avg: 679671 bps] cwnd=45, cc_cwnd=10 (cc)
# Usleeping for 35714...........
TOTAL BLOCKS: 472
TRANSMISSION TIME: 14.01 (656 kB/sec)
```

En los registros de la transmisión PGMCOOP se observa el detalle de los tamaños de ventana que usa el protocolo, los ajustes que hace para aumentar la tasa de transferencia (líneas con prefijo “+”) y comienzos y finalizaciones de los períodos de control de congestión y respectivas disminuciones de la tasa de transferencia (líneas con prefijo “-”).

Los registros de los receptores en los modos PGM y PGMCOOP son similares a los de MDP, solo que, por supuesto, no detallan información de bloques perdidos. En el siguiente ejemplo, mostramos algunos de las estadísticas relevantes que muestra en los receptores la utilidad mrecv usando la funciones de volcado de estadísticas de la librería openpgm al terminar la transmisión:

```
PGM [RECEIVED/MISSING/TOTAL BLOCKS] : Count (472/0/472)
PGM [TRANSMISSION TIME] : Time (16.60 - 555 kB/sec)
PGM [MD5SUM CHECK BEGIN]
PGM [MD5SUM CHECK END] : Time (2.80)
PGM [CONSISTENCY CHECK] : MD5SUM (fb2d15e9aff723b5f8ab745241a4536c) [   OK   ]
Performance information:
Data bytes received 9442460
Data packets received 945
NAK failures 0
Bytes received 12275931
Checksum errors 0
Malformed SPMs 0
Malformed ODATA 0
Malformed RDATA 0
Malformed NCFs 0
Packets discarded 182
Losses 0
Bytes delivered to app 9442460
Packets delivered to app 473
Duplicate SPMs 0
Duplicate ODATA/RDATA 12
NAK packets sent 41
NAKs sent 229
NAKs retransmitted 0
NAKs suppressed 293
Malformed NAKs 0
Outstanding NAKs 0
NAK repair min time 215673 µs
NAK repair mean time 0 µs
NAK repair max time 4220199 µs
```

Luego de realizadas las pruebas, se volcaron los resultados en una planilla de cálculo para cada escenario. El formato y organización de estas planillas se detalla en la sección siguiente donde se describen los escenarios de prueba.

Escenarios de prueba

Se dividieron las pruebas en dos grupos de escenarios: aquellos 3 escenarios que tienen redes de 10 Mb/s con repetidores (hubs) y los otros 3 que poseen el segmento de red implementado con switches de 100 Mb/s y tecnología full duplex.

En el caso de 10 Mb/s se realizaron pruebas usando las utilidades msend y mrecv con tasas iniciales de transferencia de 200, 400, 600 y 800 KB/s. Para cada combinación de escenario y tasa inicial de transferencia se realizaron 5 pruebas de transferencia completas con cada uno de los 3 protocolos. Esto permitió analizar los resultados tomando promedio de la tasa de transferencia y del tiempo de transmisión y luego se calculó varianza y desviación estándar de estos resultados usando la tasa de transferencia como variable aleatoria.

Para los escenarios de 100 Mb/s se utilizó el mismo esquema de pruebas, pero usando como tasas de transferencia iniciales los siguientes valores: 2000, 4000 y 6000 KB/s. No se realizaron pruebas con 8000 KB/s porque ambas variantes de PGM se veían fuertemente afectadas por la cantidad de NAKs y los resultados no eran satisfactorios.

Los resultados de cada uno de los escenarios se volcaron a una planilla de cálculo organizada con una hoja para cada tasa de transferencia experimentada y en cada hoja se agruparon los resultados de los 3 protocolos para todos los receptores de las 5

ejecuciones realizadas. El formato de las hojas de esas planillas de cálculo, que se adjuntan como anexos a este trabajo de investigación, es el siguiente:

escenario B (suc 21)

	BW	600	MDP								PGM				PGMCOOP			
		bloques	ok	missing	tmulticast	tmissing	time	rate (kB/s)	var	desv	time	rate (kB/s)	var	desv	time	rate (kB/s)	var	desv
TEST 1	caja1	4600	4029	571	21.40	6.02	27.42	335.52	7654.557		19.59	469.63	231.527		16.02	574.28	174.162	
	caja2	4600	4414	186	19.03	3.65	22.68	405.64	301.651		20.09	457.94	723.831		16.94	543.09	323.706	
	caja3	4600	4290	310	19.91	4.54	24.45	376.28	2184.041		19.59	469.63	231.527		16.96	542.45	347.161	
	caja4	4600	4600	0	15.38	0.00	15.38	598.18	30683.691		19.59	469.63	231.527		16.89	544.70	268.438	
	caja5	4600	4182	418	21.08	5.23	26.31	349.68	5378.010		19.57	470.11	217.152		17.53	524.81	1315.548	
TEST 2	caja1	4600	4165	435	19.83	4.44	24.27	379.07	1930.990		18.22	504.94	403.859		15.90	578.62	307.345	
	caja2	4600	4248	352	19.54	4.16	23.70	388.19	1212.864		18.22	504.94	403.859		15.90	578.62	307.345	
	caja3	4600	4439	161	16.83	1.45	18.28	503.28	6443.342		18.22	504.94	403.859		15.89	578.98	320.245	
	caja4	4600	4600	0	15.38	0.00	15.38	598.18	30683.691		18.23	504.66	392.803		17.07	538.96	489.642	
	caja5	4600	4030	570	21.63	6.06	27.69	332.25	8237.729		18.22	504.94	403.859		16.12	570.72	92.824	
TEST 3	caja1	4600	4391	209	19.39	3.97	23.36	393.84	851.252		18.19	505.77	438.024		14.86	619.11	3367.089	
	caja2	4600	4145	455	21.07	5.69	26.76	343.80	6275.037		18.20	505.49	426.469		14.52	633.61	5259.691	
	caja3	4600	4278	322	20.10	4.72	24.82	370.67	2739.793		18.19	505.77	438.024		14.52	633.61	5259.691	
	caja4	4600	4389	211	20.55	5.16	25.71	357.84	4247.706		18.19	505.77	438.024		15.89	578.98	320.245	
	caja5	4600	4387	213	19.49	4.11	23.60	389.83	1101.001		18.20	505.49	426.469		15.30	601.31	1617.817	
TEST 4	caja1	4600	4358	242	17.34	1.95	19.29	476.93	2907.281		19.62	468.91	253.895		16.96	542.45	347.161	
	caja2	4600	4599	1	15.41	0.02	15.43	596.24	30008.371		19.63	468.67	261.565		16.99	541.49	383.772	
	caja3	4600	4489	111	17.42	2.04	19.46	472.76	2475.341		19.62	468.91	253.895		16.97	542.13	359.175	
	caja4	4600	4218	382	18.48	3.10	21.58	426.32	10.948		19.64	468.43	269.341		16.96	542.45	347.161	
	caja5	4600	4591	9	15.63	0.24	15.87	579.71	24554.357		19.62	468.91	253.895		16.96	542.45	347.161	
TEST 5	caja1	4600	4216	384	19.54	4.16	23.70	388.19	1212.864		19.34	475.70	83.637		18.31	502.46	3437.174	
	caja2	4600	4600	0	15.38	0.00	15.38	598.18	30683.691		19.10	481.68	10.036		16.94	543.09	323.706	
	caja3	4600	4439	161	18.27	2.88	21.15	434.99	143.433		19.10	481.68	10.036		16.60	554.22	47.172	
	caja4	4600	4459	141	17.03	1.65	18.68	492.51	4829.348		19.10	481.68	10.036		17.08	538.64	503.706	
	caja5	4600	4255	345	19.38	3.99	23.37	393.67	861.114		19.10	481.68	10.036		15.84	580.81	388.996	
		4600	***4352.4***	***247.56***	***18.58***	***3.17***	***21.75***	***423.01***	***8304.484***	***91.129***	***18.98***	***484.84***	***289.088***	***17.003***	***16.40***	***561.09***	***1050.245***	***32.407***

Ilustración 15: Formato planillas de resultados

Escenario A: Sucursal 08 – 10 Mb/s – 3 receptores

Para todas las pruebas se utilizó un padrón modelo de 9420800 bytes.

1. Protocolo MDP:
   i. **<u>200 KB/s:</u>** tiene un excelente comportamiento, logrando en promedio tasas de transferencia de casi 170 KB/s. Este valor se encuentra cerca del máximo posible, ya que al dividir el archivo a transmitir en 4600 bloques de 2 KB que tienen un pequeño encabezado, esto genera un overhead que impacta en la tasa de transferencia total. En ninguna de las 5 pruebas se registraron bloques perdidos, por lo que los receptores no tuvieron que solicitar retransmisiones.
   Tiempo promedio de transmisión: 54.17 segs (desviación de 0.418)
   ii. **<u>400 KB/s:</u>** se comporta muy bien. Solo en una de las pruebas y solo para un receptor se observaron retransmisiones (169 bloques de los 4600 totales). Esto se ve reflejado en los resultados de la varianza y desviación estándar dando un valor por encima de los 5.7 puntos.
   Tiempo promedio de transmisión: 23.16 segs (desviación de 5.715)
   iii. **<u>600 KB/s:</u>** en estas pruebas, el incremento de la tasa de transferencia inicial impacta significativamente en los receptores; en el 60% de los casos se registraron retransmisiones con un promedio de 185 bloques perdidos que tuvieron que retransmitirse. De todas maneras, se logran tasas de transferencia de casi 530 KB/s.
   Tiempo promedio de transmisión: 17.39 segs (desviación de 73.424)
   iv. **<u>800 KB/s:</u>** subiendo la tasa de transferencia a 800 KB/s observamos que en todos los casos el 100% de los receptores tuvieron que pedir retransmisión de bloques. En una de las pruebas, un receptor solo perdió 8

bloques y de esta manera logró recibir el padrón completo con 787 KB/s de tasa de rendimiento, pero hubo casos en los que solo se lograron 400 KB/s. Es claro que forzar tasas de transferencia mas altas no refleja un beneficio en los resultados. De hecho, en estas pruebas el promedio de bloques perdidos sube a 360 (casi un 8% de los bloques) y se logra aproximadamente el mismo tiempo promedio de transmisión, pero con desviación mucho mas alta.
Tiempo promedio de transmisión: 17.35 segs (desviación de 116.368)

2. Protocolo PGM
   i. **200 KB/s:** en la tasa de transferencia mas baja de las experimentadas el protocolo PGM se comporta de manera excelente, logrando resultados muy parecidos a los observados con el protocolo MDP. Las 5 pruebas fueron consistentes con resultados prácticamente idénticos
   Tiempo promedio de transmisión: 54.06 segs (desviación de 0.381)
   ii. **400 KB/s:** este protocolo también se ve beneficiado al realizar el primer incremento de la tasa de transferencia. Los resultados reflejan un comportamiento muy uniforme, sin desviaciones y tasas de transferencia de aproximadamente 390 KB/s.
   Tiempo promedio de transmisión: 23.49 segs (desviación de 0.142)
   iii. **600 KB/s**: en esta tasa es donde se comienzan a observar dificultades en los receptores para mantener el ritmo impuesto por el servidor. Sube considerablemente la cantidad de NAKs; esto impacta en las retransmisiones y en los tiempos de transmisión logrados en este escenario. En la comparación con MDP, a pesar de perder en la cifra de tiempo promedio de transmisión, PGM logra tener una desviación estándar considerablemente mas baja. Esto le permitirá tener mejores resultados que MDP en las pruebas de 800 KB/s.
   Tiempo promedio de transmisión: 18.87 segs (desviación de 29.577)
   iv. **800 KB/s:** en estos casos, a pesar de que los receptores no pueden sostener esa tasa de transferencia, se logran mejores resultados que en MDP. La estrategia utilizada por PGM para retransmitir, es claramente superior a la del protocolo MDP. Asi se logran tasas de mas de 623 KB/s contra un máximo de 530 KB/s logrado por MDP. Estos resultados fueron los mejores de todo el escenario A.
   Tiempo promedio de transmisión: 14.76 segs (desviación de 33.200)
3. Protocolo PGMCOOP
   i. **200 KB/s:** el protocolo experimental desarrollado en el presente trabajo tiene buenos resultados cuando la tasa de transferencia inicial es baja y puede aprovechar la estrategia de ir subiendo esta tasa a medida que vaya cumpliendo las ventanas de transmisión y no se presenten NAKs que disparen el mecanismo de control de congestión implementado (CC). La tasa de transferencia pudo crecer hasta 545 KB/s, logrando en promedio 348 KB/s. PGMCOOP superó a los otros dos protocolos y, para estos primeros casos, las desviación estándar es aceptable. El mecanismo de control de congestión solo se tuvo que utilizar 2 veces (1ra y 3ra prueba).
   Tiempo promedio de transmisión: 26.37 segs (desviación de 5.733)
   ii. **400 KB/s:** en este caso, MDP y PGM, también fueron superados. En promedio se lograron 543 KB/s teniendo tasas máximas de transmisión de hasta 900 KB/s. El mecanismo de control de congestión tuvo que actuar 27 veces en total demostrando su efectividad. La desviación estándar aumenta considerablemente.
   Tiempo promedio de transmisión: 17.22 segs (desviación de 30.401)
   iii. **600 KB/s:** en esta tasa de transferencia PGMCOOP logra su comportamiento óptimo ya que alcanza en promedio 571 KB/s. CC tuvo que intervenir 33 veces en total en las 5 pruebas ya que cuando se aumentaba la tasa de transferencia a mas de 800 KB/s, la tasa de recepción de NAKs lo disparaba. De todas maneras los resultados logrados son satisfactorios, superando nuevamente a los otros dos protocolos.
   Tiempo promedio de transmisión: 16.11 segs (desviación de 48.003)
   iv. **800 KB/s:** en las últimas 5 pruebas de PGMCOOP, forzar el mismo a comenzar la transmisión a 800 KB/s no lo beneficia, ya que el mecanismo de control de congestión se dispara inmediatamente debido a una alta tasa de recepción de NAKs. La tasa de transferencia ronda los 700 / 800 KB/s pero, el hecho de tener que retransmitir tantos bloques, hace que el rendimiento sea mucho menor. En estos casos, el resultado termina siendo muy similar a los escenarios de 600 KB/s pero, como en el caso de PGM, con valores mas altos de desviación estándar ya que hay casos en los que se lograron casi 700 KB/s y en otros menos de 500 KB/s.

Tiempo promedio de transmisión: 16.10 segs (desviación de 77.109)

Escenario B: Sucursal 21 – 10 Mb/s – 5 receptores

Para todas las pruebas se utilizó un padrón modelo de 9420800 bytes.

1. Protocolo MDP:
   i. **200 KB/s:** tiene un excelente comportamiento, logrando en promedio tasas de transferencia de casi los 200 KB/s. De manera similar al escnario anterior, en ninguna de las 5 pruebas se registraron bloques perdidos, por lo que los receptores no tuvieron que solicitar retransmisiones.
   Tiempo promedio de transmisión: 46.15 segs (desviación de 0.121)
   ii. **400 KB/s:** en estas pruebas se comienzan a observar algunos bloques perdidos (en un 28% de los casos y con un máximo de 173 bloques) que luego se manifiestan al observar las cifras de varianza y desviación estándar. De todas maneras, el protocolo se comporta muy bien y logra 393 KB/s de tasa de transferencia promedio.
   Tiempo promedio de transmisión: 23.37 segs (desviación de 10.286)
   iii. **600 KB/s:** al subir la tasa de transferencia a 600 KB/s se incrementan considerablemente los bloques perdidos y solo en 3 de los 25 casos no hubo pérdidas. En promedio se tuvieron que utilizar 3.17 segs para retransmitir bloques, lo cual impactó de manera importante en los resultados de estas pruebas. La tasa promedio lograda es de 423 KB/s.
   Tiempo promedio de transmisión: 21.75 segs (desviación de 91.129)
   iv. **800 KB/s:** en estos casos el incremento en la tasa de transferencia genera todavía mas bloques perdidos. En todos los casos hubo que realizar retransmisiones y en el peor caso se tuvieron que retransmitir 717 bloques, lo que demoró 8.61 segs. La tasa de transferencia promedio fue de 481 KB/s.
   Tiempo promedio de transmisión: 19.12 segs (desviación de 99.931)
2. Protocolo PGM
   i. **200 KB/s:** excelente comportamiento con resultados prácticamente idénticos en todos los casos. Se logran 199 KB/s con una desviación estándar bajísima
   Tiempo promedio de transmisión: 46.18 (desviación de 0.044)
   ii. **400 KB/s:** para esta tasa de transferencia el protocolo PGM también tiene excelentes resultados, logrando en promedio casi 392 KB/s con una desviación muy baja y tiempos muy similares al escenario anterior
   Tiempo promedio de transmisión: 23.47 segs (desviación de 7.614)
   iii. **600 KB/s**: para 600 KB/s ya se comienzan a observar dificultades debido a un alto número de NAKs. El máximo data rate que se pudo lograr fue de 505 KB/s con un promedio final de 484 KB/s. La desviación estándar no resultó significativa
   Tiempo promedio de transmisión: 18.98 segs (desviación de 17.003)
   iv. **800 KB/s:** estas pruebas, como en el escenario anterior, terminan siendo las de mejor resultado de todos los protocolos. Se lograron, en promedio, casi 612 KB/s con un máximo de 645 KB/s. La desviación es aceptable. MDP se ve superado ampliamente.
   Tiempo promedio de transmisión: 15.04 segs (desviación de 27.744)
3. Protocolo PGMCOOP
   i. **200 KB/s:** se obtiene una tasa de transferencia promedio de 351 KB/s con picos de hasta 567 KB/s. Buenos resultados y tiempos similares a los del escenario anterior.
   Tiempo promedio de transmisión: 26.21 segs (desviación de 3.173)
   ii. **400 KB/s:** comenzando a transmitir a 400 KB/s, el protocolo va aumentando la tasa de transferencia hasta lograr picos de mas de 800 KB/s. El promedio logrado es de 543 KB/s con una desviación estándar aceptable. El mecanismo de control de congestión debió ejecutarse 33 veces (sumando todas las pruebas).
   Tiempo promedio de transmisión: 16.93 segs (desviación de 31.958)
   iii. **600 KB/s:** al aumentar la tasa de transferencia inicial, el protocolo aumenta la tasa de transferencia, pero llegando a los 800 KB/s (valores similares a las pruebas del punto anterior) se dispara el control de congestión

debido a un alto número de NAKs. El promedio de tasa de transferencia termina siendo de 561 KB/s.
Tiempo promedio de transmisión: 16.40 segs (desviación de 32.407)

iv. **800 KB/s:** el control de congestión tiene que intervenir de manera inmediata para estas pruebas ya que la tasa de transferencia inicial es muy alta y continúa actuando hasta que la tasa ronda los 650 KB/s. Termina logrando resultados similares a los obtenidos con 400 y 600 KB/s: unos 566 KB/s.
Tiempo promedio de transmisión: 16.25 (desviación de 42.721)

Escenario C: Sucursal 02 – 10 Mb/s – 8 receptores

Para todas las pruebas se utilizó un padrón modelo de 9420800 bytes.

1. Protocolo MDP:
   i. **200 KB/s:** se comporta de manera excelente con resultados próximos a los 200 KB/s. Incluso, para experimentar, se hizo una prueba a 205 donde se lograron 204.58 y a 210 donde se lograron 209.57 KB/s. En todos los casos no hubo bloques perdidos. El promedio fue 202 KB/s con una pequeña desviación causada por variar el data rate.
   Tiempo promedio de transmisión: 45.43 segs (desviación de 3.981)
   ii. **400 KB/s:** para esta tasa de transferencia, también se logran excelentes resultados. De las 5 pruebas, 4 se hicieron a exactamente 400 KB/s y una de éstas se hizo a 405 KB/s. En promedio se lograron 400.14 KB/s con muy baja desviación. No hubo bloques perdidos.
   Tiempo promedio de transmisión: 22.99 segs (desviación de 2.034)
   iii. **600 KB/s:** aquí, en este escenario comenzamos a obtener resultados diferentes a los otros dos que tienen redes de 10Mb/s. Para este escenario no se observan bloques perdidos y se obtiene, en promedio, una tasa de transferencia de 601 KB/s.
   Tiempo promedio de transmisión: 15.29 segs (desviación de 3.953)
   iv. **800 KB/s:** de manera sorprendente, tampoco se observan bloques perdidos en este caso y se obtienen, en promedio, 802 KB/s. Las tasas iniciales variaron entre 800, 802, 805 y 810 KB/s. La desviación no es significativa.
   Tiempo promedio de transmisión: 11.47 segs (desviación de 3.659)
2. Protocolo PGM
   i. **200 KB/s:** con PGM también se obtienen excelentes resultados; 196 KB/s al transmitir a 200 KB/s, 201 KB/s al transmitir a 205 KB/s y 206 al transmitir a 210 KB/s. Son cifras predecibles y se observa un comportamiento uniforme y consistente.
   Tiempo promedio de transmisión: 45.71 segs (desviación de 4.590)
   ii. **400 KB/s:** sucede lo mismo que en el punto anterior. Se logran cifras muy cercanas a la tasa de transferencia configurada para los envíos y en promedio vemos casi 401 KB/s.
   Tiempo promedio de transmisión: 22.95 segs (desviación de 2.487)
   iii. **600 KB/s**: también este protocolo logra mantener excelentes cifras en esta tasa de transferencia. En promedio logramos 599 KB/s con muy baja desviación estándar.
   Tiempo promedio de transmisión: 15.36 segs (desviación de 1.882)
   iv. **800 KB/s:** en la última tasa de transferencia testeada, PGM también logra maximizar la tasa de transferencia logrando 801 KB/s. Las pruebas variaron entre 800, 802, 805 y 810 KB/s (como se hizo en MDP).
   Tiempo promedio de transmisión: 11.48 segs (desviación de 3.660)
3. Protocolo PGMCOOP
   i. **200 KB/s:** en las primeras pruebas de PGMCOOP se obtienen resultados similares a los 2 escenarios anteriores; se lograron casi 370 KB/s en promedio y nunca tuvo que ejecutarse el control de congestión implementado.
   Tiempo promedio de transmisión: 24.90 segs (desviación de 11.606)
   ii. **400 KB/s:** comenzando la transmisión con esta tasa de transferencia, se logran muy buenos resultados: en

promedio, casi 730 KB/s y tampoco fue necesario, en ninguno de los casos, ejecutar el control de congestión.
Tiempo promedio de transmisión: 12.62 segs (desviación de 27.517)

iii. **600 KB/s:** con una tasa de transferencia que, en sus picos máximos llegó a mas de 1 MB/s y que promedió 949 KB/s, este es el caso donde se logran los mejores resultados de todas las pruebas de 10Mb/s. No hubo que ejecutar CC. En algunos casos, se observaron receptores que tuvieron que solicitar un alto número de retransmisiones que generaron tasas de aproximadamente 600 o 700 KB/s. Evidentemente, en estas pruebas se está llegando al límite de capacidad de los receptores.
Tiempo promedio de transmisión: 9.69 segs (desviación de 68.434)

iv. **800 KB/s:** se obtienen resultados similares al caso anterior. Al forzar la tasa de transferencia a valores tan altos, algunos de los receptores experimentan alto número de retransmisiones y se observan diferencias considerables en los resultados. Esto se manifiesta en la alta desviación estándar de estas pruebas. De hecho los resultados de las pruebas de 600 KB/s son mejores. Por lo tanto, termina siendo contraproducente elevar tanto la tasa de transferencia al acercarnos al límite de la que se puede obtener con esta combinación de segmento físico de red y receptores.
Tiempo promedio de transmisión: 10.59 segs (desviación de 201.211)

Escenario D: Sucursal 96 – 100 Mb/s – 4 receptores

Para todas las pruebas se utilizó un padrón modelo de 48715120 bytes.

1. Protocolo MDP:
   i. **2000 KB/s:** en estas pruebas resultó que en mas de la mitad de los casos se perdieron bloques (11 de los 20). De todas maneras no fueron grandes cantidades (un promedio de 468 bloques). El protocolo en definitiva se comporta bien, con tiempos de recuperación de bloques menores a 1 segundo. La tasa promedio resultó ser de 1938 KB/s. La desviación resulta levemente significativa debido a las pérdidas.
   Tiempo promedio de transmisión: 24.54 segs (desviación de 53.045)
   ii. **4000 KB/s:** esta vez, en 9 de los 20 casos se registraron bloques perdidos, pero la cantidad de bloques es mas importante: 780 en promedio con un máximo de 2299. Los tiempos de recuperación se elevan a mas de 1 segundo y esto impacta de manera mas significativa en los resultados. Tasa promedio de 3636KB/s. La desviación también sube considerablemente.
   Tiempo promedio de transmisión: 13.08 segs (desviación de 367.186)
   iii. **6000 KB/s:** al subir la tasa de transferencia, como era previsible, vuelven a aumentar los bloques perdidos. Con un promedio de 1015, y algunos tiempos de recuperación de mas de 2 segundos (sobre 9 segundos de transmisión multicast), la tasa de transferencia queda bastante lejos de la ideal, logrando 5000 KB/s. La desviación termina siendo muy elevada.
   Tiempo promedio de transmisión: 9.51 segs (desviación de 945.533)
2. Protocolo PGM
   i. **2000 KB/s:** el protocolo PGM también logra una tasa de transferencia promedio de aproximadamente 1900 KB/s. El resultado es bueno, con una desviación aceptable.
   Tiempo promedio de transmisión: 25.01 segs (desviación de 52.102)
   ii. **4000 KB/s:** para tasas de 4000 KB/s, PGM también se ve penalizado por las retransmisiones. Se registraron picos de mas de 1000 NAKs en algunos receptores. Sin embargo termina logrando 3728 KB/s de promedio con una desviación en ascenso.
   Tiempo promedio de transmisión: 12.76 segs (desviación de 107.082)
   iii. **6000 KB/s**: al llevar la tasa de transferencia a 6000, evidentemente el protocolo PGM no se ve beneficiado en estas pruebas. Se logran en promedio 4364 KB/s, lo cual es bastante menos a lo logrado por MDP. Para algunos receptores se registraron mas de 8000 NAKs. De todas formas, la desviación estándar de estos casos es mucho menor a la de MDP.
   Tiempo promedio de transmisión: 10.90 segs (desviación de 124.529)

3. Protocolo PGMCOOP
   i. **2000 KB/s:** para esta tasa de transferencia inicial, el protocolo PGMCOOP logra 2311 KB/s en promedio. Un número promedio de NAKs de aproximadamente 400 por receptor hace que el CC se deba ejecutar mas de 30 veces en total y la tasa de transferencia no puede crecer demasiado. La desviación es bastante alta.
   Tiempo promedio de transmisión: 20.58 segs (desviación de 251.078)
   ii. **4000 KB/s:** usando 4000 KB/s como tasa inicial, se lograron, en promedio, 4056 KB/s. Es decir que el protocolo casi no pudo completar ventanas de transmisión sin que la tasa de NAKs dispare el control de congestión. Hubo en promedio aproximadamente 3000 NAKs por receptor. Extrañamente, la desviación estándar fue menor a los casos anterior.
   Tiempo promedio de transmisión: 11.73 segs (desviación de 143.338)
   iii. **6000 KB/s:** al aumentar mas todavía la tasa de transferencia, suben los NAKs de los receptores a un número de mas de 6800 en promedio. Esto hace que la tasa final lograda sea menor a la inicial: 5731 KB/s y en ningún caso se pudo subir la misma a mas de 6000. De todas formas terminan siendo las mejores tasas logradas en todas las pruebas de este escenario.
   Tiempo promedio de transmisión: 8.30 segs (desviación de 163.306)

Escenario E: Sucursal 36 – 100 Mb/s – 9 receptores

Para todas las pruebas se utilizó un padrón modelo de 48715120 bytes.

1. Protocolo MDP:
   i. **2000 KB/s:** de los 45 casos de estas pruebas, solo en 4 no se registraron bloques perdidos. en promedio fueron 577 bloques sobre los 23787 que se transmitieron, por lo que no impactó demasiado en los resultados. Se lograron 1886 KB/s con una desviación aceptable.
   Tiempo promedio de transmisión: 25.21 segs (desviación de 57.298)
   ii. **4000 KB/s:** en estas pruebas el promedio de bloques perdidos sube a 1226 que luego consumen, en promedio, 1.39 segs por receptor para ser retransmitidos. La tasa que se logra es de unos modestos 3234 KB/s y la desviación es realmente muy alta.
   Tiempo promedio de transmisión: 14.71 segs (desviación de 448.738)
   iii. **6000 KB/s:** al subir la tasa de transferencia a 6000 KB/s los receptores tienen picos de mas de 4000 bloques perdidos; en promedio se registraron 1763 con casi 2 segs de recuperación. Sorprende que en algunos casos hay receptores que no tuvieron pérdidas y pudieron registrar 5990 KB/s. La tasa promedio fue de 4054 KB/s con una desviación altísima de mas de 1MB/s.
   Tiempo promedio de transmisión: 11.73 segs (desviación de 1164.838)
2. Protocolo PGM
   i. **2000 KB/s:** PGM también registra retransmisiones, en promedio alrededor de 100. No es mucho, aunque la tasa de transferencia se ve impactada ya que se lograron en promedio 1646 KB/s.
   Tiempo promedio de transmisión: 28.89 segs (desviación de 40.499)
   ii. **4000 KB/s:** en estas pruebas PGM apenas logra registrar mas de 3 MB/s en un par de casos. En promedio termina dando 2971 KB/s con mas de 800 NAKs por receptor.
   Tiempo promedio de transmisión: 16.01 segs (desviación de 197.405)
   iii. **6000 KB/s**: tampoco se beneficia con una tasa de transferencia inicial mas alta. Los NAKs suben a un promedio de 1500 por cada receptor y esto impacta en una tasa de transferencia muy baja: solo 3629 KB/s con una desviación muy elevada.
   Tiempo promedio de transmisión: 13.11 segs (desviación de 454.846)
3. Protocolo PGMCOOP
   i. **2000 KB/s:** En estas pruebas el promedio fue de 2323 KB/s; supera a MDP pero no es un rendimiento demasiado mejor. Cabe aclarar que la desviación estándar fue muy alta.
   Tiempo promedio de transmisión: 20.47 segs (desviación de 311.964)

ii. **4000 KB/s:** al aumentar la tasa de transferencia, aumenta la cantidad de NAKs y los resultados sufren debido al ancho de banda que consumieron las retransmisiones. Solo se pudo obtener 3172 KB/s en promedio.
Tiempo promedio de transmisión: 14.99 segs (desviación de 101.191)

iii. **6000 KB/s:** transmitiendo al máximo inicial de lo experimentado en estas pruebas, el protocolo PGMCOOP logra los mejores resultados del escenario: 4392 KB/s de promedio y una desviación estándar de aproximadamente 187 KB/s. Hubo picos de mas de 11000 NAKs.
Tiempo promedio de transmisión: 10.83 segs (desviación de 187.824)

Escenario F: Sucursal 42 – 100 Mb/s – 20 receptores

Para todas las pruebas se utilizó un padrón modelo de 48715120 bytes.

1. Protocolo MDP:
   i. **2000 KB/s:** en las primeras pruebas de MDP en este escenario observamos que 59 de los 100 casos de prueba tuvieron bloques perdidos y en promedio fueron 378 bloques. La tasa de transferencia que se logró fue aceptable: 1904 KB/s. También observamos una considerable desviación estándar de casi 100 KB/s.
   Tiempo promedio de transmisión: 24.98 segs (desviación de 98.586)
   ii. **4000 KB/s:** en estas pruebas observamos 64% de los casos con bloques perdidos. El promedio de bloques perdidos se incrementa además a 893. Esto impacta negativamente en la tasa de transferencia lograda que solo es de 3162 KB/s a pesar de que algunos receptores lograron sostener los 3994 KB/s de máximo. Esto se ve reflejado en la altísima desviación de los resultados de este protocolo.
   Tiempo promedio de transmisión: 15.04 segs (desviación de 672.419)
   iii. **6000 KB/s:** al incrementar la tasa de transferencia se fuerza todavía mas la situación con los bloques perdidos ya que un 71 % de los receptores experimentaron pérdidas. El promedio de bloques perdidos sube a 1168 con algunos casos de 3000 bloques o mas impactando tanto en los resultados de la tasa de transferencia lograda (solo poco mas de 4000 KB/s) como así también en la desviación mas alta observada en todas las pruebas de todos los escenarios.
   Tiempo promedio de transmisión: 11.88 segs (desviación de 1495.646)
2. Protocolo PGM
   i. **2000 KB/s:** los resultados logrados por PGM no se acercan a las cifras logradas por MDP. Con 1445 NAKs de promedio por receptor, la tasa de transferencia sufre mucho y solo se logran 1324 KB/s lo que resulta en mas de 10 segundos agregados al tiempo que logró MDP en las pruebas con la misma tasa de transferencia.
   Tiempo promedio de transmisión: 35.91 segs (desviación de 88.926)
   ii. **4000 KB/s:** la segunda prueba confirma lo observado en el caso anterior. En promedio, los receptores enviaron 2035 NAKs. El impacto sobre la tasa de transferencia resultante es directamente proporcional ya que solo se pudieron obtener 2286 KB/s. La desviación es considerable (casi un 10%).
   Tiempo promedio de transmisión: 20.81 segs (desviación de 222.246)
   iii. **6000 KB/s**: por último, para este protocolo, al aumentar la tasa a 6000 KB/s, los NAKs suben a 2478 por receptor. La tasa de transferencia resultante es de solo 3287 KB/s. En definitiva, en este escenario, fue el protocolo que obtuvo los peores resultados de los tres que se experimentaron.
   Tiempo promedio de transmisión: 14.47 segs (desviación de 80.457)
3. Protocolo PGMCOOP
   i. **2000 KB/s:** en estas pruebas, la estrategia implementada en el protocolo PGMCOOP apenas puede llegar a incrementar la tasa de transferencia a 4000 KB/s, pero inmediatamente debe ejecutar el control de congestión por una alta tasa de NAKs recibidos. En promedio se lograron 1752 KB/s con 1808 NAKs por receptor. No se pudo superar a MDP.
   Tiempo promedio de transmisión: 27.14 segs (desviación de 66.264)
   ii. **4000 KB/s:** en las pruebas con esta tasa de transferencia PGMCOOP supera a MDP. A pesar de tener 2284 NAKs por receptor, permite obtener una tasa promedio de 3450 KB/s con una desviación estándar mucho

menor, por lo tanto es un escenario favorable para este protocolo. El control de congestión se disparaba alrededor de los 5500 KB/s.
Tiempo promedio de transmisión: 13.79 segs (desviación de 141.112)

iii. **6000 KB/s:** finalmente para la tasa máxima de las experimentadas PGMCOOP también supera a MDP logrando el máximo de este escenario con 4474 KB/s y una desviación 3 veces menor que la de MDP. Los NAKs fueron 3462 por receptor y en estas pruebas el control de congestión se disparaba cerca de los 8000 KB/s.
Tiempo promedio de transmisión: 10.63 segs (desviación de 518.511)

Capítulo 9: Conclusiones

En este capítulo se realizarán las conclusiones del presente trabajo.

Tabla de resultados

Las cifras de las pruebas, que fueron analizadas en el capítulo anterior se sumarizan en la siguiente tabla, que servirá como base del análisis de las conclusiones. En dicha tabla se ingresaron los valores de tiempo, tasa de transferencia y desviación de la misma para los 3 protocolos evaluados en todos los escenarios de las pruebas (A, B, C, D, E y F). Para los escenarios de 10 Mb/s se detallan los números de las pruebas de 200, 400, 600 y 800 KB/s, mientras que para los de 100 Mb/s ser cargaron los datos de las pruebas de 2000, 4000 y 6000 KB/s. En la columna “R” se especifica la cantidad de receptores de cada caso. Para el protocolo MDP, además se incluyen los datos de bloques promedio que tuvieron que retransmitirse.

			MDP				**PGM**			**PGMCOOP**		
	BW	R	MISSING	TIME	RATE	DESV	TIME	RATE	DESV	TIME	RATE	DESV
A	200	3	0.00	54.17	169.83	0.42	54.06	170.18	0.38	26.37	348.87	5.73
	400	3	11.27	23.16	397.21	5.71	23.49	391.61	0.14	17.22	534.16	30.40
	600	3	185.47	17.39	529.07	73.42	18.87	487.46	29.58	16.11	571.22	48.00
	800	3	360.67	17.35	530.20	116.37	14.76	623.28	33.20	16.10	571.36	77.11
B	200	5	0.00	46.15	199.34	0.12	46.18	199.24	0.04	26.21	351.07	3.17
	400	5	22.00	23.37	393.65	10.29	23.47	391.99	7.61	16.93	543.27	31.96
	600	5	247.56	21.75	423.01	91.13	18.98	484.84	17.00	16.40	561.09	32.41
	800	5	273.12	19.12	481.17	99.93	15.04	611.80	27.74	16.25	566.08	42.72
C	200	8	0.00	45.43	202.52	3.98	45.71	201.25	4.59	24.90	369.52	11.61
	400	8	0.00	22.99	400.14	2.03	22.95	400.80	2.49	12.62	729.09	27.52
	600	8	0.00	15.29	601.88	3.95	15.36	599.11	1.88	9.69	949.43	68.43
	800	8	0.00	11.47	802.09	3.66	11.48	801.11	3.66	10.59	868.66	201.21
	BW	R	MISSING	TIME	RATE	DESV	TIME	RATE	DESV	TIME	RATE	DESV
D	2000	4	467.90	24.54	1938.63	53.05	25.01	1902.16	52.10	20.58	2311.83	251.08
	4000	4	780.60	13.08	3636.32	367.19	12.76	3728.81	107.08	11.73	4056.10	143.34
	6000	4	1015.35	9.51	5001.73	945.53	10.90	4364.59	124.53	8.30	5731.12	163.31
E	2000	9	577.13	25.21	1886.74	57.30	28.89	1646.65	40.50	20.47	2323.76	311.96
	4000	9	1226.44	14.71	3234.91	448.74	16.01	2971.85	197.40	14.99	3172.87	101.19
	6000	9	1763.40	11.73	4054.37	1164.84	13.11	3629.14	454.85	10.83	4392.17	187.82
F	2000	20	378.25	24.98	1904.54	98.59	35.91	1324.79	88.93	27.14	1752.83	66.26
	4000	20	893.73	15.04	3162.79	672.42	20.81	2286.17	222.25	13.79	3450.69	141.11
	6000	20	1168.76	11.88	4004.21	1495.65	14.47	3287.34	80.46	10.63	4474.23	518.51

Ilustración 16: Tabla general de resultados

Conclusiones generales

El comportamiento del protocolo MDP con respecto a la cantidad de bloques perdidos es proporcional al crecimiento de la tasa de transferencia. A medida que incrementamos la misma y nos acercamos al límite de potencia de procesamiento de los receptores, los bloques perdidos se incrementan.

Solo en el caso del escenario C no observamos bloques perdidos. Esto se debe a que un hardware mas potente en los receptores les permite sostener fácilmente la recepción de datos para tasas de menos de 10Mb/s. Por lo tanto, este es un escenario óptimo para este protocolo y es el único donde los valores de desviación estándar son bajos.

En los escenarios de 10 Mb/s que tienen receptores con hardware mas viejo y limitado, no es posible superar los 500/600 KB/s. Como ya vimos, el rendimiento es muchísimo mejor con receptores que tienen mejor hardware.

En base a estas premisas, concluimos que para el caso de 10Mb/s el cuello de botella de las pruebas no es la red. Pasa lo mismo en los 3 protocolos. El hardware de los receptores es determinante para los resultados de las pruebas.

En las pruebas de 10 Mb/s, el protocolo PGM, en general, iguala o supera a MDP, seguramente debido a su estrategia de NAKs contra retransmisión individual de bloques perdidos. Este comportamiento se invierte en los escenarios de 100 Mb/s donde PGM es superado por MDP en casi todos los casos, pero con una observación importante: MDP tiene desviaciones estándar mas altas que PGM, lo que lo hacen un protocolo menos predecible en sus resultados.

En los escenarios de 100 Mb/s observamos que, por supuesto, a menos cantidad de receptores, el rendimiento es mejor. Esto se observa en todos los protocolos experimentados. Además, no se observa mucha diferencia en los resultados entre 9 y 20 receptores por lo que los protocolos no parecen tener dificultades en su escalabilidad. De todas maneras no fue posible en el presente trabajo experimentar con escenarios de mas de 20 receptores, por lo que no podemos asegurarlo.

Conclusiones acerca de PGMCOOP

En principio, el análisis de los resultados del protocolo experimental desarrollado en el presente trabajo puede considerarse positivo.
Como vemos en los gŕaficos comparativos de cada escenario, en la mayoría de las pruebas realizadas, PGMCOOP fue el protocolo que logró las mejores tasas de transferencia.

Las pruebas se realizaron utilizando parámetros de crecimiento de la tasa de transferencia y tamaños de ventanas muy conservadores. Por eso, por ejemplo para tasas iniciales de 200 KB/s, solo se pudo lograr en promedio alrededor de 320 KB/s, pero podrían utilizarse parámetros que permitan un comportamiento mucho mas agresivo del protocolo.
Los mejores rendimientos de PGMCOOP son utilizando como tasa de transferencia inicial el 50% del ancho de banda disponible en el segmento de red. De hecho cuando en los escenarios A y B, de 10 Mb/s, se forzó una tasa de transferencia inicial de 800 KB/s y los receptores no tuvieron capacidad suficiente, PGM tuvo mejores resultados que PGMCOOP.

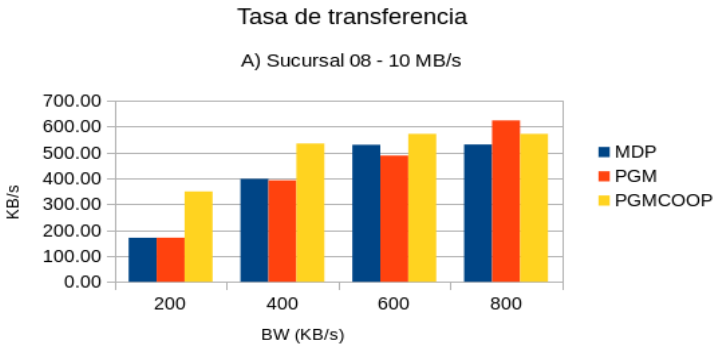


Ilustración 17: Resultados de pruebas en Sucursal 08

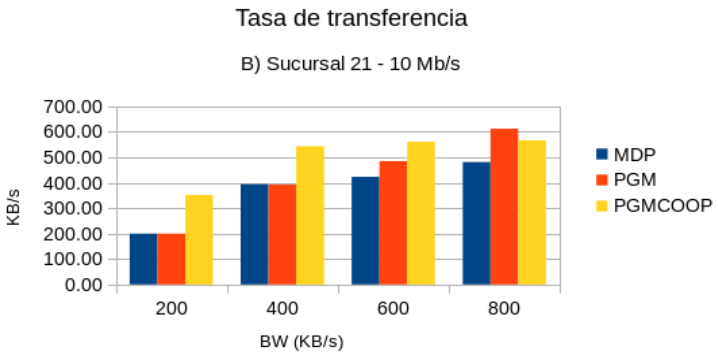


Ilustración 18: Resultados de pruebas en Sucursal 21

Este comportamiento se produce porque la estrategia de aumentar la tasa de transferencia termina siendo contraproducente para estos casos donde la misma está cerca del límite de capacidad de los receptores. Con la estrategia implementada, el protocolo pierde un intervalo de tiempo inicial hasta que se dispara el control de congestión, se disminuye la tasa de transferencia y el envío se estabiliza. Pero este proceso tiene un impacto en los resultados, suficiente para que un envío estable usando PGM a 800 KB/s termine dando mejores resultados.

En el escenario C, PGMCOOP claramente fue el protocolo de mejores resultados, en 200 y 400 KB/s casi duplicando las tasa de transferencias de MDP y PGM y en 600 KB/s logrando un 50 % mas. De estos resultados se puede concluir que resulta adecuada la estrategia de usar ventanas de transferencia para observar la tasa de NAKs que ingresan y subir la tasa de transferencia mientras que la misma no supere el umbral configurado.

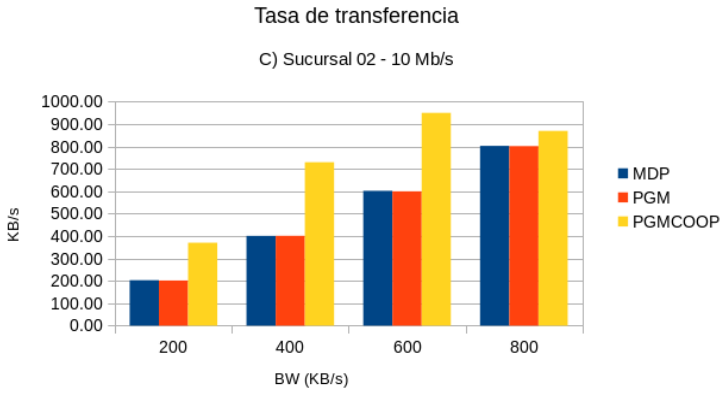


Ilustración 19: Resultados de pruebas en Sucursal 02

En la mayoría de los casos de prueba de los escenarios de 100 Mb/s, también es el protocolo que logró los mejores resultados, pero no supera a los otros 2 protocolos tan claramente como en los casos anteriores. Al utilizar tasas de transferencia iniciales mas exigentes, se llega mas rápidamente a los límites de capacidad de los receptores. Se observaron altas cantidad de NAKs para estos escenarios.

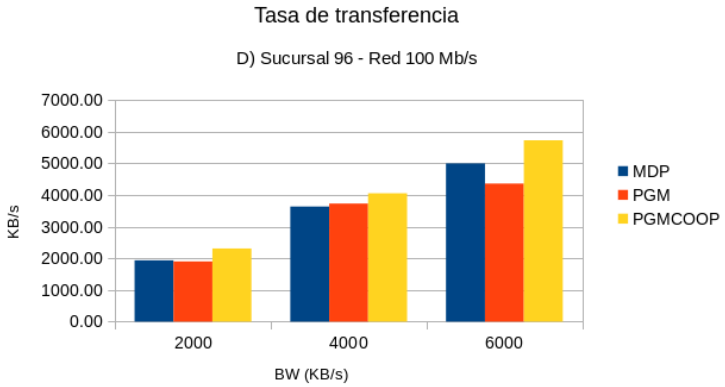


Ilustración 20: Resultados de pruebas en Sucursal 96

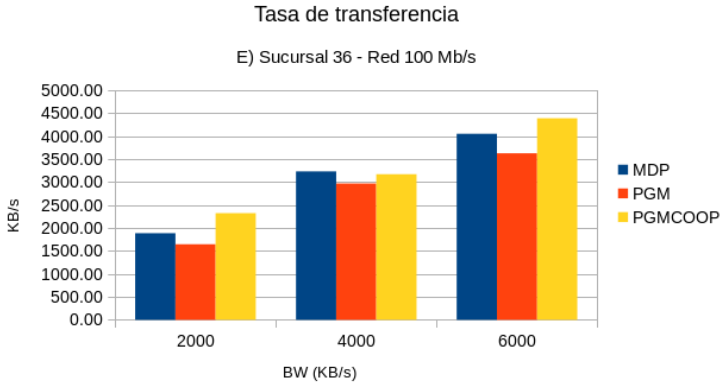


Ilustración 21: Resultados de pruebas en Sucursal 36

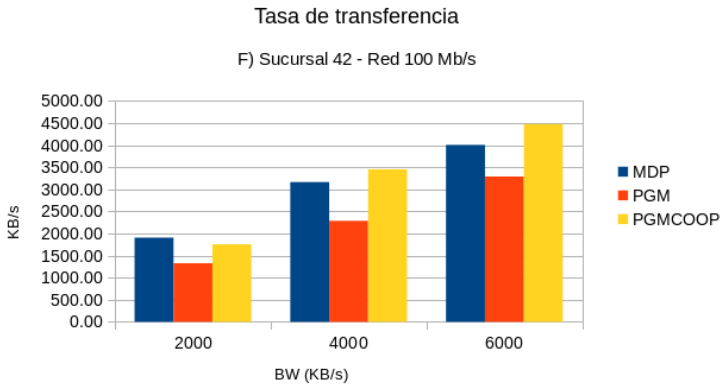


Ilustración 22: Resultados de pruebas en Sucursal 42

Inconvenientes del protocolo PGMCOOP

Estos casos nos permiten puntualizar un inconveniente importante de la estrategia del protocolo: en condiciones donde la red se encuentra en una situación de congestión, ya sea por tráfico propio del protocolo o cualquier otro tipo de tráfico que sature el segmento, puede ocurrir que los NAKs que los receptores están generando se pierdan. Esto tiene dos consecuencias en orden de importancia:

1. El transmisor no recibirá los NAKs y no se enterará de las pérdidas de bloques de los receptores, por lo cual no disparará su control de congestión y no disminuirá la tasa de transferencia
2. Los receptores generarán retransmisiones de esos NAKs, lo que genera todavía mas tráfico

Estas dos cuestiones harán que la situación empeore y si, desafortunadamente, el control de congestión no se ejecutara, es probable que los receptores lleguen a la condición del protocolo PGM de pérdida

irrecuperable de datos, donde el transmisor no tenga dentro de su ventana los datos necesarios para contestar con una trama RDATA un NAK recibido y así reparar la pérdida.

Esta condición indeseable del protocolo debe ser evitada, lo cual puede hacerse utilizando parámetros mas conservadores de transmisión y crecimiento de la ventana y de la tasa de transferencia.
De todas formas, el usuario del protocolo debe preveer un mecanismo de retransmisión / reintento del envío a nivel aplicación para los casos en los que existan receptores que experimenten la condición de pérdida irrecuperable de datos.

Análisis del protocolo en función de los objetivos planteados

El objetivo del presente trabajo era diseñar e implementar un protocolo de transporte para transmisión de datos “bulk” multicast **confiable** a utilizar en **ambientes de redes LAN** con estructura uno a muchos donde se poseen múltiples estaciones cuyos datos deben ser equivalentes. Dicho protocolo, debía contener mecanismos de control de congestión parametrizables y configurarse dinámicamente para optimizar su funcionamiento y adaptarse a diferentes escenarios de comunicación, enlaces físicos, velocidad de transmisión, throughput, buffers del receptor, etc., para lograr la mejor eficiencia posible en la transmisión. En particular, minimizar los tiempos de envío, minimizar la generación de tramas NAK de PGM, de ocupación de la red para no saturar la misma y permitir el libre flujo de otras transmisiones, maximizar el throughput sin saturar los receptores para minimizar las retransmisiones y dotar a las estaciones de mecanismos de feedback para notificar al emisor de información que le permita dinámicamente modificar los parámetros de envío y gradualmente mejorar la performance del protocolo.

Como hemos desarrollado en detalle, el protocolo PGMCOOP posee un mecanismo de control de congestión configurable mediante los parámetros cwnd, cc_cwnd, nak_th, r, etc. y su estrategia busca ser lo mas simple posible utilizando solamente los NAKs que los receptores transmiten ante las pérdidas de paquetes.

Las pruebas que se realizaron, permiten afirmar que el protocolo se adaptó bien a distintos escenarios de comunicación, variando segmentos de red y capacidades de los receptores y en la mayoría de los casos logró los mejores resultados, minimizando los tiempos de envío, y en los casos donde no fue el mejor, sus resultados de todas formas fueron aceptables.

La implementacion de esta estrategia de transmisión y de control de congestión requiere solamente modificar la lógica del transmisor. Los receptores no se modifican, por lo que el código de los mismos sirve tanto para el protocolo PGM como para PGMCOOP. Esto permite mantener su simplicidad de diseño e implementación.

Posibilidades de investigación y/o mejora

Como posibilidad de mejora o investigación adicional al presente trabajo se propone: almacenar

resultados y estadísticas de transmisiones anteriores y definir fórmulas y/o estrategias para optimizar dinámicamente los parámetros de las transmisiones sucesivas aprovechando el conocimiento del rendimiento de las anteriores.

Para esto se puede utilizar información detallada de la ejecución del protocolo que el código del transmisor implementado registra. Esta información fue muy útil para efectuar diagnósticos y ajustes en el mismo. El registro se guarda en un archivo de texto para posteriormente analizar y estudiar el comportamiento. Un ejemplo de la información detallada es el siguiente:

```
PGMCOOP using: cwnd=25,cc_wnd=5,a=1024,b=0.96,th=1250000
+ Adjusting max_rte to 458753 [avg: 408886 bps] cwnd=29, cc_cwnd=6 (wnd)
* NAK total: 0 - NAK rate: 0.00
+ Adjusting max_rte to 513804 [avg: 438006 bps] cwnd=33, cc_cwnd=7 (wnd)
* NAK total: 0 - NAK rate: 0.00
+ Adjusting max_rte to 575461 [avg: 469590 bps] cwnd=37, cc_cwnd=8 (wnd)
* NAK total: 0 - NAK rate: 0.00
+ Adjusting max_rte to 644517 [avg: 503808 bps] cwnd=42, cc_cwnd=9 (wnd)
* NAK total: 0 - NAK rate: 0.00
+ Adjusting max_rte to 721860 [avg: 541130 bps] cwnd=48, cc_cwnd=11 (wnd)
* NAK total: 2 - NAK rate: 1.20
* NAK rate threshold triggered: 6.10
* NAK total: 13 - NAK rate: 6.10
=> Congestion control started
- Adjusting max_rte to 692986 [avg: 572783 bps] cwnd=47, cc_cwnd=11 (cc)
# Usleeping for 29411. => Congestion control finished
* NAK rate threshold triggered: 5.14
* NAK total: 13 - NAK rate: 5.14
=> Congestion control started
- Adjusting max_rte to 665267 [avg: 587905 bps] cwnd=46, cc_cwnd=11 (cc)
# Usleeping for 30303. => Congestion control finished
```

Como posibilidad de mejora, se plantea almacenar esta información (ya sea en memoria o en almacenamiento persistente) para desarrollar una estrategia de configuración dinámica la cual podría analizar los valores de tasa de transferencia que dispararon controles de congestión para ajustar los parámetros necesarios para realizar mejores transferencias en el futuro.

Para el caso del ejemplo, el transmisor podría saber que puede comenzar directamente con un valor de tasa de transferencia de 644 KB/s, ya que dicha tasa de transferencia no había generado NAKs en los receptores. Otra optimización que se podría hacer, relacionada con el comienzo de la transmisión a un valor de tasa mas cercano al límite de capacidad de la red/receptores, es agrandar el tamaño de ventana y/o achicar el valor “a” (factor de crecimiento de la tasa) para no llegar tan rápidamente a 721 KB/s que fue el valor de tasa que comenzó a generar NAKs.

Con este criterio, luego se podría continuar estudiando y analizando los registros y estadísticas de las pruebas que se experimentaron en el presente trabajo para proponer mejoras que permitan al protocolo realizar ajustes dinámicos en su comportamiento y asi mejorar todavía mas los resultados logrados.

Bibliografía

- **BANIK00**: Banikazemi, Mohammad, IP Multicasting: Concepts, Algorithms, and Protocols, Año 1997, http://www.cis.ohio-state.edu/~jain/cis788-97/ip_multicast/index.htm
- **DAVIE00**: Davies, Joseph, Understanding IPv6, Microsoft Press, Año 2003 ISBN: 0-73-561245-5
- **HAGEN00**: Hagen, Silvia, IPv6 Essentials, 2nd Edition, O'Reilly Media, Año 2006 ISBN: 978-0-596-10058
- **SPURG00**: Spurgeon, Charles E., Ethernet - The Definitive Guide, O'Reilly Media, Año 2000 ISBN: 1-56592-660-9
- **STALL00**: Stallings, William, Data and Computer Communications, 5ta Edición, Prentice Hall, Año 1996, ISBN: 0-02-415425-3
- **SPEAK00**: Speakman, T., Crowcroft, J., Gemmell, J., Farinacci, D. , Lin, S., Leshchiner, D., Luby, M., Montgomery, T. , Rizzo, L., Tweedly, A., Bhaskar, N., Edmonstone, R., Sumanasekera, R., Vicisano, L., PGM Reliable Transport Protocol Specification, RFC 3208, Año 2001, https://tools.ietf.org/html/rfc3208
- **DIOT00**: Diot, Cristophe; Dabbous, Walid; Crowcroft Jon, Multipoint Communication: A Survey of Protocols, Functions and Mechanisms, IEEE, Año 1997, ISSN: 0733-8716
- **COMER00**: Comer, Douglas, Internetworking with TCP/IP, 4ta Edición, Prentice Hall, Año 2000, ISBN: 0-13-018380-6
- **FLOYD00**: Floyd, S., Jacobson, V., Liu, C., McCanne, S. y Zhang, L., A Reliable Multicast Framework for Light-weight Sessions and Aplication Level Framing. Publicado en Journal IEEE/ACM Transactions on Networking (TON) Volume 5 Issue 6, Año 1997 – Páginas 784-803
- **ADAMS00**: Adamson B., Bormann C., Handley M., Macker, J. , NACK-Oriented Reliable Multicast (NORM) Protocol Building Blocks, RFC 3941, Año 2004, https://tools.ietf.org/html/rfc3941
- **SCHROD00**: Carla Schroder, Measure Network Performance with iperf, Año 2007, http://www.enterprisenetworkingplanet.com/netos/article.php/3657236/Measure-Network-Performance-with-iperf.htm
- **TANEN00**: Tanenbaum, Andrew S., Computer Networks, 4ta Edición, Prentice Hall, Año 2003, ISBN: 0-13-066102-3
- **RIZZO00:** Rizzo, Luigi. pgmcc: a TCP-friendly single-rate multicast congestion control scheme. Dip. Ing. Informazione, Univ. di Pisa