GP- QoS

Garantía y Predicción de QoS en una Red de Distribución de Video

Proyecto final de carrera

Facultad de Ingeniería, Universidad de la República.

Integrantes: *Manuel Montaño, Diego Sanguinetti, Alejandro Sosa*

Tutor: *Pablo Belzarena*

Agradecimientos

A todos los que nos ayudaron directa o indirectamente. Desde los que nos aportaron algo de su conocimiento hasta los que nos prestaron su disponibilidad para ayudarnos.

A nuestro tutor Pablo Belzarena, el cual representó una guía fundamental a lo largo de todo el proyecto.

A nuestros amigos y compañeros por los piques, consejos, aguante y máquinas. Martín, Sebastián, Álvaro, Rosina y Joselo.

Especialmente a Inés, Mariana, María, Francesca y Mateo que nos bancaron en el día a día.

Tabla de Contenido

Resumen

El proyecto realizado presenta un sistema de control de admisión para una red de distribución de video sobre ip, con la intención de asegurar la calidad de servicio (QoS) requerida por el usuario.

Tal control de admisión asegura la calidad de los videos a lo largo de la duración de los mismos. Siempre priorizando la integridad de los videos en reproducción sobre los que se quieran reproducir en el futuro. O sea, cuando un nuevo video implique una degradación de la calidad de alguno de los que ya fluyen por la red, no se aceptará su solicitud.

Para poder brindar un control eficiente, el sistema hace uso de herramientas de estimación de parámetros de calidad de video que junto a criterios de decisión resultan en el filtrado de peticiones de los clientes.

Por lo tanto las dos principales funciones del sistema serían: estimación de QoS y control de admisión en base a dicha estimación.

Se comenzó trabajando en la segunda función mencionada del sistema, que consiste en el manejo de las peticiones de los clientes y la toma de decisión sobre su aceptación. Para ello se estudiaron programas Proxy conocidos y se buscó la manera de adaptarlos a nuestras necesidades. Esto resultó en la implementación de un Proxy capaz de escuchar peticiones de los clientes y luego consultar sobre los parámetros de QoS estimados para ese cliente.

Para la otra funcionalidad importante del sistema, que es la estimación de las calidades de servicio extremo a extremo entre servidor de video y cliente, el sistema se construyó a partir de herramientas ya utilizadas en otros proyectos. Se precisó, entonces, de una etapa de detallado estudio de trabajos previos sobre los algoritmos usados. En particular se estudió un método de aprendizaje estadístico que permite estimar los parámetros de QoS mediante métodos no intrusivos. En primera instancia, para conocer más cabalmente la herramienta, se realizaron simulaciones propias del algoritmo usado.
Una vez hecho esto, se implementaron los algoritmos en redes ip reales para testear su funcionamiento.

Cada de una de las funcionalidades se separaron en dos módulos autónomos que intercambian la información requerida para el control de acceso.

Luego se integraron ambos módulos en un software que automatiza el funcionamiento global del sistema. Esta etapa implicó necesariamente la realización de pruebas de campo para ir ajustando iterativamente el funcionamiento del sistema.

I-Presentación

1. Introducción

1.1 Motivación y planteamiento del problema

La masificación del acceso a la banda ancha y el aumento constante que ésta tiene, ha producido una demanda de las aplicaciones multimedia que consumen más recursos de las distintas redes, sea en Internet, redes dedicadas, o telefonía celular. Por lo que si bien los anchos de banda contratados son cada vez más baratos, los recursos disponibles en las distintas redes son finitos y no están diseñados originalmente para asegurar calidades de servicio (QoS) con exigencias en constante crecimiento.

Los ISP intentan amortizar el costo de su infraestructura multiplexando distintos tipos de servicio y la mayor cantidad de clientes posibles. Entonces, si bien los usuarios demandan una calidad cada vez más exigente para las distintas aplicaciones, siempre existe un compromiso con los recursos que tiene el administrador de la red para poder brindar la calidad deseada. Así es que el usuario aspirará obtener una QoS de las aplicaciones cada vez más demandantes en cuanto a la disponibilidad de recursos y lo comparará con la calidad que experimenta subjetivamente (QoE) [1].

Dicho lo anterior, y teniendo en cuenta que una de las aplicaciones más atractivas para los usuarios son las aplicaciones multimedia, surge la necesidad de poder respetar los Service Level Agreement (SLA)[1] que incluyan requerimientos de QoS.
De las aplicaciones multimedia en pleno crecimiento y que entran en lo antes mencionado, las relacionadas con el video son las que más se han desarrollado. Entre ellas, los servicios que tienen alguna interactividad con el usuario (el caso de VoD con control remoto virtual) o servicios de streaming en vivo (Live Streaming) tienen un campo especialmente fértil ya que están encaminadas hacia la mejora de la experiencia del cliente. Sin embargo, el problema de los proveedores de garantizar QoS ha impedido un crecimiento aún más acentuado.

El proyecto que se presenta, surge para atender la necesidad de poder garantizar la QoS extremo a extremo, y fue pensado para un escenario concreto en el que se establecieron pautas para acotar el alcance del problema.
Específicamente el sistema que se construyó se aplicaría en una red de distribución de video donde los usuarios harían peticiones para poder acceder a los videos y nuestro sistema ejercería precisamente el control de admisión a dicho contenido, estimando si el estado de la red permite garantizar la calidad de la nueva transmisión.

[1] Service Level Agreement: parte del contrato de servicio donde se define formalmente el nivel del mismo.

1.2 Objetivos del proyecto

Para la necesidad planteada de poder garantizar QoS, se propuso crear un sistema de control que limite el acceso del usuario a un contenido cualquiera o bien se le dé distinto grado de prioridad. Con ello el proveedor de servicios logrará no comprometer la calidad de servicio que se le garantizó a un usuario al momento de aceptar su solicitud.

Desde el inicio se ideó un sistema con los siguientes componentes:
- un módulo estadístico (M.E) que medirá el estado de la conexión extremo a extremo con cada cliente y predecirá si es posible respetar la QoS solicitada.
-un Proxy que acepte o no la petición en base a la información provista por el M.E.

La figura nro.1 muestra como estaría formado el sistema incluido en la red de distribución.

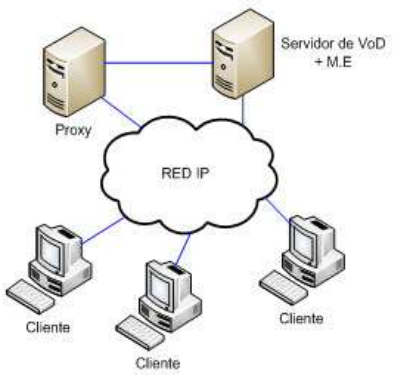


Figura 1

Antes de comenzar los trabajos se acordó que, de haber montado una red con un servidor de video, 3 o más clientes incluidos en la red de distribución y el sistema diseñado, se debiera verificar lo siguiente:

- Que el sistema permita el acceso al cliente cuando cumple los requerimientos de calidad extremo a extremo (del servidor de video al cliente).

- Que el sistema no permita el acceso al cliente cuando no cumple los requerimientos.

- Que se cumplan los parámetros de calidad una vez que se permita el acceso a un cliente.

También se fijó como objetivo que la arquitectura del sistema permita escalabilidad y que podamos reutilizar el sistema para diferentes tipos de módulos estadísticos y distintos tipos de servicio.

1.3 Estado del arte

A la hora de garantizar la QoS se han intentado aplicar muy diversas estrategias. Posibles ejemplos aplicados por proveedores de Internet son la reserva de recursos extremo a extremo para cada flujo de datos o el establecimiento de prioridades según el tipo de servicio.

Otro enfoque posible para asegurar QoS, es proveer servicios en una red privada controlada por el proveedor e implementar un control de admisión de los usuarios.
El sistema está basado en este último enfoque, realizando un control de acceso de los usuarios del sistema en base a mediciones de parámetros de red que estimen la QoS.
En el comienzo del proyecto se accedió a información sobre estimación de calidad de servicio en redes ip. Se vieron distintos algoritmos como la regresión no paramétrica utilizando la distribución empírica del tiempo entre paquetes [2] y también utilizando el estimador de cola de extremo a extremo [3], ambos para caracterizar el estado de la red.

Como el sistema utilizará el último de los algoritmos antes dicho, un estudio de las experiencias con el mismo fue conveniente.
Una de esas fuentes fue el proyecto Metronet [4] que desarrolla una metodología y un software para medir los diferentes parámetros de QoS en aplicaciones de voz y video. El software Metronet es utilizado como una aplicación para el cliente y para el operador de la red, a la hora de realizar mediciones pero no realiza ningún tipo de control de admisión.
Se estudiaron estas diversas herramientas adaptándolas a nuestro sistema de control de admisión.

1.4 Escenarios de aplicación

La motivación del proyecto surgió desde el principio con las hipótesis que se detallan en la sección 4.2 (Hipótesis asumidas) sobre el control que se tendrá sobre la red. Sin embargo con la intención de fijar ideas sobre el funcionamiento del sistema, se decidió plantear algunos escenarios de aplicación del mismo

1.4.1 Aplicación en redes punto-multipunto

Un escenario posible de aplicación puede ser el caso de una empresa que contrata una red privada a un proveedor de servicios de telecomunicaciones, para poder brindar servicios de video sobre ip.
Tal empresa contrata enlaces punto-multipunto con un puerto central, y luego enlaces dedicados de un ancho de banda acorde a las necesidades.
Seguramente el puerto central esté sub-dimensionado por una razón de costos, por lo que el administrador de la red (quien contrata la red privada) debe controlar la calidad de servicio por sus propios medios.

La empresa obtiene un enlace transparente de cierta capacidad y podrá utilizar este sistema para garantizar la QoS a sus clientes ubicados en los extremos de cada punto a

punto.
Se aclara que en este caso particular, la red es controlada por el administrador pero la degradación de la calidad de los enlaces se puede dar por otros factores además de la congestión. Por ejemplo un equipo intermedio defectuoso o problemas en la capa física donde el administrador no tiene influencia salvo por el SLA acordado con su proveedor.

1.4.2 Aplicación en redes de un ISP

Este es el caso de un proveedor de servicios de internet que cuenta con el valor agregado de ofrecer a sus clientes el acceso a una red de video con recursos dedicados.
Aquí el punto de congestión no se encuentra en el backbone que seguramente esté sobre-dimensionado, sino que probablemente esté en el acceso donde se multiplexan servicios de muchos clientes.

1.5 Esquema del documento

Capitulo 2 - Calidad de servicio para distribución de video

En este capítulo se presentan en líneas generales los métodos más utilizados para la codificación de video y se analizan los parámetros de la red que afectan a la calidad percibida de un video, como la afectan y sus relevancias dependiendo de la aplicación.

C*apitulo 3- Estimación QoS mediante aprendizaje estadístico*

Se muestran aquí los fundamentos teóricos de las herramientas utilizadas para la estimación de los parámetros de QoS presentados en el capitulo anterior. Se muestra también una metodología que permite dicha estimación utilizando métodos no intrusivos una vez completada una etapa de aprendizaje estadístico.

Capitulo 4 - Descripción global
Capitulo 5 - Desarrollo y arquitectura del sistema

En estos capítulos se presenta la arquitectura del sistema, se esquematiza el funcionamiento del mismo y se detallan las funcionalidades de los elementos que lo componen.
Además se analizan las hipótesis asumidas a lo largo del trabajo.

Capitulo 6 - Implementación del Módulo Estadístico
Capitulo 7 - Implementación del Proxy
Capitulo 8 - Interacción entre módulos

En estos capítulos se presentan los criterios y algoritmos utilizados en la implementación del módulo estadístico (M.E.) para la estimación de los parámetros de QoS relevantes. Y los del Proxy para decidir cuándo se puede permitir el acceso a un nuevo cliente en base a consultas realizadas al M.E.

Se definen también las interfaces a utilizar entre los distintos elementos para lograr el

sistema integrado, capaz de realizar el control de acceso deseado.

Capitulo 9 - Estudio de valores críticos de parámetros de calidad

En este capítulo se realiza una descripción general de los métodos de medición subjetivos y objetivos de calidad de un video.
En particular se describen los métodos de medición subjetivos según la recomendación ITU-R BT.500-11 y se explica el método utilizado en nuestro caso.

Capitulo 10 – Simulaciones
Capitulo 11 - Pruebas realizadas y resultados obtenidos

En estos capítulos se analizan por un lado las simulaciones realizadas utilizando la herramienta NS (Network Simulator) y por otro lado las pruebas y resultados obtenidos para una red de prueba real.

Capitulo 12 - Conclusiones
Capitulo 13 – Trabajo a futuro

Aquí se realizan las conclusiones finales del proyecto, se analizan posibles mejoras al sistema y se presentan distintos escenarios en los que nuestro sistema podría ser reutilizado.

II-Fundamentos

2. Calidad de servicio para distribución de video

2.1 Descripción de codecs para vídeo

El objetivo de la codificación de video es remover la información redundante de manera de representar el video en un formato más compacto.
La redundancia podrá ser espacial (en un mismo cuadro), temporal (entre cuadros consecutivos), estadística o perceptual (tomando en cuenta la capacidad de percepción humana).
Para compensar la redundancia espacial y estadística se suelen usar la transformada discreta de coseno (DCT) y la codificación entrópica.
Técnicas de compensación de movimiento (MC) son utilizadas para la redundancia temporal. Mediante esquemas de cuantificación se controla el factor de compresión, eliminando la información menos relevante según la percepción humana.

Desde la década del '80 se han estado lanzando recomendaciones respecto a algoritmos de codificación de imagen y video por parte de organizaciones como la ITU (International Standarization Union) y la ISO (International Standarization Organization).
El primer estándar de codificación de imagen fue el JPEG (Joint Picture Experts Group) para imágenes estáticas y MPEG-1 para almacenamiento audiovisual en CD-ROM. A lo largo de los años fueron surgiendo nuevas recomendaciones orientadas a distintas aplicaciones (por ejemplo: MPEG-2 para broadcast de Tv digital, ITU-H.263 para videoconferencia). Estos nuevos estándares fueron mejorando la eficiencia de compresión (igual calidad a bitrates menores) a expensas de tener algoritmos más complejos con mayor costo computacional.

Los algoritmos MPEG actuales utilizan una combinación de técnicas de compresión aplicando la transformada DCT para reducir redundancias espaciales dentro de cada cuadro y MC (estimación y compensación de movimiento) para reducir redundancias temporales existentes entre cuadros.
En primera instancia dividen la secuencia de video en GOP's (Groups of Pictures o Grupos de Imágenes) que pueden tener tres tipos de cuadros: I, P, y B. (Ver **Figura 2** extraída de [5])

Los cuadros I son los que llevan la mayor cantidad de información, siendo solamente comprimidos mediante la transformada DCT eliminándose componentes de alta frecuencia (poco perceptibles por la visión humana).

Estos cuadros son utilizados como referencia para los cuadros P (predictivos hacia adelante) y B (predictivos hacia adelante y hacia atrás).
Los cuadros P son predichos en función de la información del cuadro I o eventualmente de cuadros P anteriores mediante técnicas de estimación y compensación de movimiento. Los cuadros B son predichos en función del cuadro I o P anterior y

posterior.

La estructura del GOP quedara determinada entonces por dos cuadros I y sus cuadros P y B intermedios y esta se repetirá a lo largo de la secuencia de video.
En la **Figura 2** se muestra ejemplos de un GOP's con distintas estructuras y las dependencias entre cuadros I, B, P.

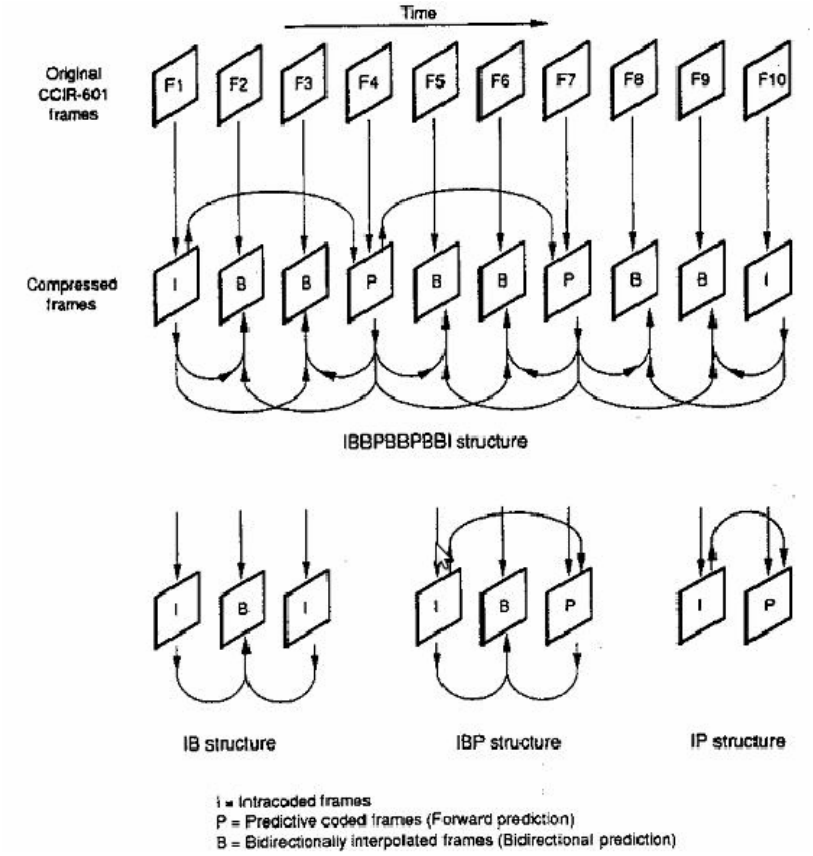


Figura 2

En general los cuadros I tendrán baja compresión, los cuadros P tendrán una compresión media y los cuadros B tendrán el mayor nivel de compresión. En consecuencia, mientras más grande sea el GOP se podrán conseguir bitrates menores aunque también habrá mayor vulnerabilidad frente a la propagación de errores.

La estimación de movimiento se realiza dividiendo el cuadro en macro-bloques de 16x16 píxeles, para luego estimar el movimiento de cada macro-bloque comparándolo con todas las secciones posibles de igual tamaño en el cuadro siguiente y buscar la que sea más parecida (se busca el mínimo error cuadrático medio). Entonces se codifican las diferencias entre los macro-bloques predichos con los del cuadro original y los vectores de desplazamiento de cada macro-bloque (se asume que todos los pixeles dentro del macro-bloque tienen el mismo desplazamiento).
Como resultado, cuando hay alta correlación entre cuadros consecutivos las diferencias serán más pequeñas y se obtendrán bitrates menores. Pero por ejemplo en escenas con alto nivel de movimiento habrá diferencias mayores entre cuadros y el bitrate obtenido será mayor.

Por otra parte, una vez aplicada la compensación de movimiento, se aplican códigos de Huffman (codificación entrópica) para aprovechar la redundancia estadística. Esto es, se codifican los símbolos más frecuentes con códigos más cortos (menos bits) y los menos frecuentes con códigos más largos (más bits).

Si comparamos el bitrate para una codificación MPEG-1 con MPEG-2, tendremos que en el primer caso la calidad y tasa de compresión varían para obtener una salida CBR mientras que en el segundo caso el bitrate y la tasa de compresión variaran para conseguir un nivel de calidad constante.

MPEG-4, es el estándar sucesor de MPEG 1 y 2 presentando mejoras de eficiencia de compresión y robustez frente a errores. En la actualidad la versión más madura es el estándar ITU H264/MPEG4 AVC el cual es superior al resto de los algoritmos de codificación para la mayoría de los escenarios de aplicación.

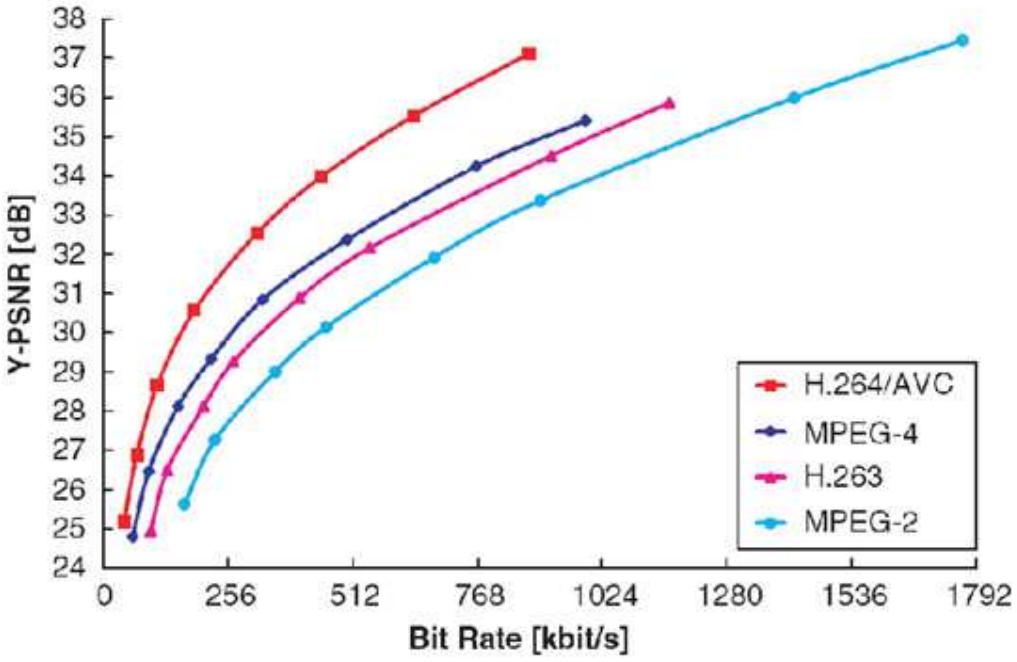


Figura 3

En la gráfica de la **Figura 3** [5] se muestra las distintas calidades obtenidas al variar el bitrate para distintos estándares de compresión.
El algoritmo de codificación H264 se destaca por su eficiencia de compresión para bitrates altos, utilizando tan solo el 50% del bitrate respecto a su predecesor MPEG-2 para obtener un mismo nivel de calidad. Esto hace que sea el codec mas usado para contenidos de alta definición en donde el ahorro de ancho de banda es crucial.

2.2 Elección de codecs de videos

Para la elección de los codecs de video a utilizar se tuvo en cuenta además de la eficiencia de compresión, la limitación en el sistema respecto a la simulación de secuencias de video a altas tasas de codificación como se explicara en la sección 6.4.3.
Por estos motivos, se decidió utilizar únicamente el codec H.264/AVC ya que esto nos permitió minimizar el ancho de banda requerido para la transmisión de los videos.

2.3 Parámetros de calidad de video en tiempo real

2.3.1 Generalidades

Existen varios factores que pueden afectar la calidad de video percibida por un usuario. Estos se pueden clasificar básicamente en parámetros de la red como ser el retardo, jitter, tasa de pérdidas de paquetes, perdidas de paquetes en ráfaga y en parámetros relativos a la aplicación utilizada (método de compresión, algoritmos de recuperación frente a perdida de información, etc.) y al tipo de contenido.
En nuestro caso, nos enfocaremos en el estudio de los parámetros de la red que afectan a la calidad de video.

2.3.2 Retardo

El efecto de los altos retardos de los paquetes IP tendrán mayor efecto en la calidad percibida por los usuarios en aquellas aplicaciones en la que haya involucrada interactividad o aplicaciones en tiempo real. Videoconferencias y transmisiones en vivo son ejemplos de este tipo de aplicaciones.
Según la ITU [6] se establece que el retardo de ida entre dos puntos no debe superar los 150 ms para que no perjudique la interacción en una conversación.

En otro tipo de aplicaciones como transmisión en vivo, por ejemplo de un evento deportivo el retardo podrá ser mayor pero podría ser perjudicial si sobrepasara unos pocos segundos.
Un posible ejemplo podría ser la transmisión de un partido de fútbol en alta definición.

En estos casos, debido a que las tasas de codificación son elevadas, posiblemente para su transmisión se use el codec H.264 para minimizar el consumo de recursos. Pero al tener un algoritmo de compresión mucho más complejo puede llegar a introducir un retardo de aproximadamente 4 - 5 segundos más que si se hubiera codificado con MPEG-2, lo cual puede ser no deseable.

En el caso de aplicaciones como VoD, donde no hay interactividad (salvo el control remoto virtual para pausar, rebobinar, etc.), el retardo no afecta a la calidad percibida.

En general, los elementos que contribuyen al retardo son: la etapa de codificación, encolamientos en los buffers de los enlaces, latencias de los enlaces y la decodificación en el destino. La figura 4 extraída de [7] ilustra las etapas de un proceso de transmisión de video, donde en cada etapa habrá una contribución a la demora en llegar la

información hasta el receptor y poder ser desplegada al usuario.

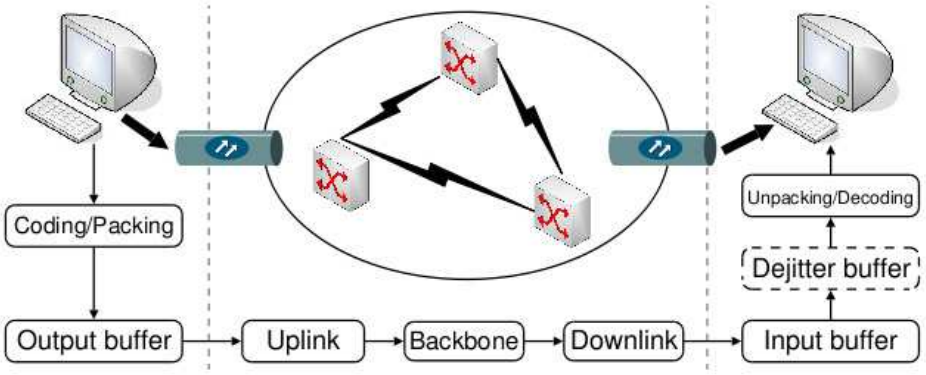


Figura 4 - Esquema de transmisión de video

2.3.3 Jitter

Otro parámetro de la red que puede afectar la calidad percibida de un video es la variación del retardo, o también llamado jitter.
El jitter se produce por las variaciones de tamaño de las colas de los enlaces presentes en el camino del servidor al cliente.
El problema radica en que si el video sale a una determinada tasa de codificación, en recepción se debería recibir exactamente a la misma tasa para poder decodificar correctamente la información.
Sin embargo, las variaciones de retardo son inherentes a las redes IP y si hay paquetes que no llegan a tiempo al receptor se perderá información pudiendo pixelarse o congelarse la imagen.
Como el jitter no se puede evitar, es usual que en recepción se utilice un de-jitter buffer que suavice el efecto de la variabilidad del retardo. El de-jitter buffer recibirá los paquetes que llegarán en intervalos de tiempo variable, acumulándolos y entregándolos a tasa constante al decodificador.
Este método compensa el jitter a expensas de agregar retardo adicional (por la acumulación de paquetes en el buffer) lo cual puede llegar a ser contraproducente para aplicaciones en tiempo real.
El de-jitter buffer compensa valores de jitter bajos por lo que no es suficiente para evitar las consecuencias del jitters altos.

2.3.4 Pérdidas

La tasa de pérdida de paquetes también será un parámetro que refleje la degradación en la calidad percibida por un usuario. Evidentemente la información contenida en los

paquetes perdidos se perderá.
En la **Figura 5** extraída de [5] se muestra el efecto de la perdida de paquetes sobre una sucesión de cuadros. Se puede observar que dependiendo si los paquetes perdidos pertenecen a un cuadro de referencia (cuadro I) o no, la perdida de información puede solo afectar al cuadro al cual pertenecía el paquete o bien afectar también a los cuadros predictivos siguientes.

En la mayoría de los algoritmos de decodificación ante la pérdida de paquetes se intenta reconstruir los píxeles perdidos a partir de la información que si está disponible, por ejemplo sustituyendo los pixeles perdidos por los de los cuadros anteriores (útil para imágenes casi estáticas)

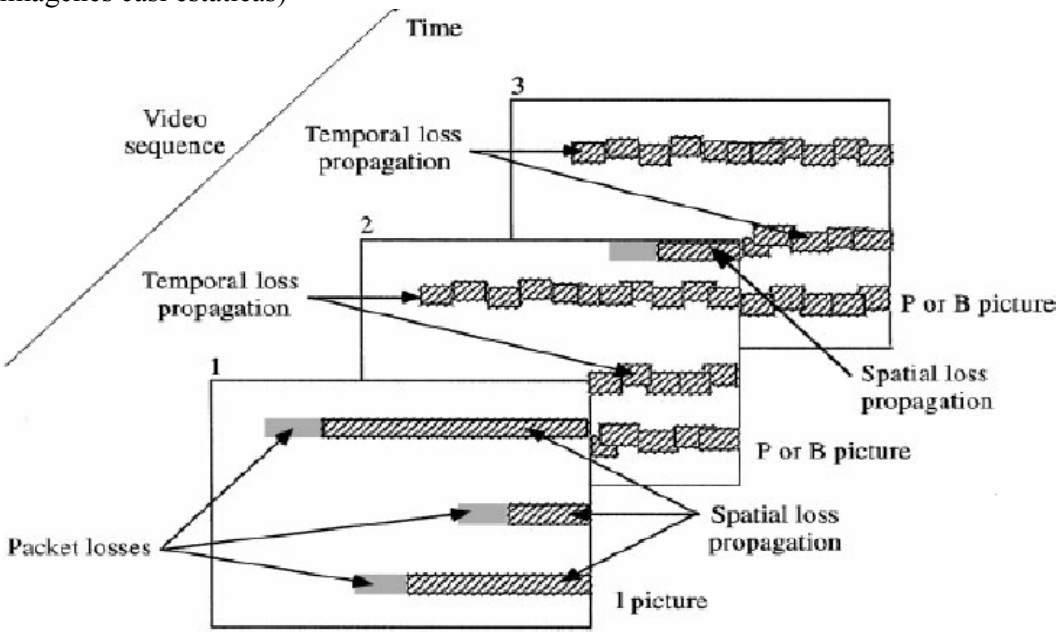


Figura 5

Se observa entonces que el nivel de degradación de calidad debido a la perdida de paquetes va a depender de qué tipo de información transporten, y también del patrón de perdidas; no tendrán el mismo efecto perdidas aisladas que en ráfagas ya que en el último caso afectara a varios cuadros consecutivos.

En este sentido, se podría proponer como otro posible parámetro de calidad, la longitud media de las ráfagas de pérdidas (mean loss burst length) para tener una mejor estimación de la calidad percibida. Por simplicidad se considera únicamente la tasa (promedio) de pérdidas.

3. Estimación de QoS mediante aprendizaje estadístico

3.1 Introducción

Con la intención de estudiar las distintas alternativas existentes a la hora de estimar la calidad de servicio, se presenta un enfoque estadístico conocido **[3]** para entenderlo y aplicarlo al problema concreto.

Para la estimación de QoS, más precisamente para la estimación de parámetros de desempeño como el jitter, retardo y tasa de pérdidas que determinan la QoS, se encuentran los métodos intrusivos y los no intrusivos. Un método intrusivo puede consistir en enviar efectivamente el tráfico de la aplicación durante el lapso de tiempo a medir y obtener directamente los valores de los parámetros de interés (retardo, jitter y pérdidas).
En el caso de una red de distribución de video, se envía un tráfico importante para cada medición. Esto podrá sobrecargar la red durante periodos de tiempo muy largos, lo que no es deseable.

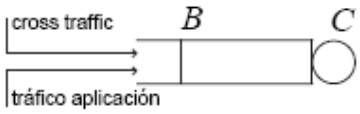
En cambio, un ejemplo de método no intrusivo es enviar paquetes de prueba (de ahora en más pp) livianos de extremo a extremo, del orden de las decenas de bytes y predecir los valores de los parámetros de interés a través de parámetros de los pp. Este método no inserta un tráfico importante en la red pero se basa en la suposición de que los pp son tratados de igual modo que los paquetes de video. Esto no es una buena aproximación ya que hay una diferencia considerable en tamaño y tiempos de envío de paquetes entre ambos tráficos.

Por este motivo se utilizó un método hibrido que combina los dos anteriores.
Este consta de enviar una secuencia de pp, seguido por una secuencia de paquetes de video de corta duración, de modo de no cargar demasiado el enlace, y medir directamente el parámetro de interés obtenido. De esta manera, si se aprende la relación entre el tratamiento que los pp reciben y los parámetros del video para cierto estado de la red, se puede estimar los parámetros tan solo enviando los pp sin necesidad de cargar la red.

Para encontrar dicha relación se utiliza el método de aprendizaje estadístico que, junto con sus dos etapas (de aprendizaje y de predicción), se explicarán a continuación.
Se presentará también el algoritmo SVM (Support Vector Machine [8]), utilizado para realizar la regresión, o sea, encontrar la función que relaciona los parámetros de pp y de QoS.

3.2 Modelo

La siguiente figura muestra un nodo de capacidad C y buffer o cola B con las principales características que tomaremos en cuenta en la estimación de los parámetros de calidad *Y*.



-CT_t es el tráfico cruzado de la red.
-V_t es el tráfico e la fuente, el cual es conocido.
-C es la capacidad del enlace y B es el tamaño del Buffer.

$$Y = F(CT_t, V_t, C, B) \qquad (1)$$

Bajo la suposición de que la capacidad del enlace C y el tamaño del buffer B no varían y el tráfico de la fuente es conocido, CT_t es la variable, ya que éste valor cambia dependiendo de la demanda de los usuarios.

Entonces $Y = F(CT_t)$.

Se plantean dos problemas, el primero es estimar el estado de la red X que depende del tráfico cruzado CT_t.
Y el segundo es estimar la función ϕ que la relacione con los parámetros de calidad de video Y ($Y = \phi(X)$).

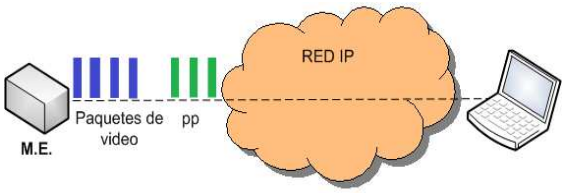
Para caracterizar el estado de la red X se envían los pp y se calcula el tiempo entre arribos de paquetes consecutivos, los cuales están fuertemente correlacionados con el volumen de tráfico cruzado.

A partir de esta información se halla una función de tiempos entre arribos que actúa como variable X.

3.3 Etapa de entrenamiento

Consiste en enviar una serie de paquetes de prueba con tiempo entre salida constante y se calcula el tiempo entre arribos de los pp.

Seguido, se envía un video de extremo a extremo, durante un lapso de tiempo, de modo de no cargar demasiado el enlace, y medir directamente el parámetro de interés obtenido.



La etapa de entrenamiento trata de hallar una correspondencia entre parámetros de interés y el tiempo entre arribos entre pp. Con el objetivo de hallar dicha correspondencia para distintos estados entre extremos se realiza este procedimiento para diferentes valores de tráfico cruzado sobre el enlace.

Para cada experimento *j* obtenemos (*Yj, Xj*). Con el conjunto de pares $(Y_1,X_1) \ldots (Y_n,X_n)$ se estima ϕ_n, un estimador de ϕ.

3.4 Etapa de predicción

Esta etapa consiste en predecir los parámetros de interés mediante el envío de pp utilizando la correspondencia determinada en la etapa de entrenamiento ϕ_n.

Se calcula la variable *X* con el tiempo entre arribos de pp, y se estima $Y=\phi_n(X)$.

Cuando un cliente realiza una nueva petición de video el sistema a través del envío de pp y midiendo el tiempo entre arribos, predice cuales serán los valores de los parámetros de calidad de extremo a extremo entre cliente y servidor VLC, lo cual se traslada a la calidad percibida por el usuario.

De esta forma la etapa de entrenamiento y la etapa de predicción determinan un método de aprendizaje estadístico preciso para determinar los parámetros de calidad de un video, con la particularidad de que no sobrecarga la red ya que utiliza pp livianos para la predicción.

3.5 Estimador de estado de la red

Para estimar el estado de la red (*X)* utilizamos el *Estimador del estado de la cola* [3] del enlace, el cual es presentado en el siguiente planteo:

Sea el paquete de prueba n que arriba a la cola del enlace en el tiempo t_n^i , sale de ella en el tiempo t_n^0 y C la capacidad del enlace, se cumple que:

$$q(t_n^i) + G = C\left(t_n^o - t_n^i\right)$$

Donde $G = P + DC$ es constante, siendo *D* la latencia fija del enlace y *P* el tamaño no variable de los paquetes de prueba. La anterior ecuación tiene la debilidad que se miden los tiempos en lugares diferentes y por ende con diferentes relojes, lo que implica que la

medida queda sujeta al sincronismo de los relojes. Buscando una ecuación que no dependa de dicho sincronismo, se restan las siguientes ecuaciones:

$$q(t_n^i) + G = C\left(t_n^o - t_n^i\right)$$
$$q(t_{n-1}^i) + G = C\left(t_{n-1}^o - t_{n-1}^i\right)$$

Resultando en,

$$q(t_n^i) + G - q(t_{n-1}^i) - G = C\left(t_n^o - t_n^i\right) - C\left(t_{n-1}^o - t_{n-1}^i\right)$$

Reordenando los términos,

$$q(t_n^i) = q(t_{n-1}^i) + C\left(\left(t_n^o - t_{n-1}^o\right) - \left(t_n^i - t_{n-1}^i\right)\right)$$

Asumiendo que hubo un instante t_0^i en que la cola estuvo vacía y aplicando en forma recurrente la anterior ecuación, se obtiene que el tamaño de la cola que ve el paquete de prueba n cuando arriba a la cola:

$$q_n = q(t_n^i) = C\sum_{j=1}^{n}\left[(t_j^0 - t_{j-1}^0) - (t_j^i - t_{j-1}^i)\right] \qquad (2)$$

Se define el estimador del estado de la cola, según la ecuación:

$$\hat{q}_n = \frac{q(t_n^i)}{C} = \sum_{j=1}^{n}\left[(t_j^0 - t_{j-1}^0) - (t_j^i - t_{j-1}^i)\right] \qquad (3)$$

que representa el tiempo que el paquete n queda esperando en la cola del enlace.

Se observa que para cada j se halla la diferencia entre tiempos de salida y tiempos de arribos, los tiempos de arribo son tomados por un equipo y los tiempos de salida por otro equipo, por lo tanto al restarse no se introduce incertidumbre por desincronización entre relojes.

Cuando de extremo a extremo participan más de un enlace, se plantea para el enlace *l* del camino la siguiente ecuación para el tamaño de la cola,

$$q_l(t_n^{i,l}) = C_l\sum_{j=1}^{n}\left(\Delta_{l+1,j} - \Delta_{l,j}\right) \quad \forall l \in \{1,\ldots..,\, N\} \qquad (4)$$

Donde,

$$\Delta_{l,j} = t_j^{i,l} - t_{j-1}^{i,l} = t_j^{0,l-1} - t_{j-1}^{0,l-1}$$

Considerando que:

$$t_j^{i,l} = t_j^{o,l-1} \quad \text{y} \quad t_{j-1}^{i,l} = t_{j-1}^{o,l-1}$$

Sumamos las colas,

$$\sum_{l=1}^{N} q_l(t_n^{i,l}) = \sum_{l=1}^{N} C_l \sum_{j=1}^{n} \left(\Delta_{l+1,j} - \Delta_{l,j}\right) \tag{5}$$

Lo que es lo mismo,

$$\sum_{l=1}^{N} q_l(t_n^{i,l}) = C_1 \sum_{j=1}^{n} \left(\Delta_{2,j}\right) - C_1 \sum_{j=1}^{n} \left(\Delta_{1,j}\right) + \ldots +$$

$$C_l \sum_{j=1}^{n} \left(\Delta_{l+1,j}\right) - C_l \sum_{j=1}^{n} \left(\Delta_{l,j}\right) + \ldots +$$

$$C_N \sum_{j=1}^{n} \left(\Delta_{N+1,j}\right) - C_N \sum_{j=1}^{n} \left(\Delta_{N,j}\right)$$

Si se suma a cada fila $C_N \sum_{j=1}^{n} \left(\Delta_{l,j}\right) - C_N \sum_{j=1}^{n} \left(\Delta_{l+1,j}\right)$ y a la última fila $C_N \sum_{j=1}^{n} \left(\Delta_{N,j}\right) - C_N \sum_{j=1}^{n} \left(\Delta_{1+1,j}\right)$, se observa que el total de la suma de los términos sumados da como resultado cero. Ya que el segundo término de cada fila se cancela con el primer término de la fila siguiente, y el segundo de la última se cancela con el primero de la primera fila. Obtenemos:

$$\sum_{l=1}^{N} q_l(t_n^{i,l}) = (C_1 - C_N) \sum_{j=1}^{n} \left(\Delta_{2,j}\right) - (C_1 - C_N) \sum_{j=1}^{n} \left(\Delta_{1,j}\right) + \ldots +$$

$$(C_1 - C_N) \sum_{j=1}^{n} \left(\Delta_{l+1,j}\right) - (C_1 - C_N) \sum_{j=1}^{n} \left(\Delta_{l,j}\right) + \ldots +$$

$$C_N \sum_{j=1}^{n} \left(\Delta_{N+1,j}\right) - C_N \sum_{j=1}^{n} \left(\Delta_{1,j}\right)$$

Teniendo en cuenta que $\frac{q_l(t_n^{i,l})}{C_l} = \sum_{j=1}^{n} \left(\Delta_{l+1,j} - \Delta_{l,j}\right)$ tenemos,

$$\sum_{l=1}^{N} q_l(t_n^{i,l}) \left(1 - \frac{C_l - C_N}{C_l}\right) = C_N \sum_{j=1}^{n} \left(\Delta_{N+1,j} - \Delta_{1,j}\right)$$

De donde obtenemos la ecuación final,

$$\sum_{l=1}^{N} \frac{q_l(t_n^{i,l})}{C_l} = \sum_{j=1}^{n} \left(\Delta_{N+1,j} - \Delta_{1,j} \right) \qquad (6)$$

El significado de la anterior ecuación es que la suma de tiempos entre arribos menos la suma de tiempos de partida de los pp hasta el paquete de prueba n, es igual a una combinación lineal del tamaño de la cola vista por este pp especifico cuando arriba a cada cola del camino.

La anterior ecuación está fuertemente correlacionada con el estado del camino.
Finalmente para la secuencia de n paquetes de prueba se define $\hat{q}_n$

$$\hat{q}_n = \sum_{j=1}^{n} \left[\left(t_j^{0,N} - t_{j-1}^{0,N} \right) - \left(t_j^{i,1} - t_{j-1}^{i,1} \right) \right]; \qquad (7)$$

donde N es la cantidad de enlaces.

Entonces se utiliza $\hat{q}_n$ para determinar el estado de la red y en particular el promedio de $\hat{q}_n$ y la varianza de $\hat{q}_n$ que representan el estado de la red y son correlacionados con los parámetros de video (retardo, jitter y pérdidas de paquetes).

Se agrega también dos parámetros más de $\hat{q}_n$ para determinar el estado del enlace X. Uno es el *percentil de Z%*, que representa el valor de $\hat{q}_n$ para el cual un porcentaje *Z%* de los valores de $\hat{q}_n$ quedan por debajo de dicho valor.
El otro parámetro es el porcentaje de cola vacía calculado como la cantidad de paquetes que son encolados con tiempo cero sobre la cantidad de paquetes enviados por cien (en la secciones 6.4.7.2 y 6.4.7.3 se explican la implementación de la detección de cola vacía y cálculo de porcentaje de cola vacía).
Se estudiará los rendimientos de las predicciones con diferentes *Z* de *percentil Z%.*

3.6 Support Vector Machine (SVM)

SVM [8] [9] es un conjunto de algoritmos de aprendizaje supervisado y es utilizado en problemas relacionados a la clasificación y regresión. Esta herramienta permite entrenar un conjunto de muestras y crear un modelo para luego clasificarlas o predecir futuras muestras. En este capítulo se muestra el método de regresión ya que, para los fines del proyecto, es de interés la predicción.

Supóngase que se quiere entrenar datos con las siguientes características: $\{(x_1, y_1)..(x_k, y_k)\} \subset \chi \times \Re$ donde χ denota el espacio de las entradas que puede ser $\chi \in \Re^d$, y que en este caso corresponde a la varianza de $\hat{q}_n$, promedio de $\hat{q}_n$, percentil

Z% y porcentaje de cola vacía. Lo que hace SVM-regresión es buscar una función $f(x)$ que pase por todos los puntos del entrenamiento con una desviación de no más de $\in$ (ver **Figura 6**), y al mismo tiempo que sea lo más plana posible.

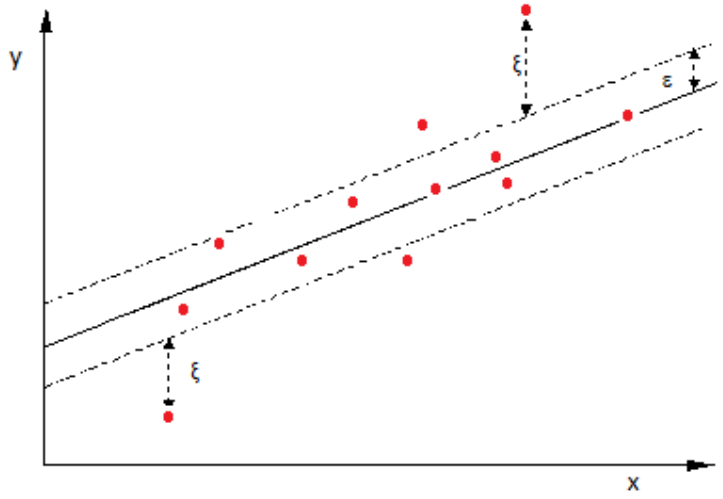


Figura 6

Según [10] se propone la función:

$f(x) = \langle w, x \rangle + b$, con $w \in \chi$, $b \in \Re$ donde $\langle \cdot, \cdot \rangle$ es el producto interno en χ.

La proposición de que sea lo más plana posible se puede traducir en minimizar el producto interno $\langle w, w \rangle = \|w\|^2$. Se plantea el siguiente problema de optimización convexa:

$$\min \frac{1}{2}\|w\|^2$$

sujeto a:

$$\begin{cases} y_i - \langle w, x_i \rangle - b \leq \in, \forall i = 1..k \\ \langle w, x_i \rangle + b - y_i \leq \in, \forall i = 1..k \end{cases}$$

En el problema planteado se debe tener una solución factible para un $\in$ dado, pero puede este no ser el caso y se deba hallar otra solución con un $\in$ mayor. En otros casos es preferible que se encuentre una función que cumpla con las condiciones en la mayoría de los casos y que se asuma un error, permitiendo que algunos puntos caigan fuera de la banda $\in$. Se plantea el siguiente problema como una relajación del problema anterior **[10]:**

$$\min \frac{1}{2}\|w\|^2 + C\sum_{1}^{n}\left(\xi_i + \xi_i^*\right)$$

sujeto a:

$$\begin{cases} y_i - \langle w, x_i \rangle - b \leq \in + \xi_i, \forall\, i = 1..k \\ \langle w, x_i \rangle + b - y_i \leq \in + \xi_i^*, \forall\, i = 1..k \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

Donde la constante C es una relación de peso que determina el equilibrio entre que tan suave será $f(x)$ y el error o cantidad de puntos que caerán fuera de $\in$. Y donde ξ_i, ξ_i^* son variables de holgura del problema relajado (ver **Figura 6**).

Para resolver este nuevo problema se debe plantear el dual utilizando el lagrangeano y la condiciones de Khun-Tucker (KKT) **[11]** para luego llegar a la siguiente solución:

$f(x) = \sum_{i=1}^{n} \left(\alpha_i^0 - \alpha_i^{0*}\right)\langle x_i, x \rangle + b^*$, donde $\alpha_i^0, \alpha_i^{0*}$ son los multiplicadores de Lagrange, y si cumplen que $\alpha_i^0, \alpha_i^{0*} \neq C$ y $|f(x_i) - y_i| < \varepsilon$, para las condiciones de KKT, entonces $\alpha_i^0, \alpha_i^{0*}$ deben ser igual a cero. Por lo que los puntos que caen dentro de la banda ε no contribuyen a la solución. Solo los puntos con los multiplicadores de Lagrange $\alpha_i^0, \alpha_i^{0*} \neq 0$, que caen fuera de la banda ε, son necesarios para construir la solución y son los llamados "support vectors" (vectores de soporte).

Vemos que la solución depende del producto interno de los vectores de soporte.

Hasta ahora se ha buscado la solución usando regresión lineal en el espacio determinado por las entradas, pero puede pasar que la solución en dicho espacio no sea la óptima. Por esto SVM plantea mapear el espacio de entradas $\chi \in \Re^d$ a un espacio de mayor dimensión. Luego busca la solución en este nuevo espacio utilizando regresión lineal.

Si se mapean los datos a través de una función θ la solución planteada quedaría:

$$f(x) = \sum_{i=1}^{k} \left(\alpha_i^0 - \alpha_i^{0*}\right)\langle \theta(x_i), \theta(x) \rangle + b^*$$

Se observa que no es necesario conocer la función θ, sólo basta conocer su producto interno entre los puntos del espacio $\chi \in \Re^d$.

Se denomina Kernel a la función que devuelve el producto interno $\langle \theta(x_i), \theta(x) \rangle$. Por lo tanto conociendo la función de kernel que devuelve el producto interno, se puede encontrar una solución en un espacio de mayor dimensión. El kernel verifica ser simétrico y semidefinido positivo.

En este caso se utiliza el Kernel con base radial por recomendación del SVM Tools aunque también se probó el Kernel lineal sin buenos resultados.

El kernel con base radial (FBR) toma la forma:

$$k\left(x, x_i\right) = \exp^{\left(-\gamma\left(\|x - x_i\|^2\right)\right)}$$

Siendo γ una constante de proporcionalidad.

III- Descripción del sistema

4. Descripción global

4.1 Generalidades.

Se diseña el sistema para su utilización en una red de distribución de video, con el fin de controlar el acceso de los usuarios. Así se garantiza la calidad de video experimentada por todos los clientes cuyas solicitudes han sido aceptadas.
Cada cliente que desee visualizar un video realiza una petición para acceder al mismo, lo que involucra un análisis del estado de la red por parte del sistema, resultando en la aceptación o no de la solicitud.

Para aceptar la solicitud el sistema debe estudiar el estado de la red y verificar las condiciones necesarias para garantizar la calidad de visualización deseada sin que esto implique un deterioro en la calidad de los videos en curso.

¿Cómo hallar las condiciones necesarias para aceptar una petición?
Teniendo una relación de equivalencia o correspondencia entre la percepción de la calidad de un video por parte del espectador y parámetros medibles del tráfico de video, se puede lograr garantizar una calidad percibida por el espectador controlando dichos parámetros, los cuales caracterizan la calidad del video.
Como se mostró en el Capitulo 2, se identifican al retardo, jitter y pérdidas de paquetes como los parámetros del flujo de video que mejor caracterizan su calidad, y es controlando sus valores que se logra garantizar la calidad de un video.

No se abarca el caso de otorgar prioridades especiales ni diferenciar a los paquetes de video frente a otros flujos porque se trabaja con la hipótesis de una red controlada donde el flujo cursado por la misma es únicamente video.

Más bien se aborda el control de los parámetros de video, prediciendo dichos parámetros para un nuevo video y prediciendo también los nuevos parámetros de aquellos videos que están cursando en caso de ingresar uno nuevo.
Se define que se utilizaran dos calidades de video distintas, de manera de poder brindar un tipo de calidad inferior si la más exigente no se puede brindar.
No tiene el mismo impacto el ingreso de un video codificado para calidad media, al impacto del ingreso de uno codificado para calidad alta, es por eso que se debe predecir dependiendo del caso, y son tantos casos como tipos diferentes de video se trafican por la red. En este caso, como se dijo anteriormente, se abarca dos tipos de video utilizando un mismo códec (H.264 como se verá en el capítulo 6) pero codificando el video con distinta resoluciones y bitrates para que requieran distintas exigencias de la red.

Por último, se destaca la utilización del método de aprendizaje estadístico (Capitulo3) como herramienta para lograr una precisa correlación entre parámetros de paquetes de prueba livianos (64 bytes), utilizados para testear el estado de la red, y los parámetros de calidad que tiene un nuevo video al ingresar, como los son el retardo, jitter y pérdidas de paquetes.

4.2 Aclaración sobre las calidades de video utilizadas

Se debe aclarar que en gran parte del documento se refiere a calidad HD y calidad SD, cuando en realidad ninguna de ellas se rigen por definiciones de las normas ITU para iptv ni de otro tipo, sino que ambas calidades que se ofrecen en nuestro sistema son también nombres que se utilizan comercialmente sin mayor rigurosidad y que a modo de ejemplificación sirven para rápidamente asociar que una calidad es mejor que la otra. Se podría haber seguido estándares definidos internacionalmente, en referencia a las calidades de los videos, pero como se explica más detalladamente en el capítulo 6 existieron limitaciones de orden práctico que resultaron en no poder seguir ese camino.

4.3 Hipótesis asumidas

En este punto describiremos las hipótesis más importantes que fueron tomadas para el proyecto, las cuales sirvieron de sustento para la realización del mismo.

4.3.1 Red controlada

La red en donde funciona el sistema se asume como completamente controlada, no se genera tráfico externo a los clientes ni tampoco hay políticas de calidad de servicio que puedan tratar flujos de tráfico de distinta manera, sean con prioridades o colas diferenciadas. Se asegura que todo el tráfico que pasa por los nodos se genera dentro de la red y no existen tráficos generados espontáneamente como podría pasar en Internet.
Con lo último se quiere destacar que se asume estacionariedad de la red mientras no se acepte una nueva solicitud de un cliente. Cuando se acepte un cliente, cambia el estado de la red a un nuevo estado estacionario.

4.3.2 Punto de congestión

El diseño de implementación del sistema se realiza pensando en una red de distribución con las siguientes características en su topología y los siguientes tipos de congestión.

Congestión en el backbone

Cada video cursa por el backbone en alguna parte de su trayecto, y por un trayecto del backbone se cursan varios videos en simultáneo. El servidor vlc, que se conecta directamente al backbone, genera tráfico a una tasa igual a la demanda de los clientes. Por lo tanto, conforme aumente la demanda, aumenta el tráfico por los distintos enlaces del backbone pudiendo presentarse una sobrecarga en uno de estos enlaces y así generarse un posible cuello de botella o punto de congestión dentro del backbone. Es responsabilidad del administrador de la red el controlar estos recursos.
Del mismo modo, conforme aumente el tráfico que genera el servidor vlc, se debe prestar atención a los recursos asignados al acceso del servidor vlc al backbone ya que allí es posible encontrar un punto de congestión si no se dimensiona adecuadamente

dicho tramo.

Congestión en el acceso

Se considera también los recursos finitos que se poseen en la red de acceso en la cual se multiplexa el tráfico de los clientes. Aquí también se encuentra un posible punto de congestión ya que por allí se cursa el tráfico correspondiente a una cantidad de clientes que por alguna razón comparten un medio físico.

Congestión en el acceso- ultima milla

Y por último se referencia dentro de la red de acceso, al tramo que parte del nodo de acceso más cercano al cliente y se extiende hasta el equipo mismo del cliente.
La congestión se genera, en este caso, si un cliente pide varios videos simultáneos.

4.3.3 Trato igualitario a todo el tráfico en la red

Trabajando bajo el supuesto de que todos los paquetes que transitan por cierto enlace de la red son tratados de la misma manera, entonces el retardo, jitter y pérdidas que presenta un video en un nodo, tendrán estadísticamente los mismos valores que el resto de los videos por ese enlace. De esta forma los valores de los parámetros de calidad predichos para un nuevo video son también los predichos para los restantes videos que compartan el camino.
Por lo tanto si la predicción a un cliente da dentro de los valores aceptables, se sabe que no se perjudica al resto de los clientes, sin importar la arquitectura de la red.

4.3.4 Siempre existe un cliente viendo video HD

Al tomar la hipótesis 4.2.3 como cierta, surge un caso de particular interés que el sistema debe resolver.
Al utilizar los algoritmos descritos en el capítulo 3, el sistema debe atravesar una etapa de aprendizaje con cada cliente para poder formar su modelo de predicción. También se definió que la red de distribución contará con 2 calidades de video distintas con distintos rates y distintos parámetros de calidad a respetar, por lo tanto se debe formar un modelo para cada calidad y luego se predice si cada calidad es garantizada.
Supongamos ahora que se predice a un “ClienteA” que comparte el acceso con “ClienteB” que ya se encuentra viendo un video con calidad HD. Si la predicción al primer “ClienteA” devuelve parámetros que garantizan una calidad HD, entonces por 4.2.3 el “ClienteB” no se ve afectado por el video HD hacia “ClienteA” si hace una petición. Ahora, si la predicción no garantiza HD y sí lo hace para SD surge el caso que se destaca como particular.
El modelo creado para la calidad SD, debe ser entrenado con la secuencia de paquetes de prueba y luego el tráfico de video SD. Por lo cual, la predicción al “ClienteA” sólo da información de cuál es la QoS que tendrá el tráfico de videos SD en ese escenario de tráfico cruzado. Cómo las condiciones de QoS de HD y SD son distintas (restricciones en cuanto a retardo, jitter y pérdidas que tiene cada tipo de video) y se asume que todos los paquetes son tratados iguales, se puede deducir que los paquetes HD que van hacia el “ClienteB” se comportan de la misma manera que los SD hacia “ClienteA”. Esto último implica necesariamente que no estemos garantizando la QoS de los paquetes a “ClienteB” ya que solo se comparó la predicción de “ClienteA” con los umbrales

definidos para garantizar SD.
El caso que planteamos aquí se resuelve de una manera un tanto restrictiva pero que asegura el correcto funcionamiento del sistema.
Se parte de la base que el sistema no conoce la arquitectura de la red y no es de interés que la entienda por lo que no sabe si el “ClienteA” y “ClienteB” comparten el acceso o no. Entonces para resolver el problema se asume que todo el tiempo existe un cliente recibiendo un video de tipo HD y las predicciones de SD se deben comparar con los requerimientos que permitan garantizar la calidad de video HD.
De esta forma se puede garantizar que permitiendo un video SD no se perjudica a un cliente recibiendo HD.

5. Desarrollo y arquitectura del sistema

5.1 Introducción.

A la hora de desarrollar el sistema se tuvieron en cuenta los siguientes requerimientos:

Modularidad
Desde su comienzo el sistema fue pensado como la unión de dos módulos independientes que se comunicaban entre sí.

Bajo Acoplamiento
Que el sistema pueda intercambiar alguno de sus módulos y manteniendo la forma de comunicación entre ellos pueda funcionar correctamente. Pensado para ganar en reusabilidad del sistema, por ejemplo que modificando el M.E se pueda reutilizar el sistema para otro tipo de transferencia de datos

Adaptabilidad a cambios de requerimientos de los parámetros de la red
Fue importante que el sistema fuera flexible en cuanto a este tipo de cambios ya que el desarrollo de su implementación fue en paralelo con el estudio de requerimientos para distintos tipos de video, también claramente resulta ventajoso para elegir distintos codecs y hasta para otras aplicaciones similares (por ejemplo en voip se utilizan los mismos parámetros para definir QoS pero con requerimientos distintos).

En referencia al punto anterior, se destaca con particular importancia la forma iterativa que tuvo el desarrollo del sistema para lograr mejores resultados. Al utilizar algoritmos que estiman la QoS hubo que diseñar, implementar, testear y luego rediseñar gracias a los resultados obtenidos, para proseguir con la iteración.

5.2 Esquema básico de control de admisión

Antes de seguir con la descripción de la arquitectura señalaremos conceptualmente como debe funcionar un mecanismo de control de admisión.

Cómo se ve en la Figura 7, el esquema de control de admisión debe contar con un *Proceso de estimación* (basado en mediciones) del estado de la red y de lo que la misma puede brindar si se admitiera una conexión. También contiene los *Requerimientos de QoS* que se deben tener en cuenta para respetar la calidad de video. Estos conllevan a los *Criterios de Admisión* que son las reglas para admitir o no a un cliente.
Por último existe una evaluación de todo lo anterior hecha por *Control de Admisión* y una toma de *Decisión de Admisión.*

Este esquema representa básicamente lo que efectúa el sistema y debe estar reflejado en la arquitectura diseñada.

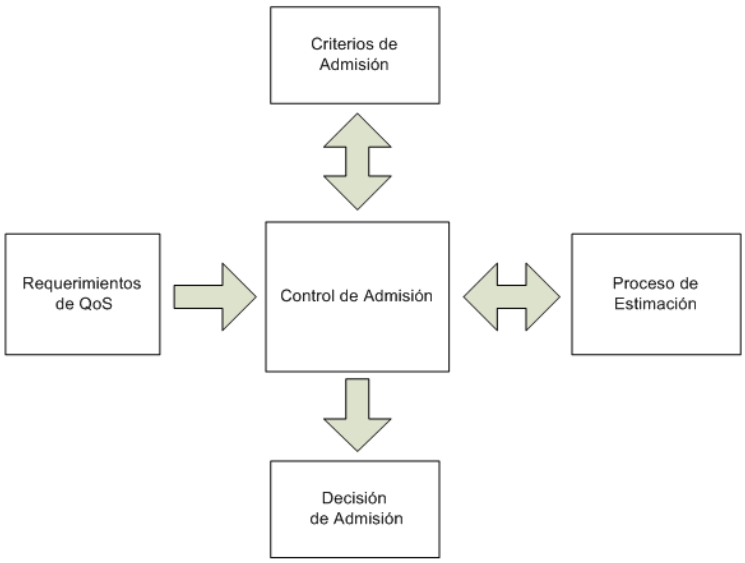


Figura 7- Esquema de Control de admisión

5.3 Impacto del lenguaje de programación elegido (lenguaje C)

Con la intención de adentrarse en la arquitectura lógica del sistema se describen algunas de las razones más importantes que llevaron a que el proyecto se desarrolle en su totalidad en el lenguaje de programación C. Se realiza tal descripción ya que el lenguaje impacta directamente en el tipo de programación que se llevó a cabo y por ende también en el tipo de arquitectura.

Razones de la elección:

-Lenguaje utilizado en Squid

Cómo se ha señalado con anterioridad, el sistema consta de dos módulos y uno de ellos es el Proxy. Para diseñarlo e implementarlo se prefirió estudiar un completo, robusto y reconocido proxy del cual está probado su funcionamiento para luego estudiar la forma de adaptarlo a la necesidad del proyecto. Si bien se profundiza en otras secciones se puede adelantar que el proxy de estudio fue el Squid. Este proxy, que se logró adaptar a las necesidades del sistema, está escrito en C [2] y para lograr un entendimiento cabal de su funcionamiento se requirió un estudio detallado de la herramienta. Una vez que se pudo adaptar el conocido proxy a las necesidades del sistema (como se verá en el capítulo 7), se comenzó a notar un aprendizaje del lenguaje por parte de los integrantes y se verificó cierta habilidad en su utilización. Lo que hizo que se presentase en inicial ventaja al otro lenguaje posible (JAVA) antes de siquiera haber comenzado a implementar el sistema.

-Restricciones de precisión de reloj

Otra ventaja importante que C tuvo a la hora de ser el preferido fue la precisión que tiene en cuanto a aplicaciones con restricción de tiempo. En comparación con el otro

[2] Si bien la versión squid-3 se encuentra escrita en C++, la documentación resultó bastante escasa en tal versión y optamos por una de las que estaban escritas en C

lenguaje en disputa, se supo que JAVA no funciona del todo bien cuando el programa a implementar es sensible a diferencias del orden de los 10ms. Cómo se vio en el capítulo 3.5 es de suma importancia que el envío y recepción de paquetes se haga de forma precisa y la resolución que tiene la medición también es vital, por lo que ésta ventaja fue la que inclinó la balanza definitivamente.

En cuanto al último argumento en contra de JAVA, hay que aclarar que existe un paliativo, una herramienta llamada JNI (Java Native Interface) que permite a JAVA dialogar e interactuar con lenguaje nativo en C.

De todas maneras, gracias al manejo que se adquirió con C para Squid y luego el buen desempeño de C para altas exigencias de tiempo, se consideró que no valía la pena diseñarlo en otro lenguaje que no fuera C. Más aún teniendo en cuenta, que en primera instancia no había necesidad de generar una interfaz demasiado amigable para el operador del sistema, ya que la mayoría de las acciones se efectúan de manera transparente para él (Aunque el sistema si concede reportes para evaluar su correcto funcionamiento). Seguramente en el trabajo a futuro se podrá implementar una interfaz gráfica con el usuario para poder mejorar la experiencia del mismo pero en esta etapa se le dio más peso a los argumentos antes mencionados.

Impacto del lenguaje en la arquitectura:

La elección tomada en cuanto al lenguaje de programación implica un diseño seguramente más rudimentario de lo que se podría haber hecho de elegir un lenguaje orientado a objetos.

Sólo se contará con dos módulos bien diferenciados que intercambian la información necesaria para tomar sus decisiones y también una aplicación que se encuentra alojada en la PC del cliente (llamada ApCliente) que recibe órdenes del Módulo Estadístico (M.E) y del Proxy.

En la siguiente sección se dará una primera descripción de cómo funciona el sistema, sin entrar en demasiado detalle pero explicando cómo interviene cada módulo.

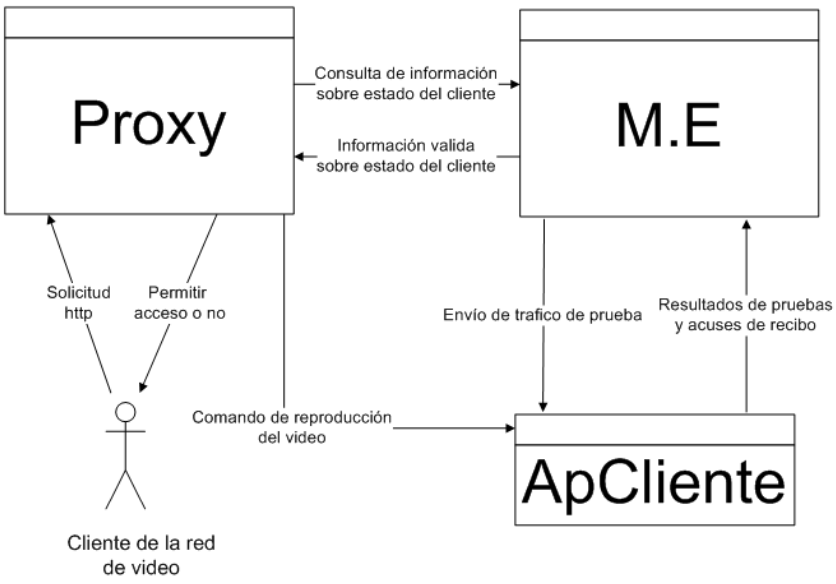


Figura 8

5.4 Descripción de módulos

Teniendo en cuenta el esquema de la Figura 7 pagina 30, debemos ir construyendo nuestra arquitectura para que los módulos que forman parte del sistema puedan respetar aquella dinámica.
Por otra parte en la Figura 8, se muestra la interacción entre los dos módulos principales, la ApCliente que es la aplicación que escucha siempre en la PC cliente y también una representación del cliente verdadero. Si bien el diagrama es un abuso de lenguaje en sí mismo y no sigue ninguna convención en cuanto a presentación de arquitecturas (sea UML o cualquier otro) es bastante sintético y explicativo.

Cada uno de los principales módulos se desenvuelve principalmente en un escenario macro de funcionamiento.
Describiremos cada módulo, primero marcando el escenario en el cual se desenvuelve.

5.5 Módulo Estadístico (ME)

El M.E es actor principal cuando el sistema está creando un modelo para la estimación de la QoS a un cliente.

Para este escenario no es necesario un diagrama alternativo al de la Figura 8. Ya que solo participan activamente el M.E y ApCliente. El M.E generando tráfico y ApCliente dando los resultados.

5.5.1 Funcionalidades

El M.E es el encargado de crear un modelo de estimación de la calidad de servicio para cada cliente de la red de video y luego predecir la misma según los distintos escenarios de tráfico en la red.
Para eso se implementan las funcionalidades que se describen a continuación.

5.5.2 Entrenamiento de clientes

Para generar el modelo, el M.E utiliza el algoritmo de entrenamiento ya descripto en el Capitulo 3. Lo que hace es generar tráfico hacia el cliente, que es recibido por la aplicación APCliente que luego envía el resultado de la transferencia. La transferencia ya fue descrita en el capítulo 3 (paquetes de prueba y luego tráfico de video).

5.5.3 Predicción de clientes

Luego de generar el modelo con suficientes entrenamientos, el M.E está en condiciones de poder estimar cuál será la QoS de un video con solo enviar los paquetes de prueba.
Entonces realiza esa tarea periódicamente, de manera de tener estimaciones actualizadas según los cambios de estado de la red.

Luego de señalado lo anterior y volviendo a la Figura 7 de la pagina 30 claramente el M.E es el encargado del **Proceso de estimación**.

5.6 Módulo Proxy

Cuando el cliente (el cliente de la red de video, no ApCliente que es la aplicación en la PC del cliente), quiere acceder a un nuevo video y su acceso está supeditado al control del sistema, allí es cuando el Proxy interviene en el rol principal.

En la Figura 9 se muestra la secuencia de interacciones del escenario actual**.**

Los distintos contenidos que el cliente puede reproducir aparecen en un menú en la página web que el proveedor del servicio pone a disposición. Cuando el cliente hace su opción de video también debe elegir la calidad a la cual quiere que se reproduzca. Una vez realizado este paso (paso 1 según la figura) el módulo Proxy entra en acción

Se aclara que el Servidor de Video se toma como externo al sistema ya que su función es servir el video cuando el cliente es validado. El servidor vlc no participa de ningún evento anterior o posterior al señalado.

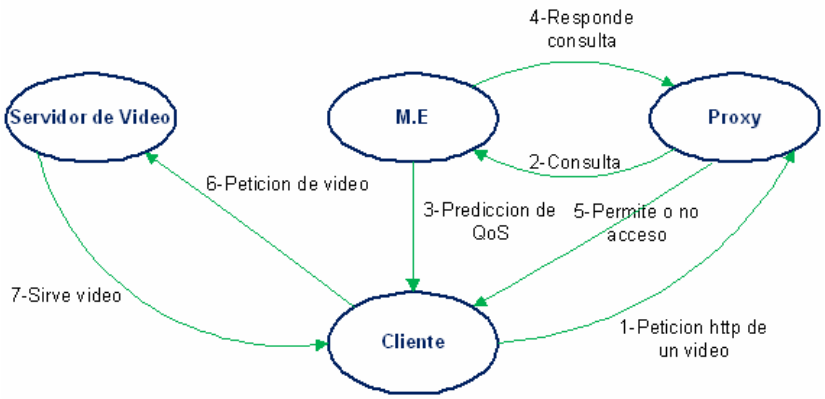


Figura 9

A continuación se describe al Proxy.

5.6.1 Funcionalidades

Cómo se dijo antes el cliente tiene acceso a los contenidos de video en una página web. Por lo tanto como todo proxy web la funcionalidad del módulo Proxy es interceptar la navegación del cliente para captar la información sobre el tipo de contenido que el cliente quiere obtener y luego en base a esa información consultar al M.E para saber si es posible otorgar lo que él desea.

5.6.2 Control de acceso

Cómo se decía en la funcionalidad, el Proxy es el que intercepta la petición del cliente, por lo cual es el encargado de realizar el control de acceso. El M.E no se entera de que tal cliente quiere acceder a tal video. Eso lo debe realizar el proxy, contando con la información brindada por el M.E

La interacción principal del Proxy es con el M.E, esta interacción se detallará más adelante, pero se puede adelantar que se usó el protocolo SNMP (Simple Network Management Protocol.), con el cual bajo una lógica de consulta y respuesta puede informarse sobre el estado de un cliente.

También existe una interacción con el ApCliente. Una vez que la solicitud del cliente ha sido aceptada, el Proxy le envía una orden al ApCliente para que se ejecute el reproductor de videos escuchando a un determinado puerto del servidor de video.

Una vez más, volviendo a la **Figura 7** de la pagina 30 se puede ver que en el Proxy se definen los **Criterios de admisión** y además es el encargado tomar la **Decisión de Admisión**, lo que lo convierte en el **Control de Admisión** en sí mismo.

5.7 ApCliente

El ApCliente participa en ambos escenarios antes descriptos.

5.7.1 Funcionalidades

En el primer escenario, participa recibiendo, procesando y enviando los resultados al M.E para que el mismo pueda construir el modelo.
En el segundo escenario, cuando el Proxy ha decidido que se acepte la petición del cliente, le envía un comando por TCP al ApCliente para que el cliente pueda acceder al video que brinda el servido.

5.8 Arquitectura física del sistema.

Se ideó el sistema de forma que los módulos principales estuviesen separados físicamente. La Figura 10 repite lo mostrado en la Figura1. Aquí se ve nuevamente como está ideada la arquitectura.

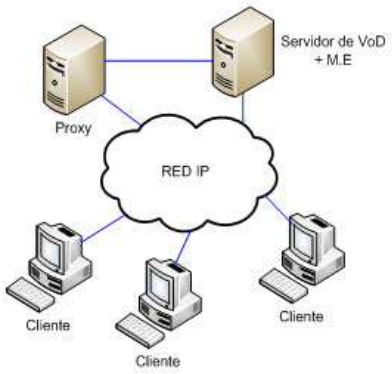


Figura 10

Lo que sí se considera deseable es que el M.E evalúe los parámetros de calidad de servicio de la red desde el punto de vista del servidor de video por lo que podríamos ubicar un M.E por servidor de video o en el caso de que se sobrepase la capacidad de procesamiento (ya que servir video consume muchos recursos) sería bueno al menos ubicarlos en un mismo lugar físico (o sea que encuentren en un mismo nodo de la red).

6. Implementación de Módulo Estadístico (ME)

6.1 Introducción

El módulo estadístico será el responsable de la estimación de QoS que percibirán los clientes ante peticiones de video y deberá comunicar al proxy del sistema si debe aceptar o rechazar dichas peticiones para garantizar la QoS deseada.

Para esto es necesario implementar una etapa de entrenamiento para cada cliente en el cual se elabore un modelo que mapea el estado de la red en los parámetros de QoS (retardo, jitter y tasa de pérdidas). Luego una etapa de validación del modelo creado donde se envíe un tren de pp seguido de una secuencia de video sobre distintos estados de la red y se comparen los parámetros de QoS predichos (a partir de la caracterización del estado de la red mediante los pp y el modelo creado) con los valores reales medidos directamente de los paquetes de video.

Como se verá en el punto *7.5, s*e necesitará también implementar la automatización de las predicciones y entrenamientos según los distintos escenarios que pueda haber en la red, para mantener una estimación razonable del estado del camino a todos los clientes y de los parámetros de QoS de interés.

6.2 Método de Entrenamiento

Como ya se mencionó, el proceso de entrenamiento es el encargado de crear una función que permita estimar los parámetros de calidad de servicio extremo a extremo de un video, a partir de los parámetros que caracterizan el estado del camino que recorre el tráfico en la red.

En primer lugar este método envía una secuencia de paquetes de prueba seguida de una secuencia de paquetes de video. Estos paquetes son recibidos por un proceso residente en el receptor o cliente el cual le envía al módulo estadístico (ME) la información relativa a cómo fueron tratados los paquetes, al pasar por el camino en el que se cursó el tráfico por la red. En esta etapa el ME procesa la información y genera un modelo de correlación entre el estado de la red (información obtenida de los pp) y la calidad del video (información obtenida de los paquetes de video) asociado al cliente. Para ello se utilizan las librerías de svm (lib-svm [12] [13]).

A partir del momento que se crea el modelo, es posible realizar predicciones sobre la calidad que tendrá un video determinado, pero para que las predicciones sean buenas es necesario crear un modelo con los suficientes entrenamientos, lo cual explicaremos en la descripción del programa principal.

Se realizan entrenamientos para dos tipos de video: “calidad alta” y “calidad media o estándar”, caracterizadas cada una por su códec, tasa de codificación, cantidad de cuadros por segundo, etc.

6.3 Método de Predicción

La predicción de un cliente se realiza cuando se tiene el modelo que correlaciona los parámetros de calidad del video con los parámetros que caracterizan el estado de la red.
Luego de tener el modelo creado por el entrenamiento, hay que tener conocimiento del estado de la red. Por lo tanto lo primero que se realiza en este método es enviar sólo una secuencia de paquetes de prueba para calcular los parámetros que caracterizan la red, que como se mencionó anteriormente, se calculan a partir del estimador de la cola o del camino en la red: $\hat{q}_n$.
Una vez recibido el archivo con los datos requeridos y procesados, se tiene conocimiento del estado de la red y utilizando las librerías de svm (lib-svm) se llama a la función *predict* (sección 6.4.7.5) para predecir la calidad que presentará el futuro video.

6.4 Implementación

A continuación se describe en forma más detallada la metodología y realización de cada una de las partes que forman el funcionamiento de los métodos entrenamiento y predicción.

6.4.1 Envío y recepción de paquetes de prueba y video

Según el método descripto en la **sección *3.4***, para poder estimar el estado de un camino de la red es necesario enviar una secuencia de paquetes de prueba y registrar los tiempos de partida y llegada de los mismos.
Para ello, el módulo estadístico inserta en la carga útil de cada paquete de prueba un número de secuencia, que servirá para identificar el paquete, y un “timestamp” que indicará el momento en el que el paquete fue transmitido al destino.
Del lado del receptor, es decir el cliente, se reciben los paquetes de prueba (eventualmente podrían haber pérdidas de paquetes) y se guarda el tiempo de llegada de cada paquete.
Es importante que el tamaño de los paquetes de prueba sea lo más pequeño posible para no cargar a la red. Por lo tanto, dicho tamaño queda determinado por el número de bytes necesarios para enviar el número de secuencia y el timestamp.
El tamaño de la carga útil será entonces de 22 bytes y considerando los encabezados y tiempos entre partidas constantes de 10ms, tendremos un tráfico CBR de 51,2kbps.

Otra característica a notar, es que los paquetes de prueba serán transportados sobre UDP ya que no tiene sentido que hayan retransmisiones.

En cuanto a la medición de los parámetros de QoS en la red (retardo, jitter, tasa de pérdidas, etc.) para un video, también será necesario contar con números de secuencia, tiempos de partida y llegada de los paquetes. Como en principio estos datos no se encuentran disponibles en la carga útil de los paquetes, para la etapa de entrenamiento será necesario generar una simulación del tráfico insertando los datos requeridos.

6.4.2 Elección de la secuencia de video para etapa de entrenamiento.

Como ya se ha establecido en la sección 2.1 el bitrate de un video codificado será más elevado en escenas más complejas y con mayor cantidad de movimiento pudiendo haber grandes diferencias entre dos escenas distintas. Por lo tanto para la etapa de entrenamiento correspondiente a cada nivel de calidad de video (determinado por el códec, tasa de codificación, resolución, frame-rate, etc.) elegimos una escena que represente el "peor" caso posible para cada nivel de calidad de video a ser transmitido.
De esta manera, nos aseguramos que si se cumplen los requerimientos de QoS establecidos para la secuencia de video usada en el entrenamiento, también se cumplirá para el video a transmitir.

6.4.3 Simulación de tráfico de video

Para lograr simular el tráfico se capturaron los paquetes del video mediante un "sniffer" (se utilizó el programa *Tcpdump* [14]) pudiendo extraer el tamaño de los paquetes y los tiempos entre partidas de los mismos. De esta forma, se elaboró un método a ser utilizado por el M.E. capaz de importar la captura de una secuencia de video y reproducirla hacia un destino. Esto es, hacia una IP y puerto de destino.

No obstante, se encontró que el método de generación de tráfico a partir de la captura lograba simular el video en forma efectiva dependiendo de la prioridad con la que se ejecuta el proceso en el sistema operativo (en nuestro caso: Ubuntu) y el bitrate de codificación del video (tiempos entre partidas de los paquetes).

En general mientras más exigentes sean los tiempos entre partidas de los paquetes se obtienen peores resultados, obteniendo bitrates medios por debajo de los esperados.

Esto se debe básicamente a que en los momentos que el tráfico presenta "picos" se demora más tiempo en enviarse los paquetes de lo debido.

En este sentido se tomó en cuenta para la elección del lenguaje de programación en el que se implementarían las funcionalidades relacionadas con el envío y recepción de

paquetes de video, que pueda efectuar las operaciones con una buena granularidad temporal.

Cómo se comentó en la sección 5.3 se eligió en lenguaje C, por su buena respuesta frente a procesos en tiempo real.

De todas maneras, los equipos con que contamos durante el proyecto no tienen un procesador demasiado potente y se notó una merma del bitrate entre el tráfico de video enviado cuando el procesador no estaba exigido a cuando sí lo estaba. Esto se debe a que el procesador debe ejecutar instrucciones que lo ocupan un determinado tiempo que hace aumentar al tiempo de salida entre los paquetes lo que implica un suavizado de los "picos" antes mencionados y por lo tanto una caída del bitrate medio (tamaño de la transferencia/ tiempo de la transferencia).

Un paliativo que se encontró para mejorar la eficiencia en el envío de las capturas, fue estimar (con una bandera de tiempos en cada iteración del envío) el tiempo que tarda el procesador en realizar otras instrucciones y restárselo al tiempo entre salidas de los paquetes.

Se vieron mejores resultados con el paliativo, sin embargo hay ocasiones en el que el tiempo estimado a restar supera el tiempo de salida entre paquetes lo cual hace que de todas maneras se llegue "tarde" a enviar el siguiente paquete. Esto introdujo un error máximo de 1.3% en el bitrate medio para videos que tenían picos de hasta 4Mbps.

Se considero utilizar videos que no tuvieran demasiada variabilidad para evitar "picos" altos. Luego de tener en cuenta lo anterior se observó que el tráfico simulado de video era razonablemente similar al real.

En la Figura 10 se puede observar los gráficos de *Bitrate vs. Tiempo* de ambos tráficos.

Para verificar en forma cuantitativa la similitud, se capturaron los paquetes de distintas secuencias de video simuladas y la reales, y a partir de dichas capturas se calcularon y compararon los bitrates medios obteniendo en el peor caso un error relativo de 2% aproximadamente.

En las Figuras 10a y 10b se puede apreciar como la transferencia del video real es seguido en forma razonable por la simulación enviada por el M.E.

Figura 10a- Video Real

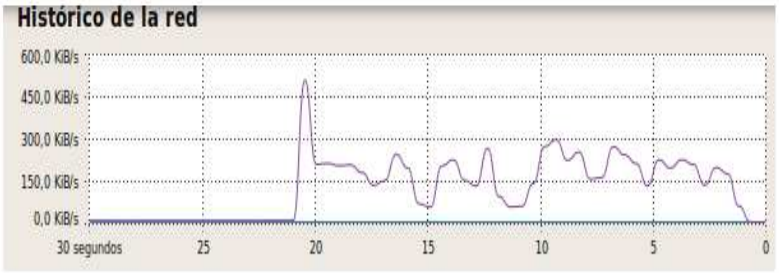


Figura 10b - Video simulado

Concluimos que se podía obtener un comportamiento aceptable para una tasa máxima de codificación de 1,5Mbps lo cual representa una limitación para nuestro sistema debido a la limitación de nuestros equipos.

6.4.4 Envío de información recolectada al M.E.

Se observa que si bien los paquetes de prueba y video simulado se envían utilizando UDP, y que el receptor podría agregar en la carga útil el timestamp de llegada y de la misma manera enviarlos de vuelta al ME, es importante que la información recaudada sea enviada en forma confiable. Por esta razón, se eligió enviar la información vía FTP (utilizando TCP como protocolo de transporte).

El receptor genera un archivo con los números de secuencia, tiempos de partida y llegada de cada paquete de prueba y video, y lo envía vía FTP al módulo estadístico para que procese los datos, elabore un modelo (etapa de entrenamiento) o prediga el estado del camino una vez elaborado el modelo (etapa de predicción).

Dado que la información se envía en texto plano vía FTP, existe la posibilidad de que esta sea capturada y/o modificada por un tercero. Existe la posibilidad de realizar la transferencia en forma segura mediante el uso de FTP con SSL/TLS [31] para el cifrado de la información.

De todas formas, esto no fue implementado y se dejó como trabajo a futuro.

6.4.5 Aplicación cliente

Para implementar estas funcionalidades del lado del receptor, se programó una aplicación cliente que permanece escuchando en un puerto preestablecido a la espera de una instancia de entrenamiento o predicción. Adicionalmente, si el usuario hace una petición (vía web) y es aceptada por el sistema, el cliente también escuchará en ese mismo puerto[3] a qué servidor de video debe realizar la petición de video a demanda. Esto último permite la flexibilidad necesaria para realizar balanceo de carga de clientes entre varios servidores de video.

Para poder diferenciar entre entrenamiento, predicción o si se trata de información respecto al servidor de VoD que debe solicitar el video, el M.E. envía un mensaje (mediante un socket TCP) en el que se le avisa al cliente de cuál de las tres posibilidades se trata.

6.4.6 Sincronización ME-Cliente

Un elemento importante para el funcionamiento del sistema es la sincronización de relojes entre el M.E. y el cliente ya que para la medición del retardo será necesario comparar los timestamps insertados a la salida y llegada de los paquetes video.
Lo ideal para lograr una buena sincronización sería hacer uso de relojes GPS debido a su alta precisión. Pero al no disponer de los mismos, utilizamos el protocolo NTP [15] (Network Time Protocol) de manera que tanto el M.E. como el cliente se sincronicen contra un servidor NTP (lo más cercano posible) en Internet.
Mediante el uso de NTP se obtuvieron errores que oscilaban entre 5 y 20 ms, que en principio no son aceptables para retardos pequeños pero sí para retardos altos que son los que realmente nos interesa medir o predecir con mayor exactitud. Por ejemplo, si se tiene como criterio de aceptación para una videoconferencia que el retardo medio no supere los 200ms, es de interés poder predecir con mayor exactitud cerca del umbral de decisión.

Para estimar los errores de sincronización se enviaron instancias de entrenamientos (envió de paquetes de prueba + video) con la red vacía de manera que no se produzcan encolamientos y se midieron los retardos en el video en los paquetes de video.
Entonces, despreciando el delay de propagación y tiempo de servicio de los paquetes (que en una LAN la suma de ambos tiempos son del orden de décimas de milisegundo), los retardos medidos corresponden al error de sincronización.

Esta misma idea se reutilizo para compensar el error de sincronización en la medición de retardo para distintos estados de la red, considerando que en los momentos en que

[3] El cliente asume que cualquiera de las tres situaciones posibles no se solapan en el tiempo.

la cola está vacía el retardo es nulo. En la sección 6.4.7.2 se explicará con mayor detalle.

Por otra parte, se observa que para el caso del cálculo de la secuencia $\hat{q}_n = \frac{q(t_n^i)}{C} = \sum_{j=1}^{n}\left[(t_j^0 - t_{j-1}^0) - (t_j^i - t_{j-1}^i)\right]$ para estimar el estado del camino hacia un cliente, no se precisa sincronización ya que sólo depende de los tiempos entre arribos (que son medidos por el reloj del cliente) y los tiempos entre partidas (medidos por el reloj del M.E.) de los paquetes de prueba. Por lo tanto, no se introducirán errores en dicho cálculo por de-sincronizaciones entre relojes.
En forma análoga, para el cálculo del jitter de video medido como el valor medio de las diferencias entre retardos de paquetes consecutivos, se puede hacer un razonamiento similar si se reescribe la diferencia de retardos como la diferencia entre tiempos entre arribos y tiempos entre partidas.

6.4.7 Proceso de información recolectada

Una vez obtenido del cliente el archivo con los datos después de cada entrenamiento, se comienza a procesar esta información. Cada línea del archivo contiene toda la información necesaria de un paquete, como lo es la secuencia, el tiempo de partida y el tiempo de arribo desde el ME al cliente.

6.4.8 Estructura de ordenamiento de datos

Para este proyecto un cliente es una dirección ip. Por lo tanto solo podrá haber un cliente para una ip de acceso. En caso que el cliente se encuentre en una red interna detrás de la ip de acceso y utilizando la misma como origen de las peticiones, como ocurre con la utilización del nat, será el único dentro de la red interna. No se implementó diferenciar los clientes a través del puerto, aunque podría tratarse esto como trabajo a futuro.

Los datos serán almacenados dentro del ME en un sistema jerárquico de carpetas. Toda la información de un cliente será guardada bajo una carpeta con su ip de acceso como nombre. Esta carpeta contendrá los entrenamientos y predicciones para cada una de las calidades de los videos así como los modelos de entrenamiento correspondientes al retardo, jitter y tasa de pérdidas.

6.4.8.1 Separación y distinción de paquetes

Cada entrenamiento a un cliente se hace para un tipo de video específico o más de uno a la vez (SD, HD o ambos). El archivo que llega al ME desde el cliente a través de ftp

tiene todos los paquetes sin diferenciaciones, tanto los paquetes de prueba como los de videos. El módulo lo primero que realiza es separar y procesar dichos paquetes en forma independiente. Por un lado procesa los paquetes de prueba y por otro los de videos de una calidad, y si hubiera una segunda calidad en el entrenamiento los procesa en forma independiente también. Para esto el ME diferencia a través del número de secuencia si un paquete pertenece a uno u otro conjunto.

La elección de la cantidad de paquetes de prueba la explicaremos en el próximo punto, pero podemos adelantar que dicha elección se debe a la necesidad de encontrar varios ciclos de cola vacía en el cuello de botella durante el envío de la secuencia de paquetes de prueba. Dicho de otra forma, se necesitan tener varias oscilaciones en la cola del enlace para obtener una lectura más certera del estado de la red. De lo contrario, se van a obtener distintas lecturas para un mismo estado (dependiendo de en qué momento se envíen los pp) y por ende se obtendrá un "peor" modelo.
En principio, para un escenario en particular de pruebas con un video en H264 de bitrate de 1.5Mbps y tráfico cruzado (alrededor de 15 Mbps) compuesto por videos de las mismas características, se estimó que 15 segundos de paquetes de prueba eran suficientes para encontrar varios ciclos de cola vacía en la mayoría de los estados de la red de interés que llegan hasta el 90% en media de la capacidad del enlace. Como se verá en el capítulo 11, fue necesario enviar paquetes de prueba durante un período mayor para obtener mejores resultados.

Uno de los puntos claves para que el sistema funcione en forma correcta es poseer el conocimiento del estado de la red, para correlacionarlo después con los valores que caracterizan la calidad de un video. Como explicamos en la sección 3.5 , la elección de los parámetros que caracterizan el estado de la red fue: el promedio de q_n, la varianza de q_n, percentil 90% y porcentaje de tiempo con cola vacía. Estos valores se calculan a partir de los tiempos de arribo y tiempos de partida de los paquetes de prueba.

Para el cálculo de q_n se utilizó la fórmula establecida en la Ec. (7) en la sección 3.5
Como ya se comento, dicha igualdad será cierta siempre que el primer pp encuentre la cola vacía y de lo contrario deberá sumarse un término q_0 correspondiente al tiempo que esta encolado el primer pp.
Para determinar el valor de q_0, es esencial que se encuentre la cola vacía en algún momento mientras se envían los pp (paquetes de prueba). En caso de que el primer paquete encuentre la cola vacía entonces el valor de q_0 es cero, pero si el primer paquete es encolado entonces el valor de q_0 es distinto de cero y si no se toma en cuenta será introducido un error a cada q_n. Afortunadamente ese error es constante y calculable, sólo se debe detectar cuándo el paquete i encuentra la cola del enlace vacía y calcular el q_i para ese punto, pues este valor q_i debería ser cero. Pero si no lo es, entonces q_i es el error introducido por tener un q_0 distinto de cero. Se resta entonces a cada q_n el valor de $q_i = -q_0$ llevando a q_i a cero y a todos los q_n a su valor correcto.

En las pruebas realizadas no se obtuvieron buenos resultados en los casos que se supuso cola vacía al comienzo del entrenamiento cuando en realidad no lo estaba, hallándose un modelo erróneo.

6.4.8.2 Método de detección de cola vacía

En el cálculo de porcentaje de cola vacía primero se define un criterio para determinar cuando la cola está vacía. En la siguiente figura se muestra la evolución del valor de q_n a través del tiempo o secuencia de paquetes.

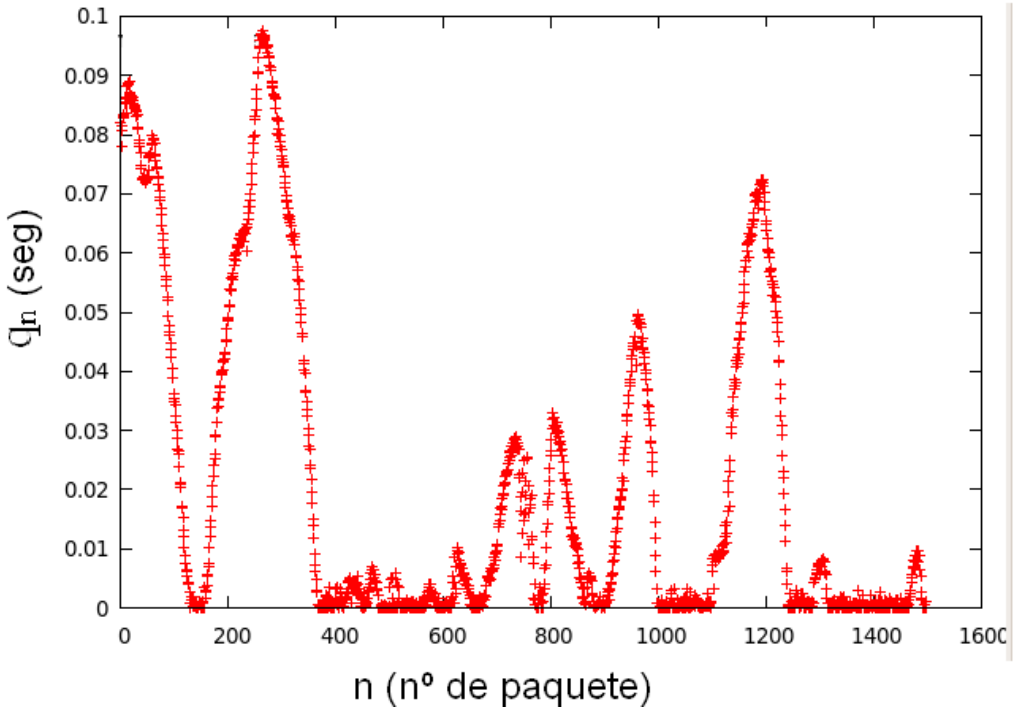


Figura 12

La gráfica corresponde al tiempo de encolamiento de los paquetes para un enlace que se saturó por momentos. Se puede observar claramente los puntos correspondientes a cola vacía.

Si un par de paquetes consecutivos encuentran la cola vacía, entonces los tiempos entre partidas y llegada de los mismos deberán ser iguales. Si observamos la Ec. (7) esto se traduce en $q_k=q_{k-1}$ para dos paquetes consecutivos k y (k-1).
Sin embargo, si bien esta condición asegura que el tiempo de encolamiento fue el mismo, no asegura que la cola estuviese vacía, ya que podría cumplirse también si estos dos paquetes encontraran un mismo tamaño de cola no nulo.
Dado que es poco probable que varios paquetes consecutivos vean el mismo tamaño de cola no nulo, se consideró suficiente aplicar la condición para 3 paquetes consecutivos.

Una vez detectado en qué momento la cola estuvo vacía, se verifica que el q_k asociado no esté alejado del menor valor de la secuencia q_n.
Analizando este comportamiento para varios niveles de tráfico, llegamos al resultado de que una forma certera de determinar cuando la cola está vacía, es encontrar una secuencia de tres paquetes que verifiquen tener una distancia de tiempo de encolamiento (q_n) menor a 0,05ms entre ellos. O sea, que se cumpla:

$$| (tarribo_n - tarribo_{n-1}) - (tpartida_n - tpartida_{n-1}) | < 0.05 \text{ ms.}$$

Dentro de todos los q_n que cumplen las condiciones anteriores elegimos el de menor valor, así no obtendremos tiempos de encolamientos negativos.

Drift de relojes

En el análisis del tiempo de encolamiento de los paquetes se vio que hay diferencias entre los relojes, no sólo desfasaje sino también diferencias de velocidades o frecuencia. En este punto se nos presenta una problemática correspondiente a esta diferencia. No habría problemas, para este caso particular, si la diferencia fuera constante, el problema es cuando una máquina tiene el reloj más veloz que otra. Aquí vemos este caso particular en la siguiente gráfica:

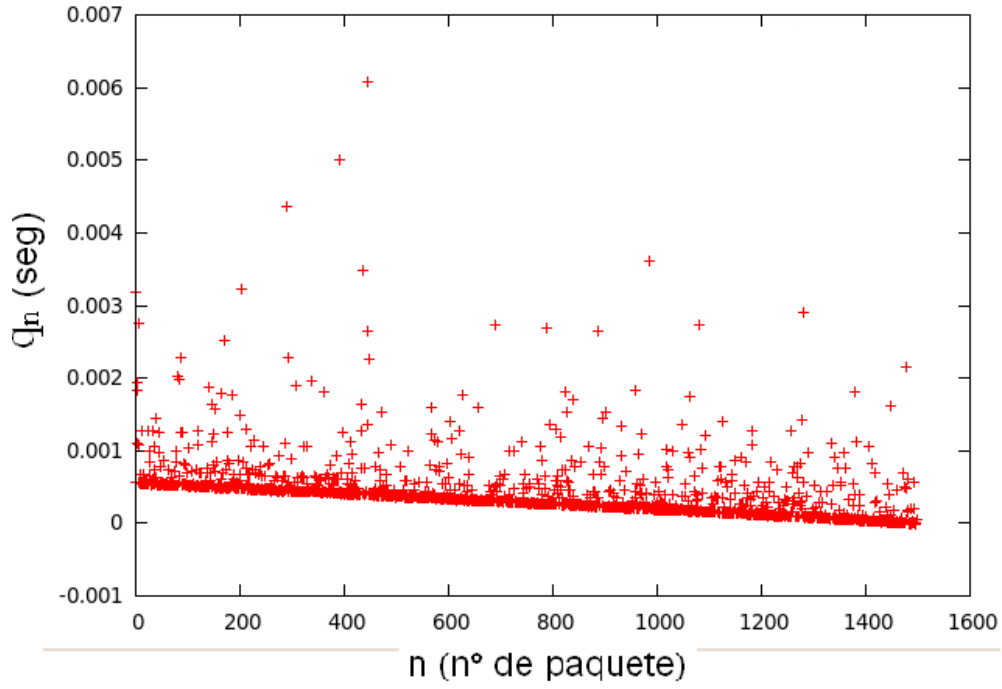


Figura 13

En la Figura 13 vemos el tiempo de encolamiento calculado (ordenadas) en segundos de los paquetes (abscisas) cuando el tráfico cruzado es nulo.
Vemos que la cola estuvo vacía todo el tiempo pero es notorio que el reloj del cliente es más veloz que el del ME. De esta manera, el tiempo entre arribos de dos paquetes consecutivos medido por el M.E. será ligeramente menor que el tiempo entre partidas medido por el reloj del cliente y por lo tanto esta diferencia se irá acumulando en la sumatoria vista en la Ec.(7). Como resultado, se obtiene en el gráfico de q_n *vs.* n. una recta con pendiente levemente negativa (debido a la diferencia aproximadamente constante entre medidas de los relojes) cuando debería ser una recta horizontal.

Cuantitativamente se observó que se alcanzaban errores acumulados de más de 0.5ms en 15 segundos en algunos casos.

En este caso también encontramos cola vacía y debemos detectarlo para todos los paquetes que lo estuvieron, para hallar luego el porcentaje de cola vacía como parámetro que caracteriza la red.

Una vez detectado q_i de referencia como valor en cola vacía con las restricciones anteriormente mencionadas, se puede determinar qué paquetes estuvieron en cola vacía, aunque no cumpla con las condiciones anteriores, simplemente sabiendo a qué distancia de tiempo de encolamiento estuvo del paquete en cola vacía de referencia. En este punto, el detalle de diferencia de velocidades de relojes afecta seriamente ya que se precisa determinar el porcentaje de cola vacía sobre una recta que decrece. Para contemplar este caso se realizaron pruebas con diferentes máquinas como M.E. y cliente. Se llegó a determinar que contemplar un valor de 0.5 ms de diferencia, cada 15 segundos, entre relojes del ME y un cliente era razonable. Entonces todos aquellos paquetes a una distancia de q_n menor al 1ms se consideran como paquetes de cola vacía. Este parámetro se pudo haber reajustado durante las pruebas para optimizar los resultados. Cabe mencionar que en cada ensayo se encontraron valores de drift diferentes lo que hizo imposible una compensación.

Para mejorar la sincronización, utilizamos el protocolo ntp para sincronizar las máquinas a servidores en Internet, llegamos a reducir hasta la mitad del valor de la diferencia marcada como límite. Surge la pregunta: ¿Cuantos paquetes debemos enviar para asegurarnos de encontrar la cola vacía cuando el enlace está a un 90% (en media) de su capacidad?

Nuevamente hicimos pruebas y generamos varios escenarios diferentes, utilizando el peor de los casos, o sea el video de más alta definición, corroborando llegar a un tráfico aproximadamente al 90 % (en media) de la capacidad del enlace. Llegamos a que 1500 paquetes eran suficientes para detectar cola en todos los escenarios de prueba. En la Figura 14 mostramos el tiempo de encolamiento para el caso descripto utilizando 1500 paquetes:

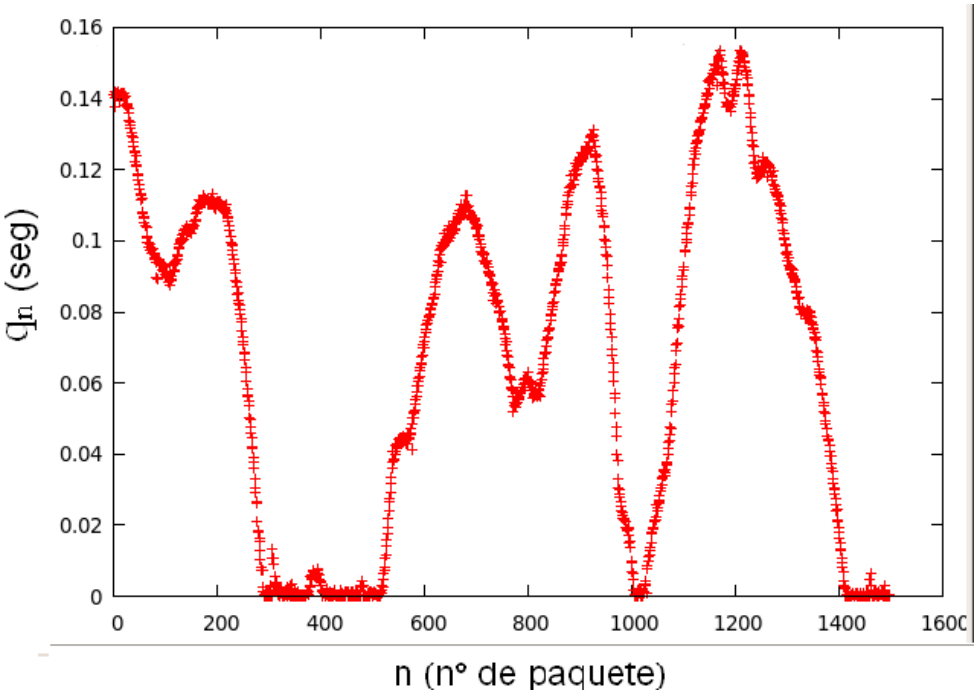


Figura 14

6.4.8.3 Cálculo de parámetros que caracterizan la red

Anteriormente definimos los parámetros que caracterizan la red y son q_n promedio, q_n varianza, q_n percentil y porcentaje de cola vacía.

Una consideración, el cálculo de q_n se realiza a través de la ecuación (7) y se puede observar que la resta entre tiempos se realiza con tiempos tomados del mismo reloj. Esto tiene la ventaja que si los relojes están desfasados por una constante no afecta el resultado, por lo tanto no se cometen errores por esta causa. Sí, se comete un pequeño error en el cálculo de porcentaje de cola vacía por diferencias de velocidades (drift) entre relojes como mencionamos en el punto anterior. Vimos diferencias de hasta 0.5ms de q_n entre el primer y último paquete; 15ms, por lo que decidimos contemplar esta diferencia para el cálculo de cola vacía.

Debemos mencionar también la existencia de otra fuente de error de la cual hemos reducido el impacto. Este es debido a que en el momento de envío de paquetes, tanto de prueba como de video, hay un tiempo entre que el sistema toma el tiempo de envío del paquete y realmente se envía el paquete. Entre medio hay un par de líneas de código que debería llevarle microsegundos en ejecutarse. El problema es que el sistema operativo puede estar ejecutando otros procesos de mayor prioridad afectando la exactitud de tiempos entre envíos de paquetes. Se pudo controlar este problema imponiéndole al proceso de envío de paquetes la prioridad máxima del sistema.

En las pruebas se observaron mejoras en el envío de paquetes y no tenerlo en cuenta implica enviar 1500 paquetes de pruebas en hasta 16seg en vez de 15seg, reduciendo el bitrate medio.

6.4.8.4 Cálculo de parámetros que caracterizan la calidad del video

Retardo, jitter y pérdidas de paquetes son los parámetros utilizados para determinar la calidad de video.

Estos tres parámetros relacionados a los paquetes del servicio, son un común denominador de los servicios en tiempo real para determinar la calidad del servicio. Se utilizará para el retardo y el jitter su promedio como referencia.

En nuestro caso el servicio es video bajo demanda (VoD) por lo que la variabilidad del retardo no tiene gran impacto en la calidad del video ya que el tráfico va en un sólo sentido, desde el servidor vlc al cliente. A diferencia de otros servicios de tiempo real como las conferencias o telefonía, donde el retardo es crucial para la determinación de la calidad del servicio. En VoD sólo implica un retardo en el comienzo de la visualización del video.

Una de las razones por la que se incluyó el retardo en la implementación es para poder reutilizar este sistema con otro tipo de aplicaciones. En definitiva el sistema encuentra una correlación entre parámetros de pequeños paquetes de prueba y los valores de retardo, jitter y pérdidas de cualquier secuencia de paquetes independiente de la aplicación.

Volcándonos directamente a los cálculos, se hallaron las pérdidas de paquetes a través de los números de secuencia asignados a cada uno de los paquetes udp en el cuerpo del mismo. Su cálculo es en porcentaje y se halla como la cantidad de pérdidas sobre la cantidad de enviados por cien.

Calculamos el jitter como la diferencia entre retardos consecutivos. Si reordenamos los términos nos queda que el jitter es la diferencia entre tiempos de arribo de los paquetes menos la diferencia entre los tiempos de partida.

O sea: $jitter^{n+1} = (tarribo^{n+1} - tarribo^{n}) – (tpartida^{n+1} - tpartida^{n})$

Vemos que el cálculo del jitter hace diferencias entre tiempos obtenidos del mismo reloj. Por lo tanto no tendremos errores por diferencias de sincronismo de relojes.

Se calculó el retardo como la diferencia entre el tiempo de arribo y el tiempo de partida de un paquete. Encontramos aquí un problema que hemos mencionado en varias

oportunidades en esta sección, que es la desincronización de relojes entre máquinas. Una solución a este problema es la sincronización a través del protocolo ntp a un servidor en común. Se implementó esta solución sincronizando la máquinas, con sistema operativo Ubuntu, al servidor "stratum 1" *a.st1.ntp.br* y *ntp.ubuntu.com* alcanzando desfasajes entre relojes de hasta 20ms.

Esta implementación tiene dos desventajas, la primera es que se debe estar conectado a Internet (o a algún otro servidor) en forma continua para que el protocolo ntp funcione en forma precisa ya que el mismo protocolo hace ajustes en forma gradual. La segunda desventaja es que encontramos diferentes valores de desfasaje en el correr del tiempo, esto es debido a reajustes que realiza ntp.

Para la validación del modelo utilizamos una solución alternativa con el fin de obtener resultados libres de afectaciones causadas por desfasaje de relojes. Se decidió calcular la diferencia entre relojes y restársela al retardo. El cálculo se hizo en forma simple la cual pasaremos a detallar.

Primero señalemos que el retardo de un paquete de un nodo a otro se compone por la suma del tiempo de servicio, el retardo del medio y el tiempo de encolamiento del paquete.

Bien, ahora si medimos el retardo con relojes desfasados podemos sumar el desfasaje de relojes al valor obtenido antes, quedando el retardo medido como la suma del tiempo de servicio, el retardo del medio, el tiempo de encolamiento del paquete y el desfasaje de los relojes.

Utilizando cables utp como medio a través de un switch, el retardo del medio es entorno de los 0.5ms. Este valor de retardo es despreciable frente a variaciones de desfasajes de los relojes que está entorno de los 20ms.

Cuando la cola está vacía, y despreciando el retardo del medio, el retardo obtenido está dado por el tiempo de servicio y el error de desfasaje de relojes, donde el tiempo de servicio es el tamaño del paquete sobre la capacidad del enlace. Si tomamos un paquete de 64bytes como los pp y un enlace de 14Mbps entonces el tiempo de servicio es de 0.036ms, por lo tanto lo consideramos despreciable frente al desfasaje de relojes.

De lo anterior podemos concluir que el desfasaje entre relojes es el retardo obtenido de un paquete de 64bytes cuando la cola está vacía. Dicho esto lo que hicimos fue encontrar la cola vacía en la secuencia de paquetes de prueba y luego calcular el retardo de dicho paquete. Así obtenemos la diferencia entre relojes de cada entrenamiento para luego restárselo a los retardos medidos en el mismo entrenamiento.

Hay que mencionar que aquí se estaría cometiendo un error para los casos en que el retardo del medio (propagación) sea comparable a los 20ms de desfasaje. Para estos

casos recomendamos un sistema de sincronización más preciso. Además debemos tener presente que se está sustituyendo el retardo por el tiempo de encolamiento.

6.4.8.5 Herramienta de Support Vector Machine (Lib-svm)

El libsvm [12] es la herramienta elegida para encontrar la función que relacionará los parámetros de los paquetes de prueba y los parámetros de calidad de video.
A este programa se le debe pasar el archivo que contiene los datos de cada instancia. Tal archivo debe respetar un formato para que se pueda interpretar correctamente cual es el parámetro de interés, o sea a predecir, y cuáles son los atributos que utiliza para la predicción. El formato es el siguiente:

<label> <index1>:<value1> <index2>:<value2>

. . . . .

. . . . .

<label> <index1>:<value1> <index2>:<value2> (línea numero 300)

Donde label es el objetivo del experimento; el parámetro a entrenar del video, e índex son los atributos; los parámetros de los pp. Debe crearse un archivo por cada parámetro de video a entrenar, en nuestro caso son 4, q_n promedio, q_n varianza, percentil q_n y porcentaje de cola vacía.
Luego, como se recomienda en la documentación del software, se normaliza la muestra para que los valores de los atributos estén entre (-1,1), creando un nuevo archivo con extensión .scale.
Ejemplo para el retardo:
svm-scale -l -1 -u 1 archivoEntrenaRetardo > archivoEntrenaRetardo.scale

Una vez construido el archivo con todas las instancias utilizamos la función training donde el libsvm creará un modelo de correlación, el cual utilizará para predecir los futuros valores de parámetros de video de interés
Para correr la fase de entrenamiento se deben encontrar los parámetros apropiados con los cuales lib-svm construirá su modelo. Podrán ser diferentes parámetros para el retardo, jitter y pérdidas. Mostramos un ejemplo para hacer una idea:

./svm-train -s 4 -t 2 -g 0,03215 –c 2 –b 1<archivo_muestras_Train_normalizado>

Donde –s es el tipo de svm, en nuestro caso es una regresión nu-SVR (el tipo de SVR fue hallado viendo cual ajustaba mejor los resultados), luego –t elige el kernel de la

regresión: radial basis function: exp (-gamma*|u-v|^2). El –g es el parámetro gamma del kernel elegido y se encontró minimizando el error cuadrático medio.
El train producirá un archivo llamado .scale.model que será el modelo a utilizar por el libsvm para predecir. Luego de ejecutado el train se procede a realizar la predicción:

./svm-predict -b 1 <archivo_dePredict_normalizado> Salida.scale.model output_file

Se genera una salida output_file para cada parámetro de video donde se guardan los valores predichos del retardo medio del video, jitter medio del video y pérdidas de paquetes.

6.5 Programa principal

6.5.1 Generalidades

Hasta aquí hemos implementado las funciones que son capaces de entrenar a un cliente y que nos permiten generar un modelo que relacione los parámetros que cuantifican el estado la red con los parámetros de calidad de servicio de video. Luego con tal modelo podremos predecir el comportamiento de la red.
Lo que resta para tener un M.E completo, es encontrar la manera de automatizar el funcionamiento de esos métodos incluyéndolos en un programa principal que pueda discernir los distintos estados de la red y de los clientes.
Por lo tanto el Programa Principal (ProgPal de aquí en adelante) será el encargado de manejar el uso de los métodos entrenar y predecir, decide cuando y como ejecutarlos y luego guarda los resultados para cada cliente. También elige cuales son los valores válidos que se le pasará por SNMP al Proxy, con los que éste último decidirá si acepta la petición de un cliente para acceder a un video.
En esta sección describiremos el funcionamiento del programa y los criterios con los cuales de diseñó.

6.5.2 Condiciónes de inicio y algoritmo de recorrido

El ProgPal deberá guardar los datos de los clientes y los valores obtenidos de sus últimos entrenamientos y predicciones.
Entonces cuando se corre el programa por primera vez, obtendrá los datos iniciales de cada cliente que forma parte de la red, recogiéndolos de un archivo inicial (llamado “Inicial.txt”).

Allí se encuentra la ip del cliente, la hora del último entrenamiento HD, del último entrenamiento SD, de la última predicción, los valores del último entrenamiento(delay, jitter, pérdidas HD y SD) y los valores de la última predicción (delay, jitter, pérdidas HD y SD).

Si bien inicialmente todos esos valores y horarios se inicializan en cero (excepto la ip que para nosotros es la única forma de identificar al cliente), este archivo sirve también para que el ProgPal pueda ir modificándolo y tengamos disponible un reporte del estado de los clientes.

Una vez que ya tiene cargados los clientes, ProgPal empieza a recorrer a los mismos uno por uno y en cada uno de ellos realizará un control sobre qué tareas corresponde realizarle.

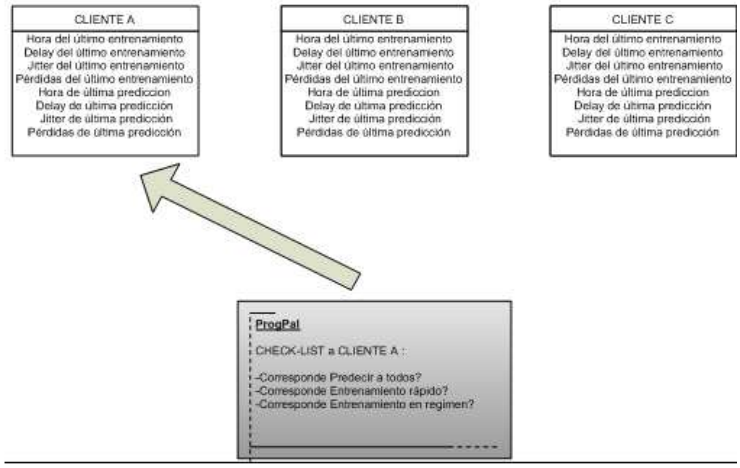


Figura 15

En la **Figura 15** se muestra la forma de recorrer el array de clientes y muestra tres clientes hipotéticos que servirán para fijar ideas:"ClienteA", "clienteB" y "clienteC".

6.5.3 Estados posibles que adopta un cliente en el sistema

Supongamos que el ProgPal se encuentra en pleno recorrido de clientes y llegamos al "ClienteA". Ahora el programa debe verificar qué tarea corresponde realizarle y para ello definimos distintos estados en los cuales un cliente puede estar y que clasifican las distintas tareas a realizarle. Los posibles estados son los siguientes:

Cliente en entrenamiento rápido

Si el "ClienteA" ha ingresado recientemente en la red de distribución de video, debemos crear un modelo de predicción confiable. Para ello debemos tener una cantidad de instancias suficientes sobre distintos estados de la red para que el svm pueda encontrar un modelo de predicción al "ClienteA" que sea preciso.
Basándonos en las simulaciones (capítulo10) y también en las pruebas realizadas con maquetas reales (capítulo 11) definimos una cantidad de instancias mínima que debe contener el svm para que comencemos a tomar en cuenta la salida del modelo. Si no tenemos esa cantidad de instancias de entrenamiento al "ClienteA" no se puede asegurar que la predicción a ese cliente sea certera.
Es importante considerar un tiempo entre entrenamientos que permita realizar un barrido sobre distintos estados de la red.

Una vez alcanzada la cantidad de entrenamientos mínimo, se verifica que la correlación entre la salida del modelo y el valor real del parámetro esté por encima del 70%.

Entonces, el ProgPal compara la cantidad de entrenamientos que se le realizaron al "ClienteA" con el umbral definido anteriormente y en caso de estar por debajo la única tarea que le realizará al "ClienteA" es entrenarlo periódicamente.
Al ser este modo un estado transitorio, en el que el modelo de predicción no es válido, se entrenará con un periodo entre entrenamientos más rápido de forma de obtener un modelo lo antes posible, pero lo suficientemente grande para encontrar diferentes estados en la red. Esta forma de entrenamiento le llamamos modo de ***entrenamiento rápido***.

Habiendo dicho lo anterior, se entiende que hasta no alcanzar el umbral el programa no predecirá al "Cliente A" ya que de hacerlo no sería confiable. Por lo tanto la manera de poder aceptar o no la petición al "Cliente A" cuando se encuentre en el modo de ***entrenamiento rápido*** es utilizando únicamente el último entrenamiento válido.

Cliente en entrenamiento en régimen

Cuando hemos podido entrenar al "Cliente A" lo suficiente como para alcanzar el umbral, consideramos que ha terminado la etapa de ***entrenamiento rápido*** e ingresa en lo que llamamos el modo de ***entrenamiento en régimen.*** En este modo el cliente contiene suficientes entrenamientos como para que se pueda crear un modelo aceptablemente confiable por lo que podremos usar el método *predecir* para estimar el estado de la red y aceptar o no solicitudes de clientes. De todas maneras se debe seguir entrenando al cliente para mejorar la confiablidad de las predicciones y poder llegar a valores de correlación razonables.

Cliente en Predicción a todos

Supongamos que el "clienteA", el "clienteB" y el "clienteC" se encuentran en el modo ***"entrenamiento en régimen".*** Digamos que el ProgPal se encuentra casualmente entrenando a "ClienteB" cuando en cierto instante al "ClienteC" se le ha aceptado una petición a un video. En ese momento el ProgPal termina el entrenamiento a "ClienteB" y comienza a recorrer los clientes desde el primero para predecir sus comportamientos ante el nuevo escenario (un video más en la red). Hasta que el programa principal no termine de predecir a todos, cada cliente de la red se encontrará justamente en el estado llamado ***Predicción a todos.***
Hay que aclarar que si un cliente se encuentra en la etapa de ***entrenamiento rápido,*** entrará en la etapa ***Predicción a todos*** pero no se le ejecutará una predicción sino que se le entrenará cuando el programa llegue a él. Esto se debe a lo antes dicho sobre la invalidez del modelo cuando no se pasó el umbral de entrenamientos mínimo.

Se aclara una vez más, que esta política de manejo de predicciones es válida bajo la hipótesis tomada en **4.3.3** la cual garantiza que si la predicción al "ClienteA" da valores que se encuentran dentro de lo establecido para tener buena calidad de video mínima tampoco se verá afectado a otro cliente con el agregado de tráfico que el nuevo video implicaría. Entonces la razon primordial de la predicción a todos es el tener

predicciones “frescas” para todos los clientes.

Cliente en verificación del modelo

Cuando el cliente entra en este estado, el ProgPal realiza una verificación del modelo utilizado para sus predicciones de calidad. Cada parámetro (Retardo, Jitter y pérdidas) utiliza un modelo creado y periódicamente se debe verificar su validez. Para ello se realiza primero una predicción y luego un entrenamiento y se comparan los resultados, valor medido vs valor predicho. En este caso, a diferencia de entrenamiento en régimen, el entrenamiento se realiza si o si, sin verificar si afecta a otros clientes o no. Esto es debido a que las verificaciones deben realizarse en cualquier estado de la red. La condición de aceptación de los modelos es que el error de la predicción esté acotado por un valor configurable para cada parámetro de calidad.

Para descartar completamente el modelo, el ProgPal espera encontrar tres veces consecutivas predicciones con errores mayores a los establecidos. Se asume que el modelo será siempre más certero en el futuro que en el presente ya que se agregarán más entrenamientos.

6.5.4 Control de Entrenamiento y Predicción.

Tiempos entre entrenamientos y predicciones

Cada tarea que el ProgPal le realiza a un cliente sea entrenamiento rápido, predicción para entrenar, entrenamiento en régimen se va ejecutar de acuerdo al tiempo en que se ejecutó la última de cada una de ellas. O sea cuando el ProgPal analiza que corresponde realizarle a un cliente, verifica primero en qué estado se encuentra pero luego hará un control sobre la hora del último entrenamiento SD y HD, la última predicción y en base a eso decide que hacer.

Los siguientes son los tiempos con los que se realizan las tareas:
periodEntren= 3600 segundos, periodo entre los entrenamientos en régimen 1hora.
periodRapEntren=1800 segundos, periodo entre los entrenamientos en el transitorio cuando no hay suficientes entrenamientos (30 minutos)
periodPred= 120 segundos, período entre predicciones individuales.
limUsoentrenamiento=60 segundos, límite para usar el último entrenamiento y no usar la predicción.

Como vimos en secciones anteriores, la ejecución del método entrenar a un cliente, implica enviar un tráfico similar al de un video, con lo que se podría estar deteriorando la calidad de un video en reproducción hacia otro cliente en la red o al mismo cliente a entrenar.
Por esta razón es que si ProgPal debe entrenar a un cliente que se encuentra en modo ***entrenamiento en régimen*** primero verifica si lo “puede” entrenar. Esto se hace ejecutando el método predecir y en base a la predicción (que nos arrojará los valores de delay, jitter y pérdidas predichos) se decide si se puede ejecutar el entrenamiento o no.

Evidentemente dicho procedimiento para ejecutar un entrenamiento(o sea predecir y luego entrenar) solamente se realiza si el cliente no se encuentra en el modo

entrenamiento rápido ya que en tal caso la predicción no es confiable.
Repaso del funcionamiento para ClienteA:

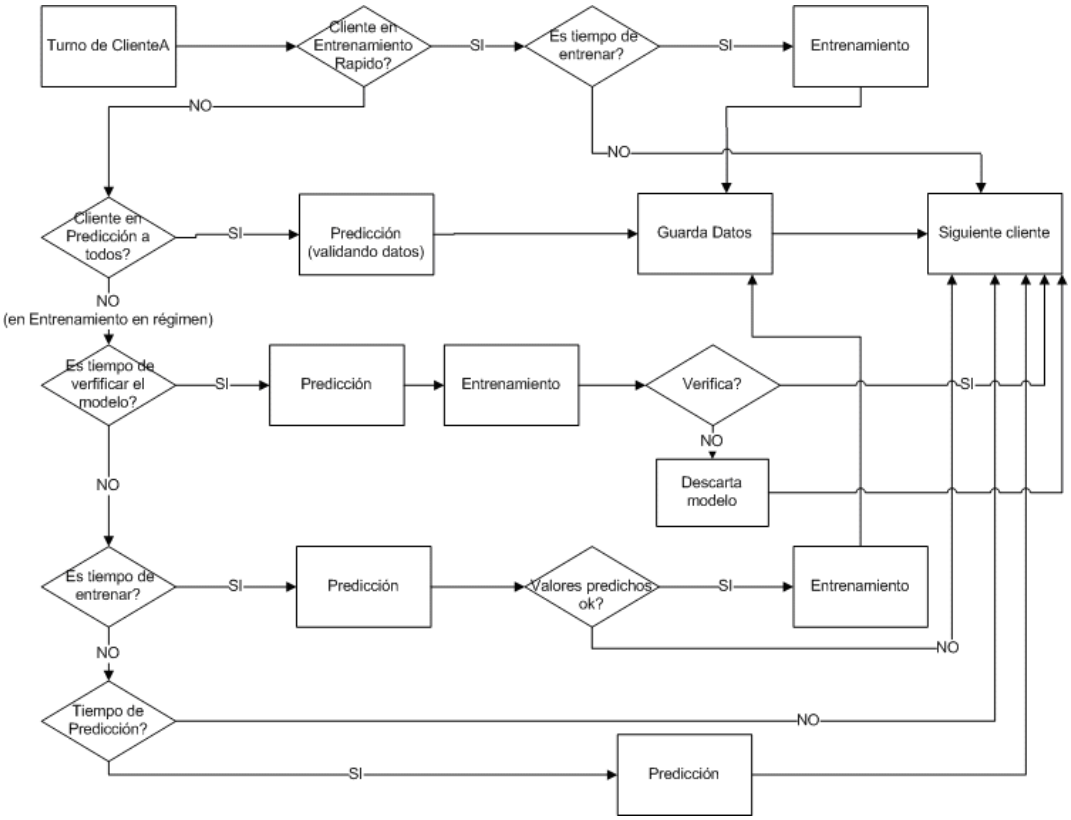


Figura 16

El ProgPal realiza tareas en paralelo cuando está entrenando o prediciendo a un cliente, de manera de poder suspender todas las actividades cuando una solicitud es aceptada y los valores predichos con anterioridad se tornan inválidos.

En la **Figura 17** se detalla el algoritmo:

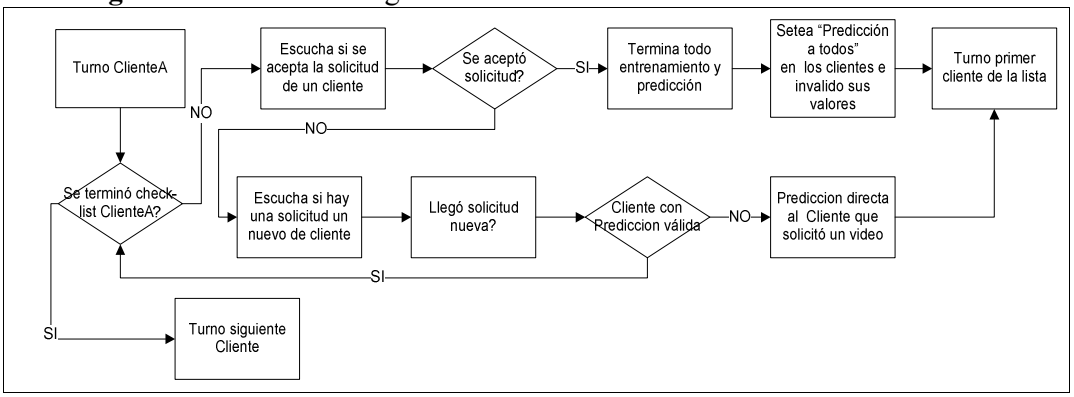


Figura 17

Observamos que en el diagrama se muestra cómo se comporta el ProgPal cuando se

acepta un cliente. Destacamos que cuando un entrenamiento está siendo ejecutado y se aceptó una solicitud, el ProgPal toma la decisión de cortar abruptamente el entrenamiento solo si está enviando paquetes ya que el entrenamiento pudiese no ser válido de ser afectado por el nuevo tráfico de video y además el video que acaba de empezar a reproducirse puede deteriorarse por el tráfico del entrenamiento en curso. En caso de haber terminado de enviar sus paquetes, el método entrenamiento no se interrumpe y puede terminar de procesar los datos de un entrenamiento válido.

6.5.5 Valores válidos a pasar al Proxy

Una vez en régimen, se diseñó para que a los clientes se les entrene muy poco y en cambio se les prediga mucho por lo dicho sobre los livianos que los "pp" son y para poder tener una predicción lo más fresca posible cuando exista una solicitud de un cliente.
De todas maneras, siempre que el último entrenamiento se haya realizado en un lapso de tiempo prudencial con respecto a la última predicción, preferimos quedarnos con el entrenamiento por razones obvias(es claramente más precisa la realidad que la predicción de la misma). Si el último entrenamiento se realizó hace más de 60 segundos, consideramos que tal entrenamiento no refleja el verdadero estado de la red en el momento actual y optaremos por la última predicción.
No se predice al momento de la predicción porque se considera que no es lo mejor que el cliente espere (lo que tarde la predicción) para obtener una respuesta.
Hay también otro caso en el cual la predicción no es válida y es cuando se aceptó la solicitud de un cliente y el programa entró en la etapa predicción a todos. En ese momento todos los clientes que todavía no han sido predichos no tienen valores válidos para pasarle al Proxy y este debería esperar a que se prediga al cliente solicitante para atender su petición.

Entendemos entonces que una funcionalidad extra que enriquece en versatilidad al programa es que se puedan aceptar solicitudes de clientes que se encuentran en ***Predicción a todos.***

Digamos que el programa se encuentra prediciendo a un "clienteA" y llega una nueva petición al "clienteB". Teniendo en cuenta que el estado de la red es otro al que se encontraba el ClienteB cuando se predijo por última vez, ¿Cual debería ser la forma de proceder de ProgPal?
En un principio se pensó que lo mejor era que el ProgPal esperara a predecir a todos los clientes antes de dar como válidas las predicciones de cada cliente. Si bien era una forma de asegurarme que ningún cliente iba ser aceptado con predicciones "viejas", era muy poco eficiente desde el punto de vista de manejo de clientes, ya que el "ClienteB" debería esperar a que se terminen todas las predicciones. Teniendo en cuenta que cada predicción son aproximadamente 15 segundos, esa demora se torna excesiva cuantos más clientes tenga en la red por lo que decidimos optar por otro enfoque.

El enfoque que se eligió al final tuvo fundamento en una de las hipótesis tomadas a lo largo de todo el proyecto y es que si el resultado de la predicción a un cliente da valores que satisfacen los parámetros de calidad para el tipo de video con requerimientos más restrictivos, entonces no solo ese cliente verá el video de la mejor manera sino que ningún otro cliente se verá afectado.
Teniendo en cuenta esa hipótesis fundamental fue que decidimos el comportamiento

que tendrá el ProgPal cuando se encuentra en Predicción a todos y llega una nueva petición de un cliente que no se predijo aún. Lo que hará es cortar la predicción actual y predecir al cliente que hizo la petición. Si el resultado de la predicción es aceptable, entonces el cliente comenzará a ver su video y el ProgPal comenzará nuevamente a predecir a todos. Si en cambio la solicitud es rechazada por malos resultados, el ProgPal continua el barrido en el que estaba.
Esto permite que el mayor tiempo de espera de un cliente luego que aceptó a otro sea el tiempo que demora en realizar una predicción (15 segundos).

El ProgPal está diseñado para que corte la predicción actual y prediga directamente al "ClienteA" de forma de reducir el tiempo de espera de ese cliente.
De lo contrario el Proxy debería esperar a que se prediga a todos los clientes antes de saber si puede o no aceptar la solicitud de un cliente aún cuando esta solicitud pueda ser denegada por una predicción con valores encima de los valores permitidos.

6.5.6 Entrenamiento inicial

Como ya dijimos, cuando el cliente se encuentra en ***entrenamiento rápido,*** no se realiza una predicción antes de entrenar por lo que no sabemos el efecto que tendrá tal entrenamiento sobre el cliente actual ni tampoco sobre otro cliente.
Si bien el sistema que hemos construído se basa en la premisa de no comprometer la calidad de servicio de ningún cliente que forma parte de la red de distribución, hemos dispuesto como inevitable que se corra el riesgo de deteriorar la QoS de un video cuando algún cliente se encuentra en ***entrenamiento rápido,*** ya que es el costo a pagar para poder llegar a crear un modelo de predicción correcto lo antes posible.
Tal vez se pueda pensar que existen formas de evitar tal degradación del servicio, ya que otra hipótesis tomada en nuestro proyecto es el total control del tráfico cursado por la red.

Sin embargo, en la creación de un correcto modelo de predicción es fundamental el entrenamiento de la red en variados escenarios, con lo cual sería muy poco productivo ejecutar entrenamientos con la red sin tráfico o muy lejos de la saturación de los distintos enlaces.
Lo que se quiere decir es que la degradación del video de los clientes, es en algún punto deseable ya que un entrenamiento realizado en condiciónes próximas a la saturación nos aportaría muy buena información sobre el estado de la red.
Pongamos un ejemplo bien concreto de lo queremos decir. En el caso del porcentaje de pérdidas de paquetes, si el ProgPal no entrenase nunca cuando el enlace esta cerca del nivel de saturación el modelo creado sería muy pobre ya que todas las instancias de entrenamiento arrojarían 0% de pérdidas. Pasa algo similar con el resto de los parámetros, aúnque claramente en menor medida.

Entonces, nos encontramos ante un compromiso entre la riqueza de información de cada entrenamiento y la garantía de conservar la calidad de servicio de un video en reproducción.
Ante este compromiso, resolvimos que exista un procedimiento, llamado EntrenamientoInicial, que sea externo al ProgPal. Este procedimiento es un programa que genera distintos escenarios de tráfico hacia un cliente y ejecuta un entrenamiento en cada escenario.

6.5.7 Funcionalidades de seguimiento para administrador

Aúnque ya lo habíamos mencionado, el ProgPal sobrescribe periódicamente (cada vez que termina de recorrer la ronda de clientes) un archivo con los valores de las últimas predicciones y entrenamientos de cada cliente. También escribe si el cliente se encuentra viendo un video y cuál es el valor que el Proxy recogerá con su consulta SNMP.

Por otro lado el ProgPal escribe otro archivo de logs del programa. En él, se destacan las acciones mas importantes que realiza el Progpal y también los errores o impedimientos para realizar acciones.
Alguno ejemplos de logs:
" Predicción a todos por ingreso de cliente 192.168.1.4"
"Se entrenó a 192.168.1.2 para HD y SD"
"Entrenamiento rápido HD para 192.168.1.3"
"Fallo de sesión TCP con 192.168.1.4"
" No se encontró q_o para 192.168.1.2, enlace satura"

6.5.8 Pequeños detalles de implementación

Durante la implementación del ProgPal se utlizaron algunas herramientas de C que vale la pena destacar.
Para almacenar los datos de cada cliente se utilizó un tipo de dato de C[16][17], llamado estructura. Es el tipo de dato que es la antesala de lo que en Java se llama objeto. Cada estructura contiene aributos y también puede contener otras estructuras.
A su vez, para agrupar todas las estructuras, en primera instancia se pensó en utilizar los conocidos array-list pero como la cantidad de clientes es variable y crear un array-list muy grande es ineficiente desde el punto de vista de manejo de memoria (en lo cual C es muy exigente). Se utilizó en cambio "asignacion de memoria dinámica" (memory allocate) con lo cual se guarda la dirección de un lugar en memoria del primer cliente y luego se van agregando clientes con esa referencia y también referenciando al siguiente cliente. Por esto último,cada cliente contien una estructura que referencia al cliente siguiente (llamada next) con lo cual si next es NULL quiere decir que terminé de recorrer la lista de clientes. Esto permite una lista variable que sólo contenga los clientes de la red y no se introduzca basura en la memoria reservada para un array-list.

Por otro lado, como se notó en 6.5.4 el ProgPal realiza tareas en paralelo. O sea realiza el chequeo de tareas a realizarle a un cliente y por otro lado escucha por si el Proxy le avisa que se aceptó una solicitud o si se hizo una nueva solicitud si el sistema està en ***Predicción a todos***. Esto lo hace gracias al comando "fork" que genera un proceso hijo, un proceso que tiene el mismo codigo que el "padre" hasta el punto de creaciòn y que luego se puede diferenciar de él por su "process id". Ahora el padre y el hijo pueden realizar distintas tareas. El padre tiene la posibilidad de matar al hijo en caso conveniente, por ejemplo si se acepta una solicitud y se entra en ***Predicción a todos.*** Para que el tráfico que puede estar generando el hijo con un entrenamiento se decide matarlo y empezar a predecir al primer cliente como se explicò en secciones anteriores.
El "fork" [18] es el simil de "threads" en JAVA.
Suponiendo que el ProgPal se encuentra con el "ClienteB" y corresponde que se le entrene, el hijo entonces tiene la potestad de llamar al método "Entrenar" hacia ese

cliente y luego modificar las variables de ese cliente cuando finalice el entrenamiento.
Entonces como hijo y padre en realidad funcionan como procesos independientes, debe haber una forma de qué ambos escriban en la misma variable compartida [19]. Se utilizaron librerías de C que lo permitian (sys/ipc.h y sys/shm.h).
También se utiliza memoria compartida con las variables que manipulan los script del servidor SNMP para avisar que la solicitud a una ip específica ha sido aceptada.

6.5.9 Conclusión

Existen prestaciones adicionales que se le pueden agregar a ProgPal que lo harían más completo aún, pero por razones de tiempo fueron imposible implementarlas(se puede ver ejemplos en trabajo a futuro). Se considera de igual forma que diseñó un ProgPal ágil y bastante simple que resuelve situaciones para que el sistema funcione correctamente.

7. Implementación de Proxy

7.1 Introducción

El proxy es el encargado de captar las peticiónes del cliente y hacer la consulta al M.E sobre los parámetros actuales para así decidir si se acepta o no la petición.
Se decidió que lo más conveniente era que la interfaz hacia el cliente del servidor de video fuera una página web por ser la aplicación más común hoy en día. Con esto último definido, el Proxy debe captar las peticiónes http del cliente, entonces se comenzó a estudiar la utilización de un proxy conocido y ampliamente aceptado que se pudiera adaptar a nuestras necesidades. El proxy objeto de estudio fue el Squid.

7.2 Descripción del funcionamiento

Después de un detallado estudio del Proxy Squid [20], probando diferentes versiones desde la 2.3 a la 2.7 [21], se entendió el funcionamiento en forma cabal y se pudo asegurar que el Proxy cuenta con las herramientas necesarias como para asociarlo a otros módulos que le hagan tomar dinámicamente una decisión sobre una petición del cliente.

Para lo anterior hubo que estudiar el lenguaje C [17], utilizado por la versión elegida de Squid (2.7-estable-3) y se estudiaron algunos módulos con mayor atención.

Squid incluye una funcionalidad llamada “External Access Control Lists” que es implementada en un proceso externo (“helper process”). Al llegar una petición del cliente squid escribe ciertos parámetros (previamente establecidos) en la entrada estándar, el proceso externo lee estos parámetros y en función de ellos responde con “OK” o “ERR” en la salida estándar.
Si la respuesta es “OK” Squid otorga la petición y si es “ERR” la rechaza.

Uno de los External ACL que vienen en el código fuente de squid es el “ip_user”. Este programa recibe (de la entrada estándar) la IP del cliente y nombre de usuario como entrada, y chequea estos dos parámetros contra un archivo de configuración (“ip_user.conf”) para decidir si se debe dar acceso o no.
Si la combinación no es válida, o bien no aparece en “ip_user.conf”, retorna “ERR” y de lo contrario retorna “OK”.

Se estudió el código fuente de la funcionalidad “ip_user” y se adaptó el mismo según nuestros requerimientos.

Adaptación de “helpers process”

En el archivo de configuración de Squid definimos nuestro ACL Externo para que escriba en la entrada estándar la ip de origen del cliente y definiendo la ruta donde correr nuestro proceso.

7.2.1 Primera etapa del helper

El orden cronológico con que se avanzó en el proyecto implicó que para probar el funcionamienteo de los Helpers, se procediera a la construcción de un módulo estadístico provisorio que midiera de alguna forma el enlace entre cliente y servidor para tomar una decisión sobre la petición del cliente.

Lo que se hizo fue un script que ejecutaba el comando ping de linux e iba guardando el resultado de los últimos 5 pings. Guardaba el promedio de RTT (round trip time) y el porcentaje de pérdidas.

Nuestro proceso prueba_helper lee el promedio RTT y el porcentaje de pérdidas para luego retornar "OK" o "ERR" y Squid toma la decisión de acceso del cliente.Los umbrales elegidos como prueba fueron 1% de perdidas y 300ms de RTT.

Cada vez que Squid recibe una petición del cliente, éste pasa a la entrada estándar la ip del cliente para que lo tome el prueba_helper. Luego el prueba_helper lee el RTT y porcentaje de pérdida de los últimos 5 pings.

Squid guarda el resultado (acceso permitido o denegado) por un determinado tiempo configurable TTL, que por defecto es 60 minutos.

Para lograr que el control de acceso sea dinámico y responda a variaciones del RTT y porcentaje de perdida (estado del enlace), debimos setear el valor TTL con un valor pequeño (1 seg). Esto quiere decir que el Proxy no consulta al helper cuando recibe solicitudes consecutivas separadas de menos de un segundo. Si eso sucede acepta o deniega de acuerdo al ultimo OK o ERR.

También se agregó el valor children=40, que permite que el squid ejecute hasta 40 procesos hijos del programa prueba_helper.
De otra manera el squid es capaz de terminar abruptamente si llega a sobrepasar la cantidad de procesos que puede manejar. El valor por defecto es children=5, lo que parece un límite demasiado bajo para la cantidad de instancias de prueba_helper simultaneas.

Resumiendo, si el squid recibe una petición para acceder a un contenido le pregunta al módulo el resultado de esos pings y en base a eso acepta o no.

La finalidad de este sistema provisorio era probar el control de acceso realizado por squid y la interacción entre éste último y procesos externos al mismo.

7.2.2 Helper Definitivo

Una vez probado el squid y su relación con procesos externos, proseguimos a realizar un Helper más sofisticado.
Se buscó que el Proxy tenga la habilidad de rechazar o aceptar una petición de un cliente en base a primero al contenido elegido por el cliente (asociado directamente a una calidad de video que el cliente quiere obtener), y segundo en base a si es posible

ofrecérsela con el estado actual de la red.

Por lo tanto el Proxy debe realizar las siguientes funciones:

1) Recibir ip de origen y la URL destino en base al video elegido desde una página web.
2) Consultar al M.E por la ip de origen (que el M.E entiende como cliente) para aceptar o no la petición.

Construída una página web con los contenidos, el cliente debe clickear lo que pretende ver y el Squid, que escucha todas las peticiones http que lo tienen como proxy, puede ver el origen y destino de las mismas.
Por lo tanto la primera función del Proxy, la realiza el Squid pasándole al Helper: la ip de origen de la petición y también la url destino que es la manera en que sabremos cual contenido y calidad que ha elegido el usuario.

Con esa información el Helper consultará al Módulo Estadístico para obtener la información necesaria para tomar la decisión final.

7.2.3 Funcionamiento práctico del Helper

Para que el proxy realizara las tareas necesarias, se debió construir un helper que ahora se llamó ipUrl.
IpUrl recibe la ip del cliente y la url destino. De la url seleccionada por el usuario en la interfaz web, se extrae la información sobre el video solicitado y su calidad asociada.
Ésta información será utilizada por el sistema para decidir si se debe dar acceso o no al usuario.

Conociendo además la ip del cliente que realiza la petición, se consulta al módulo estadístico sobre la calidad de video que se obtendrá una vez que se comience a enviar a través de la red.

Dicha consulta se hace a través del protocolo SNMP, instalando un cliente en la computadora con el Proxy y un servidor en la computadora con el M.E (la interacción entre módulos se detalla más adelante en la sección *Interacción entre módulos*)
Cuando se hace la consulta, el Proxy pasa como argumento la ip del cliente y la calidad requerida por el usuario.
Al recibir la respuesta del agente snmp, los valores son guardados en un archivo desde donde el ipUrl lee el delay, jitter y pérdidas que se predijo que el video tendrá.

El helper ipUrl compara los valores obtenidos de la consulta al módulo estadístico con los valores deseados para la calidad que el cliente elegido y si no se satisfacen los requerimientos el helper rechaza la petición.
En el mensaje de error que el cliente ve en su explorador se solicita que intente más tarde, o se ofrece una calidad inferior en el caso de que los parámetros obtenidos indiquen que es posible garantizarla.
Si en cambio, la solicitud del cliente es aceptada, por un lado se informa al usuario mediante la interfaz web y el Helper envía un comando al servidor para que le sirva el video al cliente. El Proxy tambien envía un comando al cliente para que éste levante el video en determinado puerto. Esto último es posible porque la aplicación alojada en el cliente escucha un determinado puerto y es capaz de ejecutar el comando cuando el

helper se lo envía.

7.3 Servidor Web Apache

Al haber decidido que el cliente acceda a los contenidos mediante una página web que le ofrezca las distintas opciones, debimos instalar un servidor web y una página de acceso a los contenidos.

Para eso levantamos un servidor web (el utilizado fue Apache, versión 2.2.14) y realizamos una modesta pagina de presentación, escrita en lenguaje html básico, donde podrá aparecer la oferta de videos.

Figura 18

Así cuando el cliente elija una opción el Proxy captará su petición http y elaborara las consultas correspondientes. Para ver los archivos html ir al anexo.

7.4 Criterios de control de acceso

Como se explicó en la arquitectura, es en el Proxy donde se definen los parámetros límite de QoS y es el módulo mismo quien toma la decisión final. El criterio para la decisión es muy simple. Si los valores predichos de retardo, jitter y pérdidas son menores a los valores críticos impuestos por el administrador del sistema, entonces es permitido el acceso. En caso de que uno de estos no cumpliera la condición anterior, entonces el acceso será denegado. Los parámetros que tomamos como límite se verán en el capítulo 9.

7.5 Conclusión

Se encontró en Squid, un programa muy versátil que encajó perfectamente en lo que se buscaba: un programa con solido funcionamiento y vasta documentación,que se adaptó a nuestras necesidades con la implementación de un Helper Proccess capaz de tomar la decisión sobre la admisión o no de la solicitud de un cliente.

8. Interacción entre módulos

Para implementar la interfaz entre el M.E. y el Proxy se decidió utilizar el protocolo SNMP [22] ya que permite que el proxy pueda consultar al M.E sobre los parámetros de QoS de interés para realizar un control de acceso al sistema y además es un estándar ampliamente usado, lo cual contribuye a la posibilidad reusabilidad o interoperabilidad entre distintas implementaciones de los módulos.

En consecuencia, se instaló en el Proxy un cliente SNMP (conjunto de herramientas que permite realizar las consultas) y en el módulo estadístico un servidor (agente) SNMP que escucha las consultas a una MIB donde se aloja una tabla con información de todos los clientes.
Cuando el agente recibe una consulta por un cliente particular, devuelve los últimos valores de delay, jitter y pérdidas para ese cliente.
La respuesta a la consulta por los parámetros de calidad, se implementó mediante un script, que toma de un archivo la información provista por el programa principal del M.E. Luego envía a la salida estándar la información recogida. Estos valores son tomados por el agente snmp para contestar la solicitud del proxy.
Se utiliza también la comunicación via SNMP para que el Proxy avise al M.E. cuando se ha aceptado a un nuevo cliente para que comience rápidamente a realizar predicciones a todos los clientes para el nuevo estado de la red

Por otra parte, fue necesario implementar la comunicación entre el agente SNMP y el M.E. ya que en nuestro caso estarán corriendo en el mismo equipo pero como dos procesos independientes.
Para esto se utilizaron mecanismos de IPC (Inter-process Communication) que permiten el intercambio de datos entre uno o más procesos y eventualmente entre distintos hilos de cada proceso.

Existen distintos métodos dentro de estos mecanismos y se usó el método de memoria compartida. Esto permite a un proceso compartir un bloque de memoria, asociarlo a una variable y que otro proceso pueda acceder a dicha variable.
Dependiendo de los permisos otorgados por el proceso que comparte la variable, los demas procesos podrán leerla o sobrescribirla.
En nuestro caso utilizamos este mecanismo para que el agente SNMP pueda setear (mediante un script) variables en el programa principal del M.E. que indiquen que un usuario ha realizado un petición, o que ha ingresado un nuevo usuario y específicar que usuario ingresa.
Además, se utilizaron variables compartidas entre proceso hijos (hilos) y padre del programa principal para el manejo de predicciones y entrenamientos de los clientes (ver seccion 6.5.8) y la actualización de los últimos parámetros de QoS predichos.

9. Estudio de valores críticos de parámetros de calidad

9.1 Medición de calidad percibida

Introducción

Existen distintas técnicas y recomendaciones para la medición de la calidad de un video. Se estudiaron estas técnicas de diversas fuentes [5][7][23], las cuales se pueden clasificar como métodos objetivos o subjetivos.
El uso de métodos objetivos tiene la ventaja de que se pueden obtener resultados precisos y repetibles. Sin embargo, todavía no existe alguno de estos métodos que pueda describir en forma completa la percepción subjetiva que tendrá un observador al mirar el video.
Los métodos subjetivos tienen la ventaja de brindar resultados mas confiables pero son mas complicados de llavar a cabo y mas costosos.

Métodos subjetivos

Varios procedimientos de evaluacion de calidad subjetivos son definidos en la recomendación ITU-R BT.500-11 .
Uno de los mas populares es el método DSCQS (Double Stimulus Continous Quality Scale o Escala de calidad continua con doble estimulo en español).
A un evaluador se le presenta un serie de pares de secuencias de video cortas A y B, una a continuación de la otra, y se le pide que asigne un puntaje a cada una marcándolo en una escala continua con 5 intervalos como se muestra en la figura 19.
En cada par de secuencias, una es la original y otra es la misma secuencia degradada por el sistema o proceso a testear (por ejemplo: una secuencia de video afectada por cierto jitter, loss-rate o bien luego de ser codificadas y decodificadas por cierto codec y un cierto factor de compresión).
Terminada la sesión, se normalizan puntajes en una escala del 0 al 100 y se hace un promedio obteniendo un resultado (MOS: Mean Opinion Score) de calidad relativa a la secuencias de referencia.
Otro procedimiento similar definido en la misma recomendación de la ITU para el testeo de los efectos de la transmisión en la calidad de un video es el DSIS (Double Stimulus Impairment Scale o Escala de degradación con doble estímulo) en la cual se compara una secuencia de referencia con otra distorsionada pero sólo se puntúa el grado de degradación de la misma respecto a la de referencia en una escala discreta del 1 al 5 (ver Figura 19 [24]). Se recomienda en el caso de estudiar como afecta la calidad al variar por ejemplo determinado parámetro en la red de transmisión, elegir un conjunto de valores que cubran el rango de grado de degradación en un número pequeño de pasos (razonablemente) iguales. La medida obtenida sera un DMOS (Degradation Mean Opinion Score).
Existen también variantes en las que no hay secuencia de referencia y se puntúa directamente la secuencia distorsionada (ver Figura 19 [24]).
Estos métodos y el resto de los recomendados por la ITU para la medición de calidad subjetiva tienen en común que para obtener resultados precisos, deberán usarse varias secuencias y repetirse el test con varios evaluadores (preferiblemente no expertos). Esto

hace este tipo de procedimientos más costosos y más lentos.

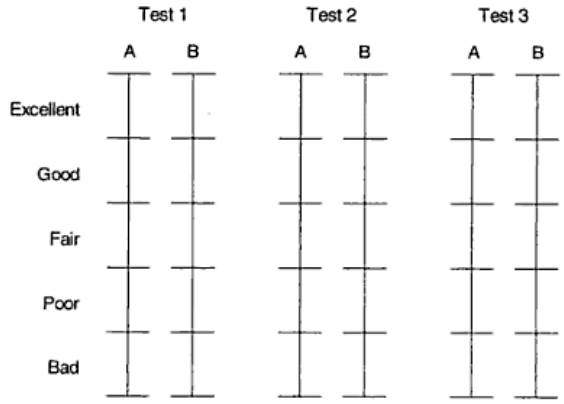


Figura 19- Esquema de puntuacion para DSCQS

ITU-R quality and impairment scales

Five-grade scale			
Quality		**Impairment**	
5	Excellent	5	Imperceptible
4	Good	4	Perceptible, but not annoying
3	Fair	3	Slightly annoying
2	Poor	2	Annoying
1	Bad	1	Very annoying

Figura 20 - Escalas de calidad y degradación percibida

Métodos Objetivos

Los métodos de medición objetivos, en contraste con los métodos subjetivos, son mas fáciles y rápidos de realizar y son ampliamente usados por este motivo para comparar la calidad de video.
El más usado es el PSNR (relación señal a ruido de pico) definido por la Ec. (9) y está basado en el error cuadrático medio (MSE) entre los cuadros del video original y del video distorsionado (comparación pixel a pixel) relativo al rango de valores posibles (L) para un pixel (por ejemplo: L=255 para el caso de 8bits por pixel).

$$PSNR = 10\log\left(\frac{L}{MSE}\right)(DB) \qquad (9)$$

Se observa que para utilizar este método es necesario conocer la secuencia de video de referencia.
La desventaja que tiene este método es que la medida no esta directamente

correlacionada con la calidad percibida por un humano y en algunos casos puede llevar dar a lugar a conclusiones erróneas.
La figura 20 muestra que para valores similares de PSNR pueden haber grandes diferencias en la calidad percibida en la imagen, dependiendo de donde ocurren las degradaciones.
En el par de figuras de "Tiffany" de mas arriba se muestra la imagen original y la imagen distorsionada observando una clara degradación. Sin embargo en el par de figuras del mandril teniendo casi la misma PSNR la degradación es casi imperceptible.

En general en las partes con mayor textura las degradaciones son menos percibidas por la visión humana y se dice que estas quedan enmascaradas.

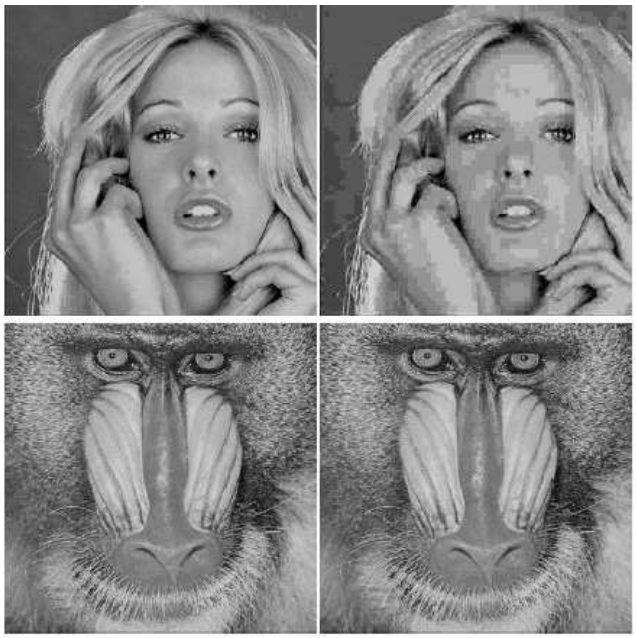


Figura 21 – PSNR como medida de degradacion percibida
Arriba: imagen "Tiffany" original y comprimida PSNR=165
Abajo: imagen "mandril" original y comprimida. PSNR = 163

Como conclusión, estos métodos pueden ser muy útiles como referencia pero no son suficientes para determinar la calidad percibida de una imagen o secuencia de video.

9.2 Método implementado

El objetivo de de esta parte sera determinar criterios que nos permitan realizar un control de admisión en la que se decida si se debe aceptar o no el ingreso de un nuevo cliente de video dependiendo de la calidad solicitada.
En nuestro caso manejaremos dos calidades : calidad media (SD) y calidad alta (HD). El siguiente cuadro muestra sus características :

Calidad	Bitrate	Variabilidad del bitrate	Codec	Frame-rate
Alta (HD)	1.5Mbps	20%	H.264/AVC	24 fps
Media (SD)	570 Kbps	10%	H.264/AVC	24 fps

Se intentará entonces obtener para distintos valores de jitter y porcentaje de pérdidas de paquetes (el retardo tendrá poco efecto en esta parte ya que usaremos video streaming) qué calidades percibidas le corresponden a secuencias de video de calidad media y alta.

Antes de describir las pruebas realizadas debemos aclarar que si bien para hacer un estudio "fino" se debería implementar alguno de los métodos de la recomendación ITU-R BT.500-11 mencionados en la sección 11.1, no lo haremos ya que implicaría un insumo de tiempo importante y está fuera del alcance del proyecto.
Lo que se hará es implementar una versión mucho mas simplificada extrayendo elementos de dichos métodos.
En particular, se utilizarán los mismos conceptos del método DSIS con escala discreta del 1 al 3 y sin secuencia de video de referencia. Es decir, se evaluarán las secuencias de video directamente asignándole una puntación correspondiente al nivel de calidad (bueno, pobre o malo) como se ve en la figura 22.

Quality	
3	Good
2	Poor
1	Bad

Figura 22- Escala de calidad

Se generarán distintos estados en el camino de la red desde el servidor de video hasta el cliente haciendo variar por un lado el jitter y por otro la tasa de pérdidas. En cada instancia se enviará una secuencia de video observando la calidad del video (asignándole un puntuación según nuestra propia percepción) y a su vez midiendo el parámetro de estudio. Para medir los parámetros se simuló el video mediante la técnica explicada en la sección 6.4.2 y de donde se pueden extraer la información para su cálculo. Tanto la simulación (captura) como el video real (enviado como tráfico cruzado) son enviados en forma simultánea, y bajo el supuesto de que todos los paquetes son tratados de igual forma, relacionamos los parámetros con la calidad percibida.

De esta manera se irán mapeando los distintos valores de jitter o porcentaje de pérdidas de paquetes a uno de los tres niveles de calidad definidos.
A diferencia con las recomendaciones de la ITU, no se realizarán las pruebas para varias

secuencias de video ni usando varios evaluadores.
También destacamos que utilizaremos secuencias de video con alto nivel de movimiento ya que este tipo de escenas son más susceptibles a degradaciones.

Pruebas haciendo variar el jitter

Para poder generar un jitter elevado precisaremos tener alta variabilidad en el tamaño de la cola. Además no vamos a querer que haya pérdidas de paquetes para poder ver en forma independiente los rangos de valores de jitter en los cuales la calidad es buena, pobre o mala.
Para lograrlo, generamos un cuello de botella con la herramienta TC [25] [30] (Traffic Controller) con un buffer grande (idealmente infinito) de manera de hacer saturar el enlace y que puedan haber retardos de hasta 5 segundos sin pérdidas.

Pruebas haciendo variar el porcentaje de perdida de paquetes

Es este caso se implementó un escenario con disposición a pérdidas de paquetes en forma gradual conforme al tráfico cursado, manteniendo en valores bajos al retardo y al jitter de forma tal que sólo las pérdidas sean las causas de la degradación del video.

Utilizando nuevamente la herramienta TC para generar el cuello de botella, esta vez se configuró un buffer de menor tamaño y se generó tráfico gradualmente. Se envió el video en diferentes estados de la red calificándolo y obteniendo los parámetros. Se alcanzo un valor crítico de 1% para el cual las pérdidas comienzan a afectar la calidad del video.

9.3 Resultados

Luego de las pruebas realizadas,obtuvimos los siguientes resultados:

Para HD:
-Jitter:
Haciendo variar el jitter entre 2ms y 14ms encontramos los siguientes rangos :

2 – 5,1 calidad buena
5,3 – 6,0 calidad pobre
6,6 – 14ms calidad mala

-Pérdidas:
Haciendo variar las pérdidas entre 0% y 7% encontramos los siguientes rangos :

0 – 0 calidad buena
0.1 – 1 calidad pobre
1 – 7 calidad mala

Para SD:

-Jitter:
Haciendo variar el jitter entre 2ms y 14ms encontramos los siguientes rangos :

2 - 6,8 buena
7,4 - 9,8 – pobre
10 – 14 mala

Pérdidas:
Haciendo variar las pérdidas entre 0% y 7% encontramos los siguientes rangos :

0 – 0 calidad buena
0.1 – 1 calidad pobre
1 – 7 calidad mala

9.4 Conclusiones

Se establecieron rangos de jitter para los cuales la calidad percibida para cada tipo de video era buena, mala o pobre.
Esto nos permite definir criterios para controlar el acceso de los nuevos usuarios al sistema y garantizar la calidad de servicio deseada.
Para los videos de alta definición se consideraran aceptables los parámetros que pertenezcan al rango correspondiente al nivel de calidad bueno y rechazándose aquellas estimaciones que caigan fuera de este rango aceptable.

Para los videos de calidad media o estándar se aceptarán peticiones de video que caigan dentro del nivel de calidad bueno para SD pero por la suposición 4.3.4 también deberán cumplir las condiciones para HD. Esto es, se rechazarán las peticiones para los cuales se estimen jitters superiores a 5.1ms o tasa de pérdidas superiores al 0.1 %.
Se aclara que para el retardo no se tienen restricciones ya que el tráfico es en un sentido, de servidor a cliente. Sin embargo a la hora de reusar el sistema, por ejemplo en video conferencia, el retardo sí sería vital para la asegurar la buena calidad de la aplicación y por lo tanto se entendió que era útil estimarlo.

IV-Pruebas

10. Simulaciones

En las simulaciones se intenta representar el comportamiento del modelo de correlación entre los parámetros de calidad de video y los parámetros de los paquetes de prueba, mediante la herramienta ns2, un simulador de red. En esta etapa se proponen como parámetros que caracterizan el estado de la red al q_n promedio, varianza de q_n y percentil de q_n. No está incluido el porcentaje de cola vacía ya que este fue introducido al sistema en la etapa de pruebas en equipos reales e implementación.

10.1 Procedimiento

Con el objetivo de llevar a la práctica el modelo descripto, y encontrar la relación entre los parámetros de los paquetes de prueba (q_n medio, varianza de q_n y percentil Z% de q_n) y los parámetros de video a predecir (retardo medio, jitter medio y pérdidas de paquetes), se simuló una red con el programa NS2 (network simulator 2), donde se pudo aplicar la técnica de aprendizaje y predicción antes mencionada. Se realizaron una serie de entrenamientos de los cuales algunos fueron utilizados para crear el modelo de correlación y otros para verificarlo prediciendo los valores de calidad de video a través del modelo y comparándolos con los reales.

10.2 Presentación de ns2 como herramienta de simulación de red

NS2 [26] es una herramienta de software muy potente destinada a la simulación de redes. Es muy utilizada tanto en el entorno de la investigación como en el entorno educativo.

Permite realizar simulaciones de múltiples tipos de redes como lo son las cableadas, inalámbricas y satelitales. Utiliza un lenguaje de scripts tcl con el cual nos permite generar modelos. Presenta la facilidad de crear y modificar topologías con características específicas en los enlaces y de crear agentes generadora de tráfico orientado o no a conexión, en tan solo unos minutos. De esta forma NS2 brinda una facilidad de cambios que le permite al usuario a obtener resultados en forma más rápida.

Entre otras cosas, el NS2 permite generar topologías seteando valores de enlaces, tamaño de colas, retardos del medio, nodos, conexión entre nodos y también a diferentes generadores de tráfico como agentes udp y tcp. También se puede variar el tamaño de los paquetes udp como el tiempo de salida de los mismos.

Se encontró en NS2 todos los elementos necesarios para realizar las simulaciones.

10.3 Implementación

Se realizó una maqueta virtual con la siguiente topología:

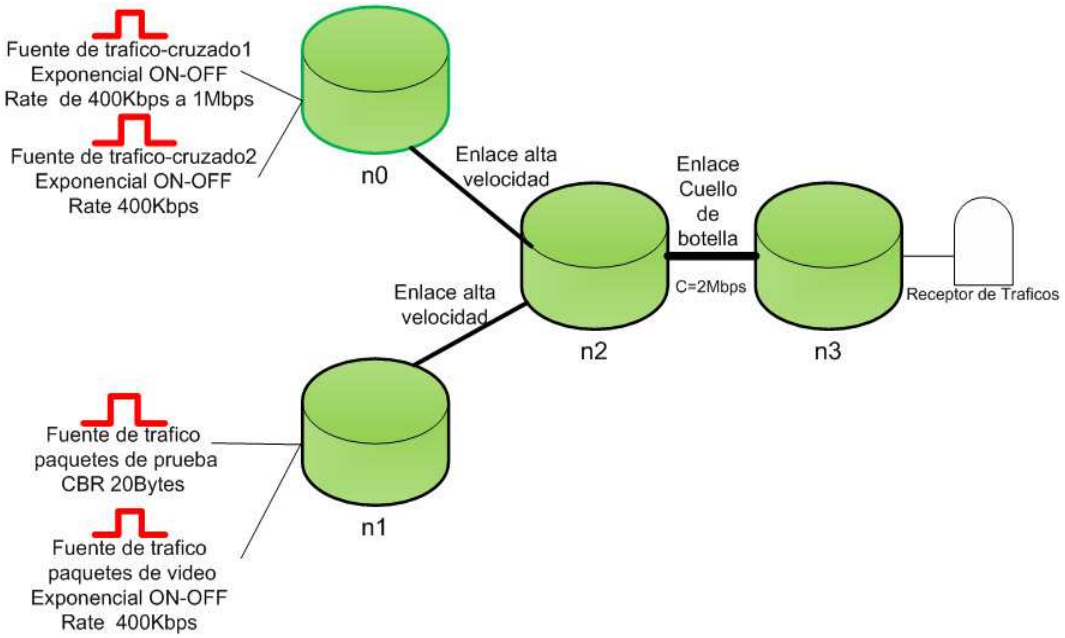


Figura 23

La idea era que por el cuello de botella transitara un cross traffic que simule al tráfico de paquetes de video de otros usuarios de la red. Estos paquetes son los que compiten con los paquetes de video a los cuales se quería predecir su retardo medio.

El cuello de botella es el enlace n2-n3, de capacidad C=2Mbps y con una cola lo suficientemente grande como para que no se descarten paquetes. Los enlaces n0-n2 y n1-n2 son de alta velocidad por lo que introducen un retardo despreciable.

En n0 se encuentran las dos fuentes de ese cross traffic que fueron modeladas como dos exponenciales on-off. La exponencial on-off es un tipo de fuente del ns que tiene la forma de una onda cuadrada donde el tiempo en on (llamado burst time) transmite con una media definida (parámetro rate) y en el tiempo en off (llamado idle time) no se transmite. Los tiempos en on y off son variables aleatorias con distribución exponencial.

Para una fuente (udp02), se definió un rate 800kbps y un burst time e idle time de 100 ms, determinando una media de 400kbps cada 200 ms. El otro cross traffic (udp01) fue el que se varió para realizar ensayos sobre diferentes estados del enlace, su rate iba desde 800kbps a 2Mbps, determinando una media de 400kbps a 1Mbps cada 200 ms.

Por otro lado en el nodo n1 diseñó la fuente del tráfico de video (udp1), modelada como una exponencial on-off con rate de 800kbps, donde la media en 200 ms es de 400kbps. Este modelo se eligió porque el video se transmite en ráfagas separadas por periodos de poca tasa de transferencia ya que solo se envía información de la diferencia con los cuadros anteriores.

En el nodo n1 también se definió la fuente que envía los paquetes de prueba, un tráfico cbr (constant bit rate) con un tamaño de 20bytes por paquetes y un tiempo constante entre salida de 10 ms. Luego en el nodo n3 creamos el agente receptor de tráfico.

Se puede ver que la suma de las medias de todos los tráficos satura el enlace cuello de botella en un 60% a 90% en peor de los casos.

Cada simulación tuvo una duración de 20s y se separa en dos etapas, la primera dura los primeros 10 segundos y se trafica cross trafic + paquetes de prueba, y en la segunda etapa que dura los 10 segundos restantes se trafica cross trafic + paquetes de video.

Luego de construido el archivo .tcl que se le pasará al ns, se creó dos programas en java para encontrar los valores buscados. O sea en base a la salida del ns, llamada out.tr donde se ven todos los paquetes que pasaron por el enlace, se obtiene el q_n medio, varianza de q_n y percentil de q_n con el programa en java Qn.class. Luego con RetardoVideo.class se obtiene el retardo medio, jitter medio y pérdidas nuestro video.

Así con un script se corrió el mismo procedimiento 300 veces para distintos cross traffic, obteniéndose 300 valores distintos de varianza, media y percentil de q_n y retardo medio, jitter medio y pérdidas de los paquetes del video. A cada una de ellas se le llamó instancia de entrenamiento o entrenamiento. Así se obtuvo un conjunto de 300 entrenamientos de los cuales algunos se usaron para realizar el modelo de correlación y otros para verificarlo.

10.4 Correlación de datos

Como se ha mencionado en la implementación del Módulo Estadístico (ME), el libsvm fue la herramienta elegida para encontrar la función que relacionará los parámetros de los paquetes de prueba y el parámetro del video.

Una vez obtenidos los 300 entrenamientos se procedió a normalizarlos, para esto se utiliza la herramienta scale que presenta el lib-svm.

El archivo que se le ingresa debe seguir una estructura ya indicada en la implementación del modulo estadístico en la sección de lib-svm 6.4.8.5 [12]. Se recomienda en la documentación del software, que se normalice la muestra para que los valores estén entre (-1,1).

Una vez construido el archivo con todas las instancias de las simulaciones normalizadas, se dividió en dos partes dejando 200 instancias para el training donde el libsvm creó un modelo y 100 instancias con el que libsvm predijo el parámetro de interés y lo comparó con el verdadero.

Para correr la fase de entrenamiento se deben encontrar los parámetros apropiados con lo cuáles el svm construirá su modelo. En el caso nuestro se utilizaron los siguientes parámetros:

./svm-train -s 4 -t 2 -g 0,32 –b 1 <archivo_conlas200instancias_deTrain_normalizado>

Donde –s es el tipo de svm, en este caso es una regresión nu-SVR (el tipo de SVR fue

hallado viendo cual ajustaba mejor los resultados), luego –t elige el kernel de la regresión: radial basis function: exp(-gamma*|u-v|^2). El –g es el parámetro gamma del kernel elegido y se encontró minimizando el error cuadrático medio. El train produjo un archivo llamado Salida.scale.model que es el modelo a utilizar por el libsvm para predecir.

Luego de ejecutado el train se procedió a realizar la predicción:
./svm-predict -b 1 <archivo_conlas100instancias_dePredict_normalizado> Salida.scale.model output_file

De la predicción se obtuvieron por un lado los valores predichos para un determinado estado de la red y por otro lado están los datos reales por lo que el lib-svm predict los comparó y halló un nivel de correlación y devolvió un error cuadrático medio. Fue minimizando este último valor para que se llegaran a los valores óptimos de los parámetros c y g.

10.5 Resultados

Luego de realizar las simulaciones para distintas intensidades de tráfico cruzado, y utilizar el método de aprendizaje estadístico verificamos que es posible estimar el retardo, jitter y pérdidas de un tráfico de video con una exactitud relativamente alta.

Utilizando como estado de la red a la media, varianza y percentil del estimador de cola del enlace cuello de botella (q_n), se obtuvieron los siguientes resultados:

Mostraremos el resultado de las simulaciones para un percentil del 75%. Además se hicieron simulaciones con percentil de 95%, 90%, 85% y 80% pero se obtuvieron mejores resultados con uno del 75%, aunque no hubo diferencias significativas entre los diferentes percentiles.

Respecto a las predicciones del retardo:

Error cuadrático medio

$$RMSE = \sqrt{\left[\frac{1}{N}\sum_{i=1}^{N}\left(y_i^{predicta} - y_i^{medida}\right)^2\right]} = 0.0079\text{seg}$$

Error cuadrático medio relativo

$$RMSER = \sqrt{\left[\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\left(y_i^{predicta} - y_i^{medida}\right)}{y_i^{medida}}\right)^2\right]} = 9{,}76\%.$$

Error de bias (medida de tendencia del modelo a sub o sobre predecir)

$$BE = \frac{1}{N}\sum_{i=1}^{N}\left[y_i^{predicta} - y_i^{medida}\right] = 4.0203\text{e-}004 \text{ seg}$$

En las siguientes gráficas se puede observar los valores medidos contra los predichos para una fase de entrenamiento de 200 muestras y 100 muestras de validación o predicción.

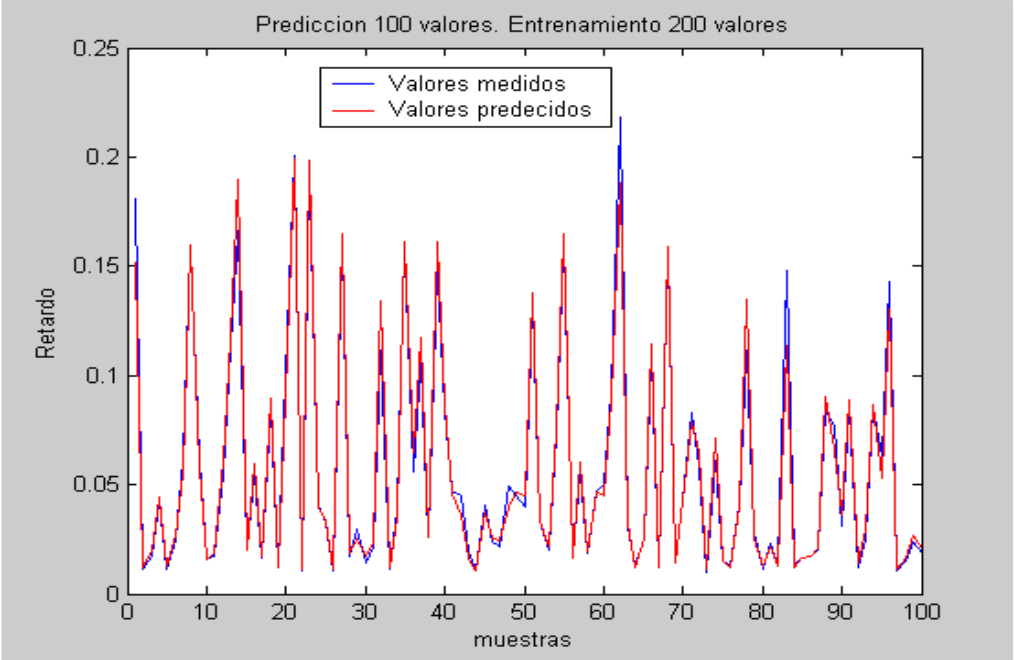


Figura 24

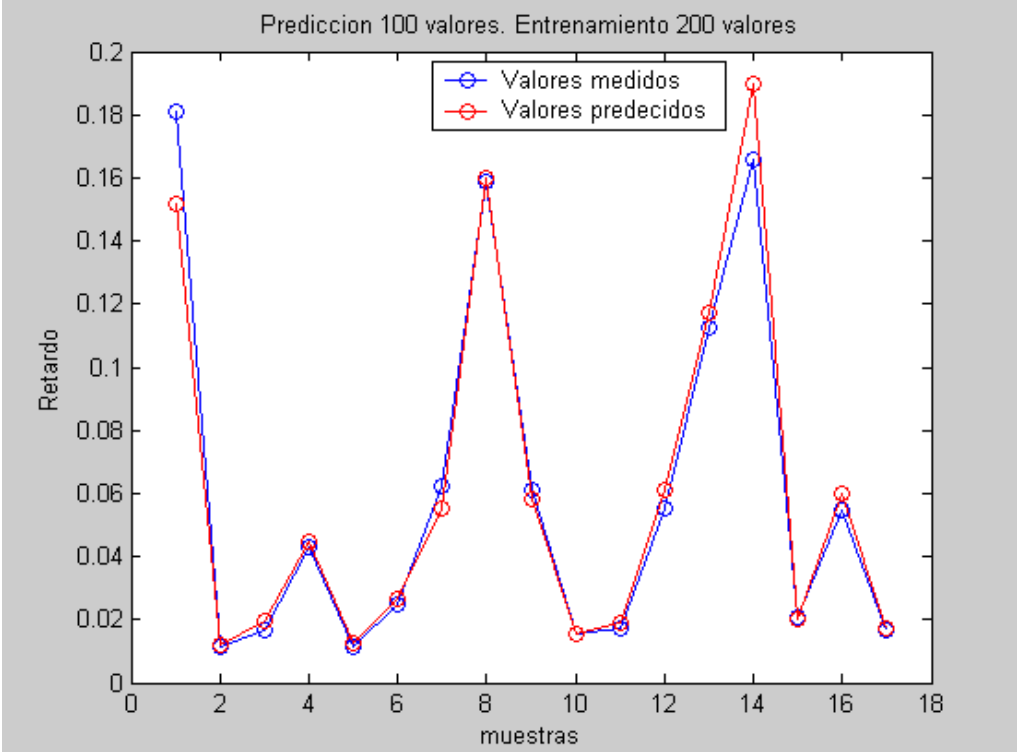


Figura 25

Se puede apreciar que la estimación es lo suficientemente precisa como para no alejarse del valor real en un 9.76% (8ms) en media y en el peor de los casos un 16% como se observa en el primer valor de la grafica de 17 valores.

Los resultados para el jitter son mejores aún, lo cual destacamos ya que junto con las pérdidas de paquetes son los parámetros que afectan en mayor medida la QoS de un video:

RMSE = 1.7086e-004
BE = -2.1617e-006
RMSER = 4,67%

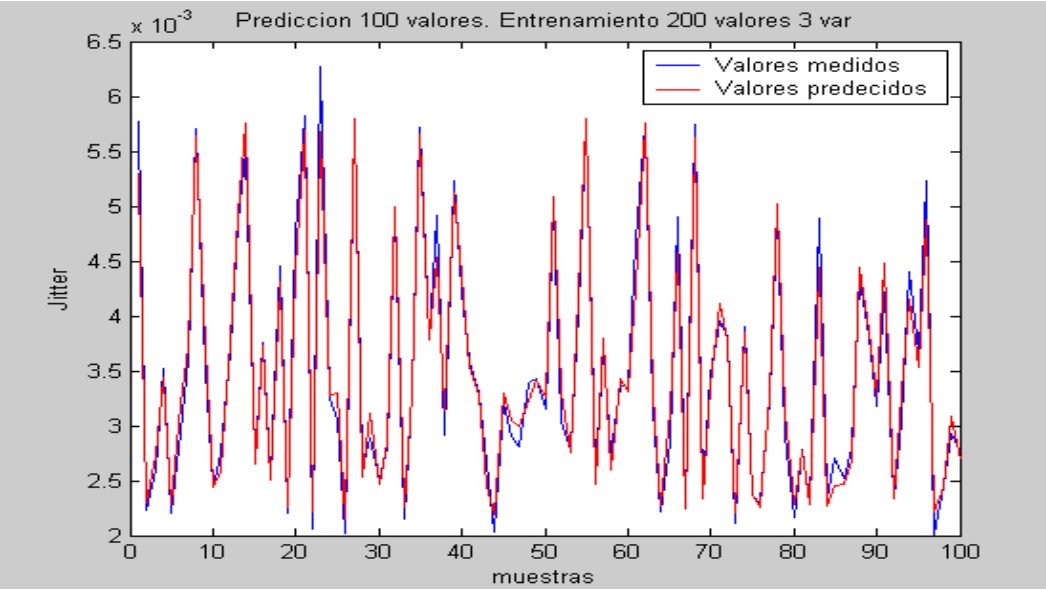


Figura 26

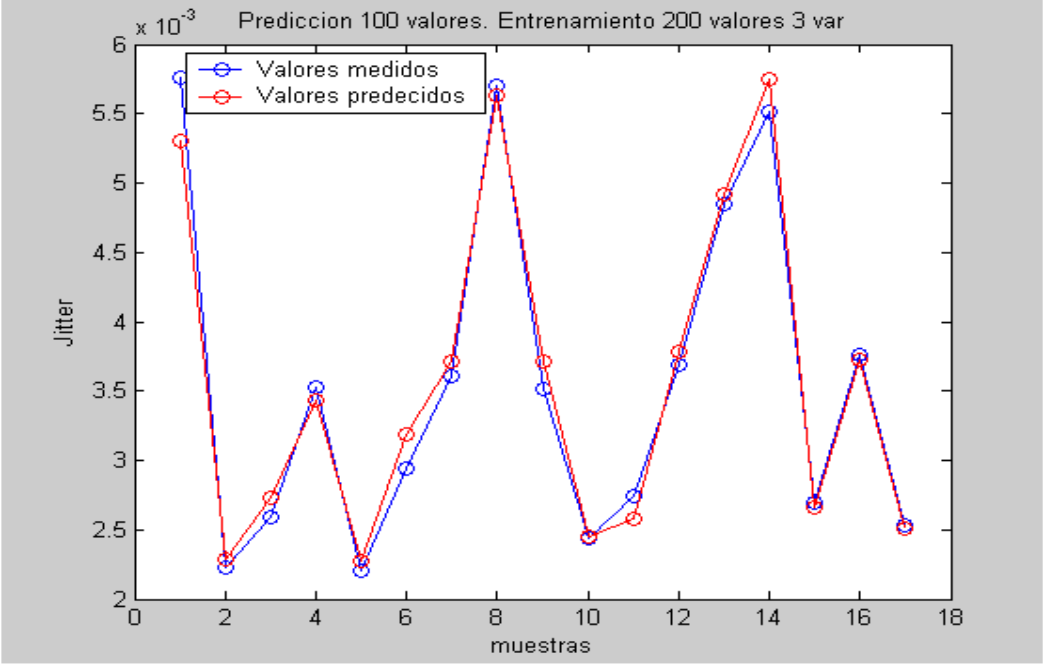


Figura 27

Se observa aquí también una buena aproximación en la predicción.

Para el cálculo de las pérdidas solo se utilizo el promedio y varianza de q_n y además se impuso un valor constante al buffer del cuello de botella, obteniendo:
RMSE = 0.4315
BE = -0.0343
RMSER =Infinito; esto es debido a que hay divisiones entre cero.

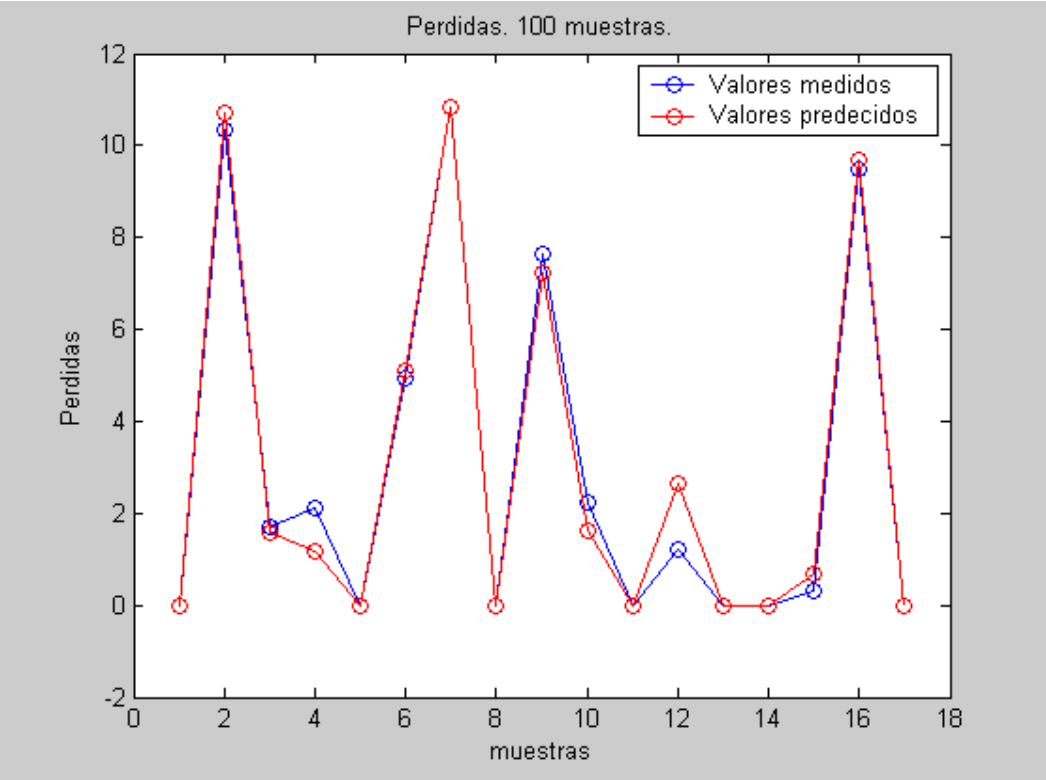


Figura 26

10.6 Conclusiones

En general se obtuvieron buenas predicciones salvo en algunos puntos donde hay errores relativos de alrededor al 16% para el retardo aunque el error en media es del 9.76% lo cual consideramos aceptable. Hay una mejora en la estimación del jitter y las pérdidas lo cual se vio como positivo ya que estos dos últimos parámetros tienen mayor peso en la calidad de un video.

En estas simulaciones se ven buenos resultados para las pérdidas porque se pudieron generar múltiples escenarios diferentes con valores variados de pérdidas, pero en la implementación y en la pruebas reales no será sencillo entrenar clientes con pérdidas por encima del 1% ya que para entonces los algunos clientes obtendrán un servicio con

un porcentaje de pérdidas que pone en riesgo la calidad de los videos en curso.

11. Pruebas realizadas y resultados obtenidos

11.1 Pruebas de estimación de QoS

En esta sección se describirán las pruebas realizadas para la estimación de los parámetros de calidad y los resultados obtenidos. Además se detallan las adversidades enfrentadas en el proceso y las acciones tomadas para lograr los resultados finales.

11.1.1 Procedimiento

Inicialmente se procedió con la implementación del sistema y realización de las pruebas siguiendo el modelo utilizado en las simulaciones. Por lo tanto en la implementación se intento calcar la forma de estimación de los parámetros que caracterizan la red así como los parámetros de calidad.

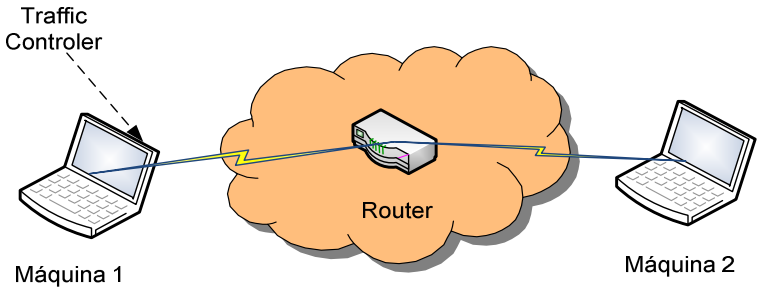
La fuente generadora de tráfico cruzado estuvo siempre compuesta (a excepción de la Etapa 0) por videos reales codificados en H.264 y de diferentes tasa de bit, de forma de crear un escenario lo más "real" posible. Adicionalmente se inyecta tráfico constante de bajo bitrate y en pequeños saltos de forma de generar diferentes escenarios de estado de la red sin perder las características del tráfico.

Se creó una topología tipo en el que el sistema se testeará, a continuación describiremos los elementos que participaron de esta prueba:

-Maquina 1: -servidor vlc: provee video streaming a los clientes vlc. -modulo estadístico: entrenamiento y predicción de clientes.

-Maquina 2: -cliente:

-TP-LINK: Router



Se conectaron los elementos Maquina 1 y Maquina 2 a través de router TP-LINK que oficia de switch. Se creó un cuello de botella estrangulando el enlace a una capacidad de 14.7Mbps con la herramienta Traffic Controler (TC) de linux. Durante las pruebas se

trabajó con un percentil del 90%. En un principio se realizó una captura de 20 segundos del video a entrenar.

Etapa 0 – El inicio.

Con el objetivo de verificar los algoritmos y funciones implementados en el sistema, se crea el escenario ideal en el cual el sistema debiera de predecir satisfactoriamente. Por lo tanto, se creó un cuello de botella de 10Mbps con un buffer de 700 ms, se intentó armar una fuente de tráfico estacionario compuesto por un tráfico constante y un tramo de video de una duración de 45 segundos. Se realizó el entrenamiento variando el tráfico constante tal que la suma del tráfico cruzado varió entre el 85% y el 100% de la capacidad del enlace en media. El entrenamiento se realizó siempre sobre el mismo tramo del video que compone el tráfico cruzado, por lo tanto cada entrenamiento encuentra el mismo tipo de tráfico, a menos de una constante, que afectará a los parámetros de calidad y de la red en forma distinta según qué tan cerca de la capacidad del enlace se encuentre. Así, podríamos suponer que el tráfico que ven los entrenamientos es algo más que estacionario, es más bien cíclico.

En estas condiciones se hicieron 152 entrenamientos de las cuales se usaron 100 para realizar el modelo y 52 para la validación.
El resultado fue muy bueno, alcanzándose una correlación del 92% para el retardo y 90% para el jitter. El error cuadrático medio (MSE) fueron 4.90374e-05 y 3.93538e-08 para el retardo y el jitter respectivamente. En la Figura 29 se muestran las estimaciones para ambos casos.

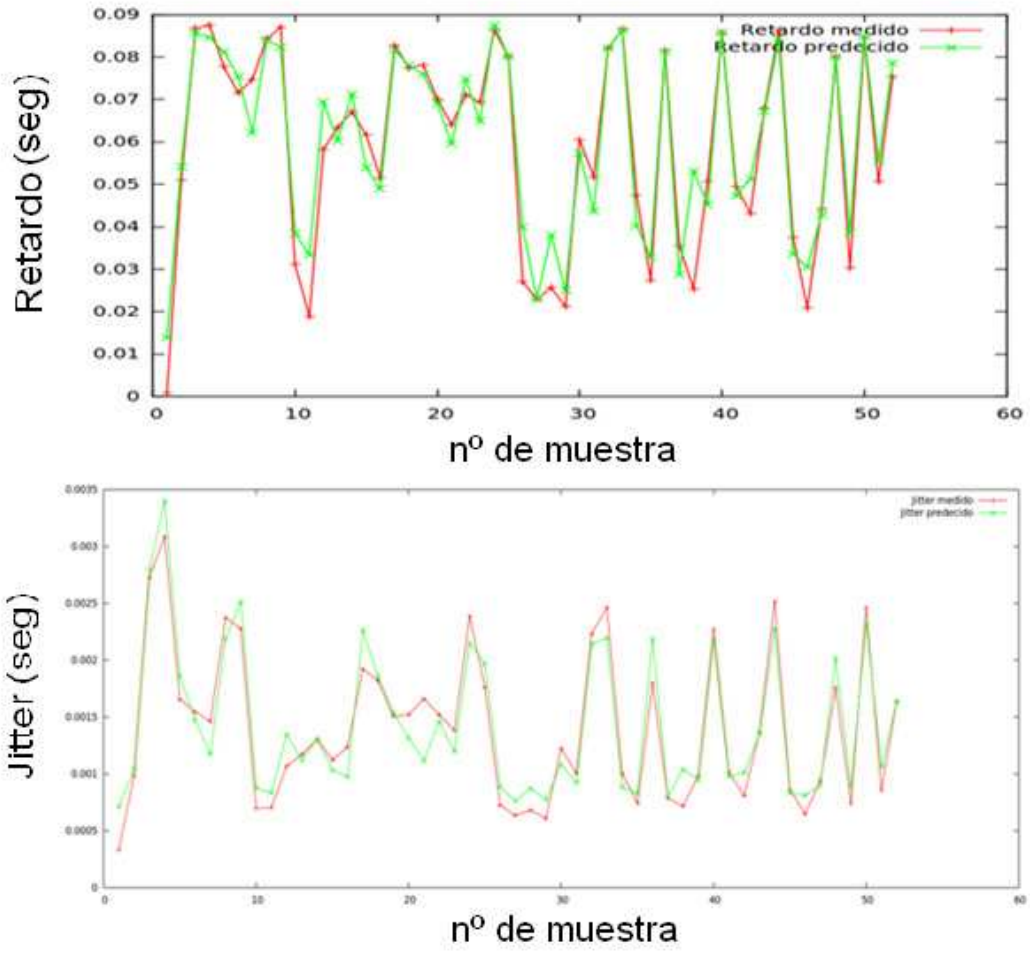


Figura 29- Predicciones con tráfico cruzado compuesto por constante más un video. Arriba predicción Retardo. Abajo predicción Jitter.

Era de esperarse resultados buenos en esta prueba ya que se cumplieron con las hipótesis para el estimador, especialmente con la hipótesis de que el tráfico total y el tráfico cruzado es estacionario a tramos.

En esta etapa se concluyó que las funciones y algoritmos empleados por el sistema, en un principio, responderían a lo esperado, prediciendo en buena medida los valores de Retardo y Jitter.

Etapa 1 – Calculo de q_0**.**

Luego de la etapa 0 se procedió a realizar el entrenamiento al cliente pero esta vez videos reales. Para ello se generó tráfico cruzado que abarcó desde el 70% al 95% de la capacidad del enlace. La fuente de generación de este tráfico fueron videos de 1.5Mbps y 750kbps.

Se obtuvieron 180 entrenamientos de los cuales se utilizaron 100 para crear el modelo y 80 para validarlo.

Cómo ya se ha mencionado cada entrenamiento consta del envío de 15 segundos de paquetes de prueba (pp) seguido de 20 segundos de paquetes de video. Luego con la información de secuencia, tiempo de arribo y tiempo de partida de los paquetes, transportados en el payload, se estiman los parámetros de la red y de calidad del video.

En esta primera etapa no se alcanzaron buenos resultados obteniendo una correlación muy baja. El problema radicó en el cálculo de los qn que caracterizan la red dado por la siguiente ecuación:

$$\hat{q}_n = \sum_{j=1}^{n} \left[t_j^{0,N} - t_{j-1}^{0,N} - t_j^{i,1} - t_{j-1}^{i,1} \right]$$

Donde el tiempo de encolamiento del paquete cero (q_0) es igual a cero si éste encuentra la cola vacía. Pero en los casos en que no se encuentre la cola vacía al inicio del envío de pp, habría que saber el valor de q_0. En caso contrario se introducirá un error igual al valor de q_0 en todos los valores de q_n.

Para solucionar este problema se plantea detectar la cola vacía y usar dicho valor de q_n como referencia ya que debería de ser cero. Luego llevarlo a cero sumándole el opuesto de su propio valor. Y por último se le suma a todos los valores de q_n dicho valor de referencia llevando a todos los q_n incluso q_0 a su valor correcto.

Los detalles de cómo detectar la cola vacía se encuentra en la sección 6.4.7.2

Etapa 2 – Tiempo de envío de captura de video.

En esta etapa se corrigió el problema de la etapa anterior y se realizan nuevamente las pruebas conservando la fuente de tráfico cruzado.

Los resultados esta vez tampoco fueron buenos, se obtuvo baja correlación entre parámetros que caracterizan la red y los parámetros de calidad. Esta vez los malos resultados se relacionaron a dos situaciones independientes, una la cantidad de entrenamientos y la otra relacionada al tiempo de captura del video a entrenar.

Para el análisis de la segunda causa, recordemos que una de las hipótesis del algoritmo de estimación por aprendizaje estadístico es la estacionariedad a tramos del tráfico para que los paquetes de prueba y video vean la misma estadística en el tráfico cruzado en la

etapa de entrenamiento. Por otro lado, se debe encontrar el tiempo necesario de envío de paquetes de prueba y video, de forma tal que en dicho tiempo se logre capturar el estado de la red y medir los parámetros del video.

Consecuentemente se analizó el tráfico cruzado generado. Se midieron la media y varianza para 20 segundos, 30 segundos y 1 minuto en diferentes velocidades sobretodo entorno a la saturación. Se muestran algunos valores del estudio a modo ejemplo:

Para 30 segundos:
Bitrate medio = 15434500.019424
Bitrate medio = 15433209.628776
Bitrate medio = 15182610.398895
Bitrate medio = 15340348.311574
Bitrate medio = 15254386.550400
Bitrate medio = 15433404.834780

Para 20 segundos:
Bitrate medio = 15022266.597586
Bitrate medio = 14945131.306500
Bitrate medio = 14593304.238310
Bitrate medio = 14667908.225974
Bitrate medio = 14776014.908257
Bitrate medio = 14693941.947853
Bitrate medio = 14544078.466378

Estos datos son para un tráfico entorno a los 14.5Mbps provocado por 14 videos de diferentes bitrate, pero se analizaron la media y varianza del tráfico cruzado para diferentes valores de tráfico.

Se puede observar que para 30 segundos la media del tráfico se comporta de forma más estable que para 20 segundos.

Estos resultados se comprobaron también para diferentes tráficos que van desde el 70% de la capacidad hasta la saturación.

Se decidió entonces estirar el tiempo de envío de paquetes a 30 segundos ya que en ese intervalo, dado un estado del tráfico, se halló menos variabilidad en las medidas de promedio y varianza.
.

Etapa-3 – Tiempo entre partidas de pp

En esta etapa se modificaron los algoritmos para capturar mejor las variaciones en el tráfico cruzado además de aumentar la cantidad de entrenamientos.

Para esta etapa se creó un cuello de botella de mayor capacidad estrangulándose el enlace a 15.9Mbps, con la intención de que la multiplexación de una mayor cantidad de videos resultara en una mejor estacionariedad del tráfico. Se generó entonces una fuente de tráfico conformado por videos de diferentes bitrate adaptándose al nuevo cuello de botella.

Se hicieron 350 entrenamientos de los cuales se usaron 225 para el modelo y 125 para la validación alcanzándose mejoras en los resultados. En la siguiente figura se muestran las estimaciones para sesenta valores:

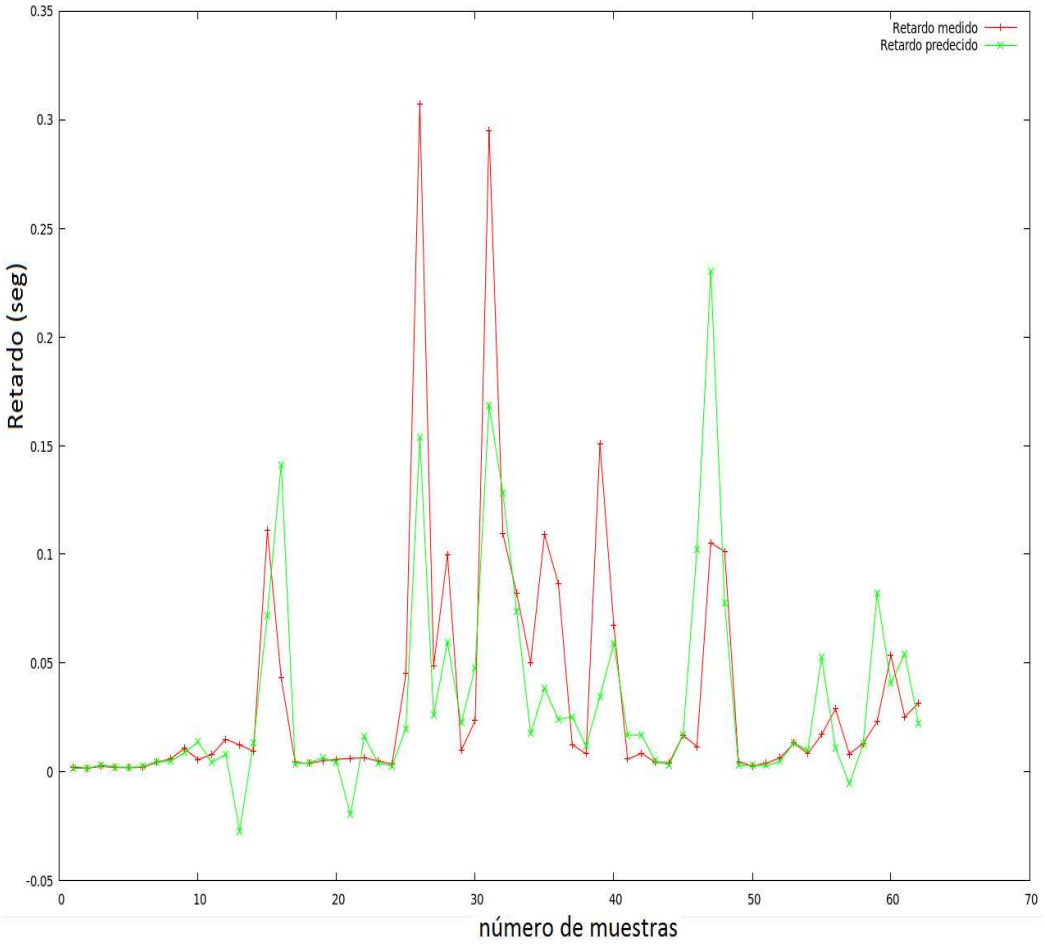


Figura 30 - Retardo para 30 segundos de captura. Etapa 3

Se puede observar que hay cierto seguimiento de los valores predichos a los valores medidos aunque no es el óptimo.

Para mejorar los resultados obtenidos, esta vez se hizo un análisis más exhaustivo con tres partes diferenciadas, los cuales se implementaran en la siguiente etapa.

Por un lado se estudió el tráfico cruzado observándose que el tiempo entre partidas de paquetes, de todo el tráfico, variaba entorno a los 0.8ms. Con el afán de tener una mejor descripción del estado de la red, se propuso cambiar el tiempo de partida entre pp a 3ms (se probaron valores más bajos pero no fueron bien reproducidos por nuestros equipos).

Por otro lado se analizó el comportamiento de la cola del enlace cuando entrenaba cerca de la capacidad máxima del enlace y especialmente cuando se envía la captura del video. Se generaron las condiciones de saturación y se graficó el tiempo de encolamiento de los pp en dos momentos: mientras que por el enlace solo cursa tráfico cruzado y por otro lado cuando cursa tráfico cruzado más la captura del video. En la Figura 31 muestra ambas graficas.

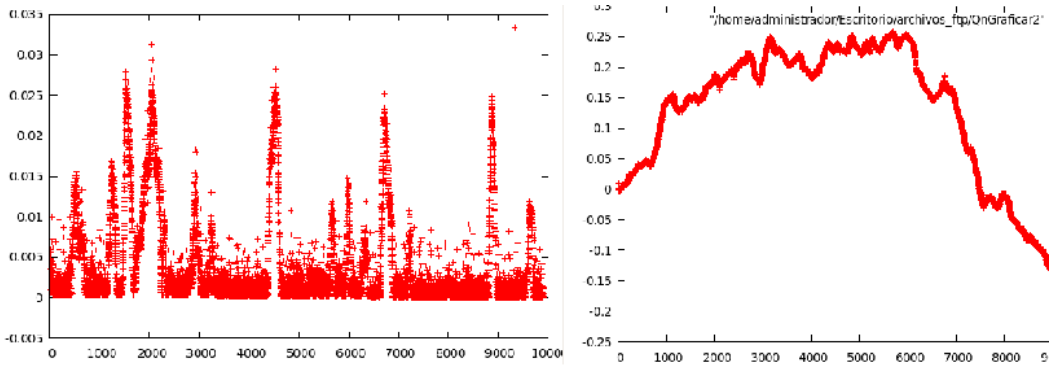


Figura 31 - Izquierda: Gráfica del tiempo de encolamiento de pp con solo tráfico cruzado. Derecha: Tiempo de encolamiento trafico cruzado mas captura del video. Este tiempo no fue corregido a través de q_0 para llevarlo a su valor "real", es por esto que hay valores negativos (ver 6.4.8.3).

De la segunda gráfica se observa que no se está tomando el suficiente tiempo para capturar al menos un par de ciclos de cola vacía, lo que daría indicios de estacionariedad. Para solucionar esto, se plantea o bien no se toman en cuenta estos entrenamientos, o bien se estira el tiempo de captura del video aún más.

Por último, de la gráfica se observa poco margen entre que comienzan a encolarse los paquetes de prueba (izquierda) y que se pueda capturar en 30 segundos cierta estacionariedad del tráfico cuando cursa la captura (derecha).

Es conveniente que los entrenamientos se hagan en el rango mencionado ya que fuera del mismo se encuentran una de las siguientes dos situaciones: 1- Que los pp encuentren la cola vacía en todo el tiempo de entrenamiento, lo cual no aporta mucha información. 2- Que no se encuentre una representación de la estacionariedad en el tráfico durante el envío de la captura de video. Un tráfico más variable, en términos reales, ayudaría a ampliar el margen donde es deseable el entrenamiento.

Etapa 4 – Resultados Finales

En esta etapa se implementan las soluciones a las mejoras presentadas en la etapa anterior. Se tuvo especial atención en generar un tráfico cruzado un poco más variable, , y en agregar tráfico cruzado al punto tal que sumado a la captura del video se observaran ciclos de la cola vacía. Se mantuvo el cuello de botella a 15.9Mbps.

Se hicieron 251 entrenamientos utilizándose 191 para crear el modelo y 60 para validarlo.

Los resultados tuvieron mejorías alcanzando valores aceptables para el jitter y el retardo y no tantos para las pérdidas como se mostrará en el próximo punto.

11.1.2 Resultados

Luego de la descripción de la evolución del procedimiento de las pruebas, se presentan los resultados finales obtenidos.

Para el Retardo se alcanzó una correlación del 80% y un error cuadrático medio (MSE) igual a 0.00756324
En la Figura 32 se observa una mejora respecto a las pruebas anteriores. En el caso del retardo se usaron preferentemente valores altos (eliminando los bajos) para generar el modelo y valores altos y bajos para la validación, buscando así una mejor predicción para los valores altos que son los de mayor interés. Efectivamente se consiguió una mejoría.

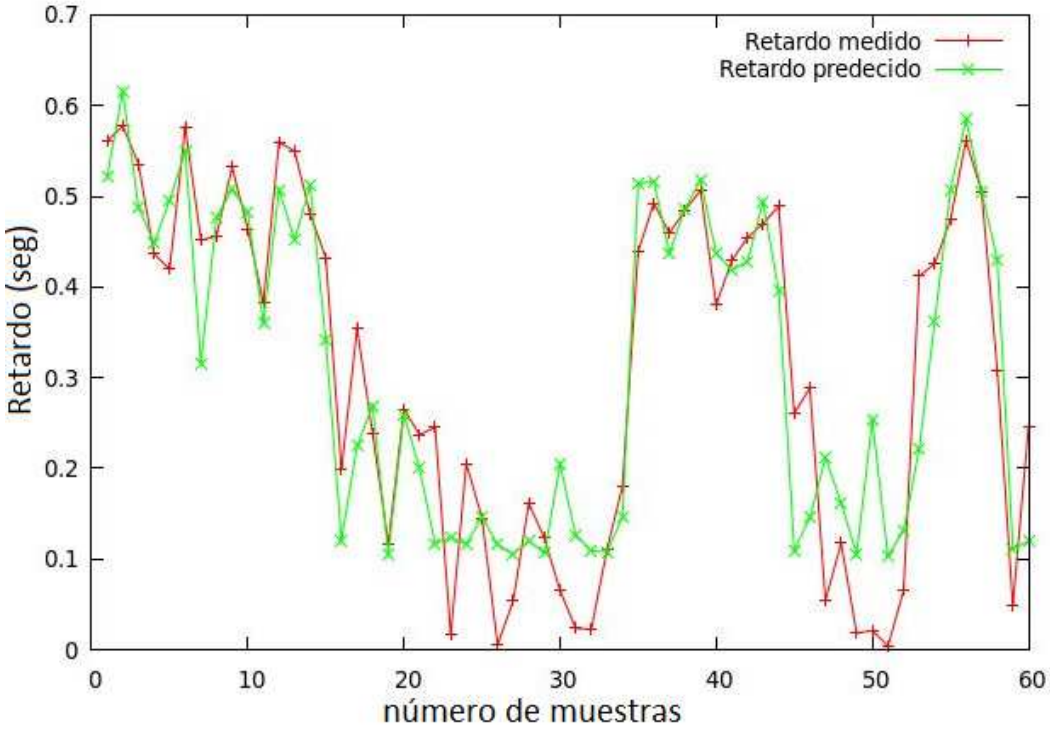


Figura 32- Predicción Retardo. Correlación 80%. MSE = 0.00756324.

En el caso del Jitter se alcanza una correlación de 84% con un MSE igual a 2.47148e-07

En la Figura 33 se puede observar un seguimiento pronunciado en el jitter con algunos errores de hasta el 22% como el caso del séptimo entrenamiento el cual predice 3,5ms cuando debería de haber sido 4,5ms. En los casos en que sobre-predice en la zona cercana a la saturación, no preocupa ya que representa el caso en que un cliente solicita un video y se le niega el acceso cuando en realidad estaba en condiciones de visualizarlo. El problema es cuando sub-predice, en ese caso se podría cometer el error de permitir el acceso a un cliente cuando no se debiera. Como consecuencia el nuevo video circulará por la red sobrecargándola y degradando la calidad de todos los videos que comparten el camino.

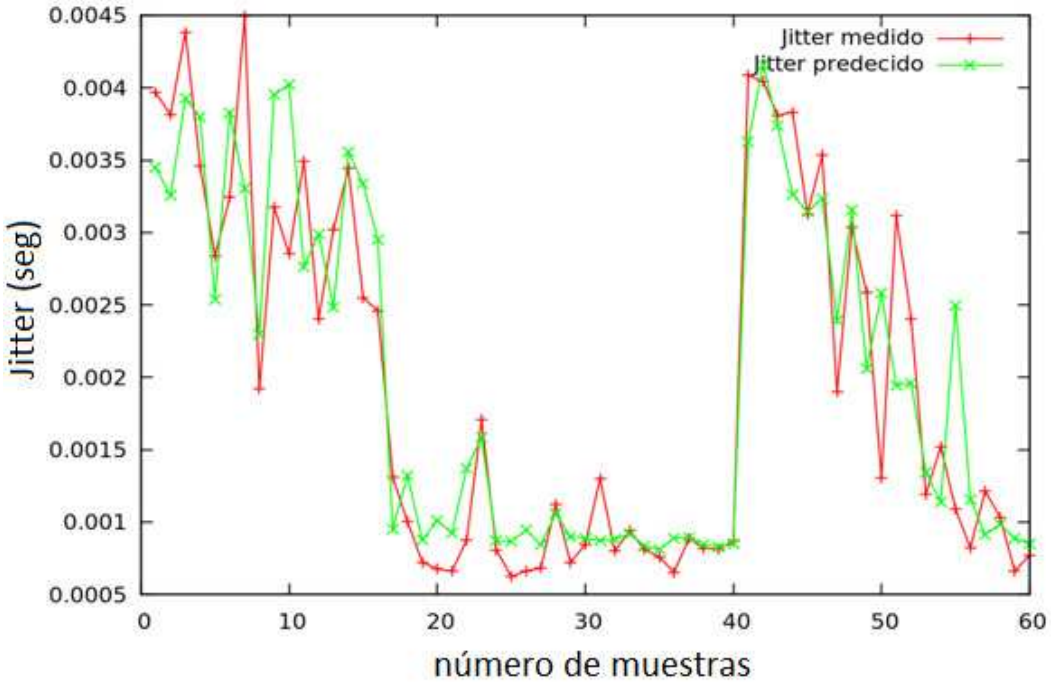


Figura 33- Predicción Jitter. Correlación 84%. MSE = 2.47148e-07.

Por último, en la Tasa de Pérdidas se alcanza a una correlación del 66% con un MSE igual a 0.0146993

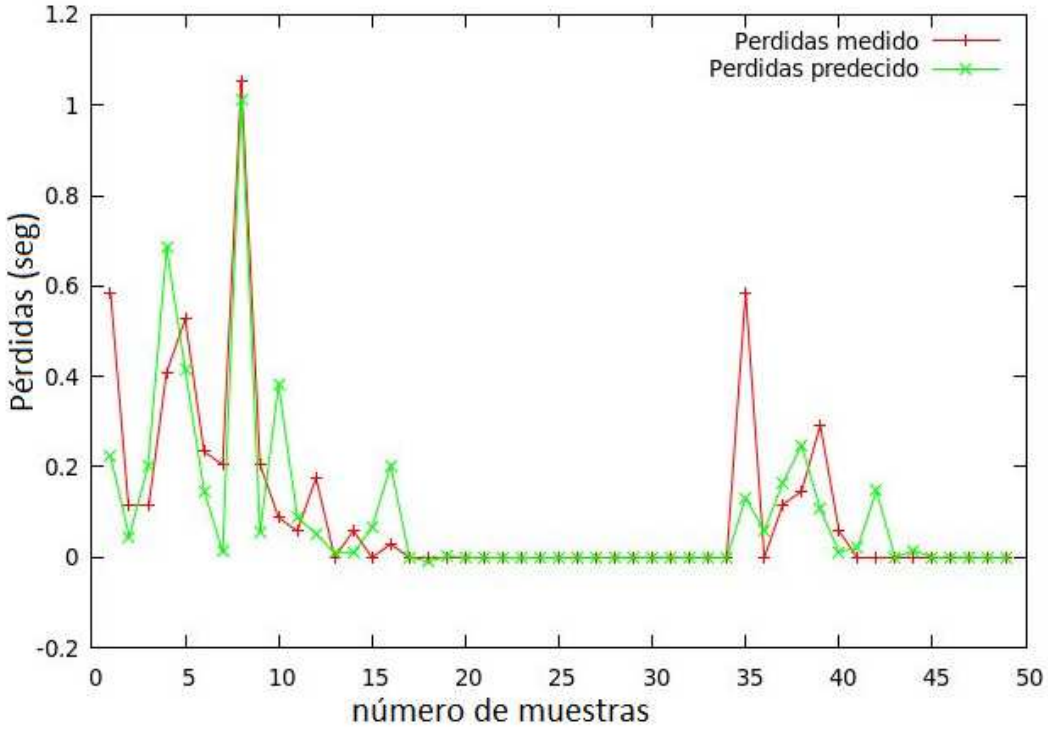


Figura 32- Predicción Perdidas. Correlación 66%. MSE = 0.0146993.

En las pérdidas no se obtuvieron buenos resultados en esta última prueba y se atribuyen a la poca información generada para la creación y validación del modelo.
Se refiere a poca información, ya que se encontraron valores iguales a cero en la tasa de pérdidas de la mayoría de los entrenamientos.
Para comprobar que se podían predecir correctamente las pérdidas, se creó un nuevo ensayo con diferentes características a los anteriores. Con la intención de obtener mayores instancias con diferentes tasas de pérdidas, se redujo el tamaño del buffer del TC. Como se muestra en la Figura 35, con mayor información mejora la predicción de las perdidas. En este caso la correlación subió al 90%.

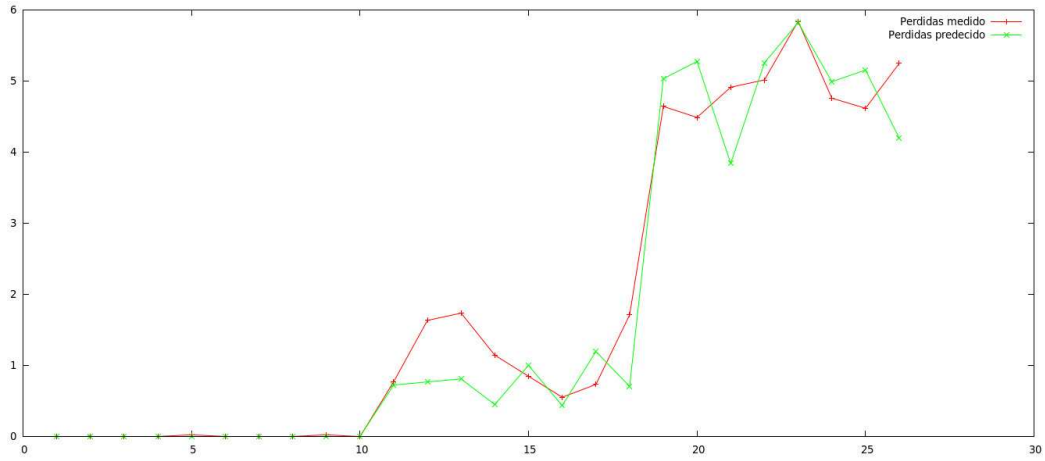


Figura 35- Predicción Tasa de pérdidas.

11.1.3 Conclusiones

La estimación o predicción de los parámetros de calidad de un video dependen fuertemente de las condiciones de estacionariedad que presente el tráfico cruzado. Precisamente estas condiciones deben cumplirse para el rango de valores de tráfico donde se efectúan los entrenamientos. Pero no solo lo anterior basta, sino que se debe verificar que el tiempo de entrenamiento sea suficiente como para capturar dicha estacionariedad, sobre todo cuando transcurre la captura. Este tiempo es mayor a medida que el tráfico se acerca a la saturación del enlace.

Entonces existe un compromiso entre el tiempo de cada entrenamiento y el nivel de tráfico en que trascurren los entrenamientos. Por un lado mientras mayor sea el tiempo de cada entrenamiento, mayor será el tiempo de sobrecarga de la red. Pero en contrapartida se logra entrenar valores más cercanos a la saturación, y por lo tanto se tendrán predicciones más certeras para valores de retardo, jitter y pérdidas cercanos a los críticos. Teniendo en cuenta dichas consideraciones se lograron obtener los resultados finales.

11.2 Testeo final del sistema integrado

Al comienzo del proyecto se planteo (ver sección 1.2) como objetivo la siguiente validación del sistema:
"Montar una red con un servidor de video, 3 o más clientes incluidos en la red de distribución y el sistema diseñado, se debiera verificar lo siguiente:

- Que el sistema permita el acceso al cliente cuando cumple los requerimientos de calidad extremo a extremo (del servidor de video al cliente).
- Que el sistema no permita el acceso al cliente cuando no cumple los requerimientos.
- Que se cumplan los parámetros de calidad una vez que se permita el acceso a un cliente."

Con el objetivo planteado se procedió a las pruebas. Primero se creó el escenario generando un cuello de botella de 14.7Mbps de igual forma que se describe en la sección de pruebas de estimación de QoS.
De acuerdo a los resultados obtenidos en las pruebas de la sección anterior, donde se hacía notar la dificultad de obtener un jitter alto, se decidió generar un escenario donde se pueda discriminar en base a las pérdidas.

Se corrió el programa principal del M.E con un cliente ya entrenado previamente. Lo primero que éste realiza, cuando se pone en funcionamiento, es buscar en todos los clientes las tareas a realizar con ellos, como por ejemplo si debe predecirlos o entrenarlos, dependiendo de si ya tiene creado un modelo válido o no. Como el cliente ya está entrenado el programa principal hace una predicción y queda esperando el tiempo requerido para hacer una nueva acción.

En la máquina del proxy se levanta un servidor web con links a los videos disponibles.
Se generó un tráfico alto entorno a los 12.5Mbps luego el cliente hizo una petición a un video de "alta definición" (de 1.3Mbps de media) mediante la página web. Se verificó que el cliente visualizó el video con buena calidad (testeado por los integrantes del grupo con el mismo criterio utilizado en el capítulo 9). Luego se agregó tráfico adicional (el video de la petición aceptada) alcanzando valores cercanos a la capacidad del enlace. El cliente realiza una nueva petición a un video y el proxy no le permite el acceso, desplegando en pantalla la sugerencia de visualizar un video de menor calidad. Para constatar que ese video presentaría una mala calidad, se mantuvo el tráfico, se envió el video y se verificó la mala visualización del mismo. Por último el cliente accedió a un video de media definición con la calidad debida.

V- TRABAJO A FUTURO Y CONCLUSIONES GENERALES

12. Conclusiones

En esta sección se mencionan algunas conclusiones generales sobre el sistema completo ya que cada capítulo tiene conclusiones particulares.

En líneas generales se implementó un sistema de control de acceso y predicción de QoS para la aplicación VoD dentro de una red controlada, bajo ciertas hipótesis, garantizando mantener la calidad de los videos en curso.
Compuesto por dos componentes principales, como lo son el M.E. (Modulo estadístico) y el Modulo Proxy, el método implementado como criterio de control fue la predicción de los parámetros que describen la calidad que tendrá un video si es cursado por la red.

Debemos destacar que la comunicación entre los módulos es estándar (SNMP), lo que permite la posibilidad de cambiar alguno de los módulos sin perder la funcionalidad general y sin la necesidad de nuevas implementaciones en el resto del sistema. Logramos así cumplir con el objetivo de reusabilidad planteado al comienzo.

En la implementación del Proxy se utilizó el conocido Squid adaptándolo a las necesidades del sistema. Se alcanzó un módulo estable logrando óptimos resultados en el control de acceso de los clientes a la visualización de los videos. Su función es permitir o denegar el acceso en función de los valores retornados por el ME sobre la predicción del video.

En cuanto al M.E. se alcanzaron resultados razonables en las predicciones de los valores de calidad, siempre y cuando se cumplió con las condiciones de estacionariedad en el tráfico cruzado.

Como se vio en las pruebas, los valores críticos de los parámetros de calidad se presentan en estados de tráfico entorno a la saturación del enlace. Por otro lado, entrenar sobre tráficos cercanos a la saturación implica, o bien no encontrar estacionariedad en el tiempo de entrenamiento, o bien aumentar la ventana de medición del estado de la red para encontrarla. Una forma de verificar la estacionariedad sería corroborar que para el mismo estado de la red, entrenamientos sucesivos den resultados similares.

Teniendo en cuenta los resultados obtenidos en el capitulo anterior, se observó una adecuada estimación de los parámetros de QoS. Sin embargo, en el Capítulo 9 se definieron rangos de los distintos parámetros de video en los cuales se alcanzaban diferentes niveles de calidad. Por esto último, se entiende que no es fundamental que el sistema prediga con exactitud los parámetros medidos, pero sí que clasifique correctamente en qué rango se encuentran dichos parámetros.

13. Trabajo a futuro

13.1 Mejoras al programa

Se dice que “un proyecto no se termina sino que se abandona” porque siempre hay cosas para mejorarle.
Haremos un breve repaso sobre algunas de las cosas que no dieron tiempo de incluir:

Interfaz grafica

El programa genera reportes donde se puede leer el estado de los clientes: horarios y valores de últimas predicciones y entrenamientos, si están viendo videos y cuantas solicitudes le han sido rechazadas. También se encuentra disponible un archivo con logs donde dice cuales entrenamientos y predicciones se realizaron y cuáles no se pudieron concretar por falla de conexión con el cliente u otros errores.
Igual entendemos que una interfaz más amigable con el operador sería más conveniente. Evaluamos que sería relativamente sencillo realizarlo, porque la información ya se encuentra disponible y solo habría que decidir como diseñar la interfaz en sí misma. Podría ser una página web donde los clientes figuran como símbolos que cambian de color según el estado, desplegándose con un click la información requerida (al estilo del típico programa de monitoreo de red “What’sUp”).

Resolución de la simultaneidad de solicitudes de clientes

El programa principal del M.E es el que debe manejar la simultaneidad de solicitudes. Tal y como quedó implementado, el programa dará por validas las ultimas predicciones salvo que el sistema se encuentre prediciendo a todos por la aceptación de una solicitud reciente. Esto es debido a que cambia el estado de la red cada vez que hay un nuevo video cursando por ella y debo conocer el nuevo estado de los caminos hacia cada cliente.
Entonces si llega una nueva solicitud de un cliente al momento que se encuentra prediciendo a todos, el M.E predice a este nuevo cliente (lo pone primero en la fila a predecir). Con lo que el nuevo cliente deberá esperar que se realice la predicción para saber si puede acceder al video. Si bien la predicción se realiza en unos 30 segundos y en principio no parecería una espera tan grave, el problema se agrava cuando son varios clientes lo que solicitan un video al mismo tiempo. Por ejemplo si son 10 clientes los que coinciden en su petición con una diferencia de pocos segundos, la espera del último sería aproximadamente de cinco minutos. Si bien la probabilidad de que eso ocurra depende de la dimensión de la red de distribución, esto se debería solucionar si se quiere lograr escalabilidad.
La forma de resolverlo sería que una vez que se comenzó a distribuir un nuevo video, el M.E pueda predecir a varios clientes al mismo tiempo. Se podría crear un proceso hijo por cada cliente que trabaje en paralelo con el resto, prediciendo de a bloques de pocos clientes para no cargar demasiado la red.

13.2 Escalabilidad

Sin duda, la tendencia actual en lo que refiere a VoD y a streaming en general, es llevar el servidor de video más cerca del cliente. De esta manera no se introduce tráfico innecesario a la red cuando se le podría dar el servicio desde el nodo de acceso más cercano al cliente. Por esta razón es que se han desarrollados sistemas de video on demand con arquitecturas más complejas de servidores en cascada. Un ejemplo de esto es el llamado VoD masivo o LVoD(large-scale VoD) [27]
El sistema que se presento, fue pensado para que el servidor de video no esté en el mismo equipo que el Proxy ya que es posible que haya más de un servidor de video. Sin embargo preferimos que exista un solo proxy que reciba las consultas de los distintos clientes.
Claramente si hay más de un servidor de video, cada uno de ellos deberá contar con un M.E al menos "colgando' desde el mismo nodo de la red. Así las mediciones de extremo a extremo comparten el camino que haría el video solicitado por el cliente.
En el caso de tener varios servidores de video (por lo tanto las misma cantidad de M.E) se podría implementar que cuando un cliente haga una solicitud, el proxy evalúe las respuestas de todos los M.E y decida cuál es el servidor que le dará el video en las mejores condiciones. Esto puede permitir también realizar balanceo de carga entre servidores de video, siempre y cuando existan dos servidores o más que cumplan con los requerimientos de QoS solicitados.

13.3 Reusabilidad del sistema

13.3.1 Descarga TCP

Una posible reutilización del sistema es la estimación de tiempos de bajada de ficheros mediante TCP y el control de admisión para poder acceder a los mismos. Para ello habría que modificar el M.E de forma de que se estime el throughput de la conexión extremo a extremo entre servidor y cliente.
Esto no requeriría grandes cambios gracias a que ambos módulos principales se diseñaron con un bajo acoplamiento.
Si observamos que el throughput de TCP depende básicamente del RTT (Round-Trip Time) y de la tasa de pérdida de paquetes y que ambos están correlacionados con el estimador de la cola que utilizamos, se podría utilizar el mismo mecanismo para hallar una correspondencia entre el estado de la red y el throuput. Para esto la aplicación en el cliente debería devolver el tráfico que recibe para medir el estado del camino en ambos sentidos.

13.3.2 Video conferencia

En el caso de video conferencia se requeriría que cada cliente que participa de la conferencia tenga en su pc una aplicación cliente un poco más sofisticada. Ya que debería enviar la secuencia de paquetes de prueba y luego los de video para la fase de entrenamiento y luego enviar solo los paquetes de prueba para la predicción. El procesamiento de la información lo podría seguir haciendo el M.E pero el debería impartir órdenes a la aplicación alojada en la pc del cliente para realizar lo antes mencionado.
También se debería estudiar los requerimientos de QoS para codecs de video conferencia (H.263) y en todo caso cambiar los criterios de decisión para dar acceso a la video llamada.

13.3.3 Voz sobre ip

Los cambios serian muy similares a los vistos en la video conferencia. Los clientes se enviarían paquetes entre sí para entrenarse y predecirse, y el M.E controlaría cuando se hacen esas tareas. Luego el proxy tomaría la decisión en el momento de la solicitud de llamada en base a los requerimientos propios de voz sobre ip.

BIBLIOGRAFIA

[1] *ITU-T. (2007b). Vocabulary for performance and quality of service: Appendix I – Definition of Quality of Experience (QoE) (Vol. P.10/G.100): ITU-T*

[2] *Tesis de Maestría Regresión no paramétrica para datos funcionales no estacionarios*. Laura Aspirot [Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay]

[3] *End-to-end quality of service seen by applications: a statistical learning approach*; Pablo Belzarena, Laura Aspirot [Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay].

[4] "*Proyecto Metronet: software para medición de calidad de servicio en voz y video*" Pablo Belzarena, Víctor González Barbone, Federico Larroca, Pedro Casas

[5] *Estudio de la Medida de la Calidad Perceptual de Video*, Jose Joskowicz. Master thesis from Universidad de Vigo, España - 2008

[6] *One-way transmission time,* ITU-T Recommendation G.114

[7] *End-2-End Evaluation of IP Multimedia Services, a User Perceived Quality of Service Approach*, Pedro Casas, Pablo Belzarena and Sandrine Vaton.

[8] *Support Vector and Kernel Machine*; Nello Cristianini [BIOwulfTechnologies]

[9] *A Tutorial on Support Vector Machine for Pattern Recognition*; Christopher J.C. Burges [Bell Laboratories, Lucent Technologies]

[10] *The nature of statistical learning theory*. V.N. Vapnik [Springer NY, 1995]

[11] *Modelos, Mediciones y Tarificación Para Redes con Calidad de Servicio*. Pablo Belzarena. Doctor en Ingeniería Eléctrica de la Universidad de la República, 2009.

[12] *A Practical Guide to svm clasification*. url: www.csie.ntu.edu.tw/~cjlin/libsvm/

[13] *LIBSVM: a Library for Support Vector Machines*; Chih-Chung Chang, Chih-Jen Lin

[14] Sitio web de Tcpdump, http://www.tcpdump.org/

[15] Sitio web de NTP, http://www.ntp.org/

[16] *Introducción a la programación con C*. Andrés Marzal, Isabel García.[Universitat Jaume I]

[17] *Beginning C: From Novice to Professional, Fourth Edition*. Ivor Horton

[18] http://www.fismat.umich.mx/mn1/manual/node21.html

[19] *Inter-process Communication: Shared memory*. Cardiff School of Computer Science & Informatics, http://www.cs.cf.ac.uk/Dave/C/node27.html

[20] *"Squid: The Definitive Guide"*. Duane Wessels

[21*] Sitio web de Squid,* http://www.squid-cache.org/

[22] Sitio de Net-Snmp, http://www.net-snmp.org/

[23] *Video Codec Design. Iain E. G. Richardson.* Editorial John Wiley &Sons

[24] *Methodology for the subjective assessment of the quality of television pictures,* Recomendación ITU-R BT.500-11

[25] *Enrutamiento avanzado y control de tráfico en Linux,* Bert Hubert Netherlabs BV bert.hubert@netherlabs.nl

[26] Sitio web del Simulador "*Network Simulator 2*", http://www.isi.edu/nsnam/ns/

[27] *Arquitecturas distribuidas para sistemas de video-bajo-demanda a gran escala.*

Fernando Cores Prado [Universitat Autonoma de Barcelona]

[28] RFC:2326-Real Time Streaming Protocol (RTSP)

[29] RFC 1757- *A simple Management Network Protocol.* http://www.ietf.org/rfc/rfc1157.txt

[30] *Traffic Controller HOWTO*, http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html, The Linux Documentation Project.

[31] http://packages.ubuntu.com/karmic/ftpd-ssl

ANEXOS A

A.1 Función entrenar

Entrenar.c

Introducción

Este software está programado en lenguaje C y se encarga de realizar el entrenamiento de los clientes. Es parte del modulo estadístico y es llamado por el mismo cada vez que se desee entrenar a un cliente, quienes están diferenciados e identificados por una dirección IP.

La tarea del entrenamiento consiste en enviar los paquetes de prueba (pp) y los paquetes de video. Hacer los cálculos que determinan, a través de los pp, los valores que caracterizan el estado de la red (X), estos es *promedio de qn*, *varianza y percentil de qn y porcentaje de cola vacía.* Calcular los valores que caracterizan la QoS del video, estos son *retardo*, *jitter* y *pérdidas de paquetes*. Correlacionar los valores que caracterizan la red con los que caracterizan la QoS del video y construir la función $\varphi(X)$ que los relacionan, para que a través de ella se realice la predicción.

Funcionamiento

Entrenar tiene cuatro parámetros de entrada. El primero es una estructura cliente donde lleva los datos característicos como la IP. El segundo parámetro indica el puerto del cliente en que le llegaran los paquetes de entrenamiento Luego el tipo de video “HD” o “SD”, sabemos que el tráfico de un video codificado aumenta conforme aumenta la cantidad de movimiento del video, si el video tiene poca variación entre imágenes consecutivas entonces es de esperar un tráfico pequeño ya que solo se enviará la diferencia de información con el cuadro anterior. Por lo tanto es deseable el entrenamiento de varios “tipos” de videos con diferentes calidades y cantidades de movimientos y así clasificarlos y entrenarlos por separado, luego se hará una predicción dependiendo del “tipo” de video a visualizar. Por último tiene un parámetro de uso interno en la cuarta entrada. El parámetro de salida de este programa es un entero indicando que el entrenamiento se efectuó en forma correcta.

int entrenar(struct cliente *cli, char puerto[6], int calidad, int indice)

A continuación daremos en detalle la secuencia de eventos que efectúa este programa.

En primera instancia, una vez verificado que la entrada sea correcta, se llama a la función que envía una secuencia de paquetes de prueba (PP) a la ip destino, de tamaño 64 bytes y cada 10 ms, generando un tráfico de 52kbps. Seguido se envía una serie de secuencias de videos, tantas como parámetro de entrada tenga. En el payload de cada paquete, sea de video o PP, se le ingresa el número de secuencia y el tiempo de salida.

En el otro extremo está el cliente con un programa, llamado *cliente*, escuchando en forma continua a un puerto determinado, esperando la secuencia de paquetes de entrenamiento. Conforme va llegando los paquetes al cliente, se genera un archivo con

la información del payload y se le agrega el tiempo de arribo de cada paquete. Cuando no llegan más paquetes se envía al modulo estadístico, donde se está efectuando el entrenamiento, un archivo de nombre la ip del cliente, a través del protocolo ftp.

Una vez que el archivo este a disposición de *entrenar* se procede a realizar los cálculos de los parámetros de los PP y video.

El archivo tiene el siguiente formato:

Num_Secuencia	tiempo_salida	tiempo_arribo.
Num_Secuencia	tiempo_salida	tiempo_arribo.
.	.	.

Cada línea corresponde a un paquete. La información de los pp y los paquetes de video se encuentran en el mismo archivo. Para identificarlos se observan los números de secuencia así no se cometerán errores en caso de pérdidas de paquetes. Se separan los datos de video con los pp.

Luego procede al cálculo de los valores que caracterizan el estado de la red. Esto lo hace llamando a la función *calculoQn* a la cual importa de *calculoQn.h*. Explicaremos su funcionamiento en el próximo anexo, pero por lo pronto solo precisamos saber que tiene como entrada la secuencia, tiempo salida y tiempo arribo de los pp, también el número de pp que llegaron al cliente y además como entrada le pasamos un puntero de la variable global en la cual deberá escribir la salida. Dicha salida será el promedio de qn, la varianza de qn y el percentil de qn. También se dejo configurado las pérdidas de pp como posibilidad de variable que caracteriza la red X.

Para determinar la función $\phi(X)$ solo resta calcular los valores que caracterizan la QoS de un video. Pues *entrenar* calcula el retardo, jitter y pérdidas de paquetes utilizando la secuencia, tiempo de salida y tiempo de arribo de los paquetes de video.
Simplemente calculando:
Retardo = tiempo de arribo – tiempo de salida
Jitter = retardo paquete **(x)** *– retardo paquete* **(x – 1) = (***tiempo de arribo* **(x) -** *tiempo de arribo***(x – 1)) – (***tiempo de salida***(x) -** *tiempo de salida***(x – 1))**
*Perdidas de paquetes = Perdidas de paquetes+ secuencia***(x)** *– secuencia***(x-1)** *+ 1*

Una vez reunida esta información *entrenar* crea, en caso que no hayan sido creados, los archivos necesarios para ser ingresados en SVM, programa que correlacionará los datos de pp con los de video.
Como mencionamos anteriormente el entrenamiento consiste correlacionar los parámetros que caracterizan QoS de video con los parámetros que caracterizan X (estado de la red).
Para esto se debe correlacionar cada parámetro de video con todos los parámetros de X, y lo hacemos de a un parámetro de video por vez.

Entonces *entrenar* genera tres archivos con el siguiente formato, poniendo como ejemplo en retardo:

Retardo 1:*qn_promedio* 2:*qn_varianza* 3:*percentil_Z%* 4:porciento_cola_vacia

Se deben respetar los espacios entre valores. En forma análoga se forman los archivos de entrenamiento del *jitter* y las *pérdidas de paquetes*.

Como se puede ver cada entrenamiento genera una *línea de entrenamiento*, que junto a otras puestas en secuencia se forma un archivo de entrenamiento para el cliente y mientras más líneas mejor será el modelo que relaciona QoS de video con estado de la red

Retardo 1:*qn_promedio* 2:*qn_varianza* 3:*percentil_Z%* 4:porciento_cola_vacia
Retardo 1:*qn_promedio* 2:*qn_varianza* 3:*percentil_Z%* 4:porciento_cola_vacia

Este archivo refleja toda la historia de entrenamientos para un cliente.

Finalmente *entrenar* llama a la aplicación SVM, ingresándole como parámetro las rutas a estos archivos, y devolviendo un modelo que es la pura representación de la función que correlaciona el estado de la red con los parámetros de calidad.

A.2 Función calculo Qn

Función *CalculoQn*

Esta función se limita a calcular los valores que caracterizan el estado de la red: promedio de qn, varianza de qn, percentil qn y porcentaje de cola vacía. Es utilizado tanto en el entrenamiento como en la predicción.

Se le ingresa como entrada los datos de los pp; un array de secuencias, un array de tiempos de salida y un array de tiempos de arribo. Además se ingresa el número de los pp contados en el cliente, este valor puede ser menor que el número de pp que se enviaron si hubo de las pérdidas. Además como entrada le pasamos un puntero de la variable global en la cual deberá escribir la salida que será el promedio, varianza y el percentil de qn y porcentaje de cola vacía. También se dejo configurado en la salida las pérdidas de pp como posibilidad de variable que caracteriza la red X, pero no está activo.

A través de los datos de entrada *calculoQn* calcula qn como:

$$\hat{q}_n = \sum_{j=1}^{n} \left[\left(t_j^{0,N} - t_{j-1}^{0,N} \right) - \left(t_j^{i,1} - t_{j-1}^{i,1} \right) \right]$$

Luego calcula el promedio de qn como:

$$\hat{q}_n \_ promedio = \frac{\sum \hat{q}_i}{n}$$

Varianza de qn como:

$$\hat{q}_n \_ \text{var}ianza = \frac{1}{n}\sum_{1}^{n}(\hat{q}_i - \hat{q}_n \_ promedio)^2$$

Calcula el percentil Z% como el valor de qn para el cual un porcentaje Z% de los valores de qn quedan por debajo de dicho valor.

Por último se calcula el porcentaje de cola vacía como la cantidad de valores de qn que están cercanos a cero (a menos de 1 ms) dividido la cantidad de pp por cien.

A.3 Descripción detallada de instalación y configuración de Squid

Descomprimir los paquetes bajo /usr/local/
Posicionarse en /usr/local/squid-2.7.STABLE3
Loguearse como root
Para compilarlo se precisa tener instalado gcc, Perl y awk.
Ejecutar ./configure
make
Para instalarlo:
make install
Dar permisos a la carpeta /usr/local/squid/var y sus subcarpetas, donde se guardarán los logs en archivos que aún no han sido creados.
chmod 777 -R /usr/local/squid/var
Se crean los directorios Swap (se crea la carpeta /usr/local/squid/var/cache)
/usr/local/squid/sbin/squid -z
Se corre una prueba y automáticamente se crean los archivos store.log cache.log y cache.log bajo la carpeta /usr/local/squid/var/logs
/usr/local/squid/sbin/squid -N -d 1 -D
Para salir Ctrl-C
Si en la prueba anterior aparece la siguiente línea "Ready to serve requests" es que la prueba fue satisfactoria.
Dar permisos al archivo de configuración.
chmod 777 /usr/local/squid/etc/squid.conf
Editar squid.conf en caso de querer cambiarlo.
gedit /usr/local/squid/etc/squid.conf
Para correr el Squid
/usr/local/squid/sbin/squid
Se debería navegar sin problemas si no se hicieron cambio en el archivo squid.conf
Para parar el squid
/usr/local/squid/sbin/squid -k kill
Para parar el proceso desde Ubuntu (en caso que no funcione el comando anterior)
pkill /usr/local/squid/sbin/squid
pkill squid

Archivo de configuración (básico) de Squid para 1era aproximación de Proxy

```
http_port 3128


acl all src all

external_acl_type prueba ttl=1 children=20 %SRC /home/manuel/squid-2.7.STABLE3/helpers/external_acl/prueba_helper

acl aclExterno external prueba
http_access allow all   aclExterno
http_access deny all
```

A.4 Servidor VLC

Comenzamos instalando el vlc (versión 0.9.9A) y se procedió a intentar que otra máquina en la misma red pudiera acceder a un video en el servidor. Esto funcionó correctamente y se pudo transmitir tanto en forma de broadcast como de unicast. O sea que el servidor transmita a todo quien quiera escuchar o bien que el cliente pida para ver un video desde el comienzo.

Como nuestro sistema en primera instancia está pensado para video bajo demanda (VOD), estudiamos la manera de que VLC pudiera cumplir nuestros objetivos.
La idea de VOD es que el cliente controla la reproducción del video. Para ello, primero se crea un objeto vod en el archivo de configuración de vlc.

En caso de querer transcodificar el video original y emitirlo con otro formato se puede definir el codec .Por ejemplo h264 de calidad hd en imagen y mp3 en audio. También se puede ajustar el rate de salida, los cuadros por segundo y otros parámetros.

Lo mismo se hace para cada video que deseo agregar como objeto.

De esta manera permitimos que los usuarios puedan acceder al video desde su propia pc. Solo basta ejecutar el siguiente comando en el cliente

```
cliente% /usr/bin/vlc rtsp://ipservidor:5554/Mivod
```

Luego se abrirá el reproductor vlc y se podrá comenzar a ver el video con la posibilidad de pausar, retroceder y avanzar en el contenido.

El cliente usará el protocolo rtsp(Real Time Streaming Protocol) que es con el que vlc maneja el vod. Este protocolo utiliza un intercambio de información de control mediante TCP, donde negocia puertos,, origina sesiones, las libera y también controla la reproducción. Por otro lado utiliza UDP para enviar la información de video y audio.

Archivo de configuración VLC (con un solo objeto vod)

new Mivod vod enabled
setup Mivod input "file:///home/diego/Escritorio/Peliculas/ElCirculo.avi"

En caso de querer transcodificar el video original y emitirlo con otro formato se agregaría algo así:

"#transcode{venc=ffmpeg,vcodec=h264,vb=384,height=120,width=160,fps=20,aenc=ffmpeg,acodec=mp3,ab=64,channels=1}"

Donde quedó definido el codec (h264 de calidad high definition en imagen y mp3 en audio) también el rate y los cuadros por segundo.

Para cada video que deseo se pueda acceder debo crear un objeto vod.

Procedimiento para levantar VLC

Luego ejecuto el siguiente comando en el terminal:

diego@diego-desktop% vlc -I telnet --telnet-password videolan --rtsp-host 0.0.0.0:5554

[00000405] telnet interface: telnet interface started on interface 4212

Esto permite el acceso a la interfaz telnet del vlc.

diego@diego-desktop% telnet 192.168.0.2 4212
Trying 192.168.0.2...
Connected to 192.168.0.2.
Escape character is '^]'.
Password:
Welcome, Master
> load /home/diego/media/vlc_vod_config

Con esto permitimos que los usuario puedan acceder al video desde su propia pc ya que cargamos el archivo de configuración que el vlc usa y donde ya está creado el objeto vod llamado Mivod. Solo basta ejecutar el siguiente comando en el cliente.

cliente% /usr/bin/vlc rtsp://ipservidor:5554/Mivod

ANEXOS B

B.1 Protocolo rtsp

El “Real-Time Streaming Protocol” (RTSP)[28] es un protocolo a nivel de aplicación para control de entrega de datos en tiempo real. Está definido en el RFC 2326 e implementa un control remoto de red para servicios multimedia.

En una sesión de RTSP, el cliente RTSP, puede abrir y cerrar varias conexiones de transporte TCP con el servidor. Por ejemplo toda la información de control se traspasa mediante el establecimiento de una conexión TCP. También dispone de la posibilidad de enviar el contenido multimedia a través de TCP o a través de un protocolo de transporte no orientado a conexión como UDP. La operación de RTSP no depende del mecanismo utilizado para transportar el servicio multimedia, el cual podría RTP.

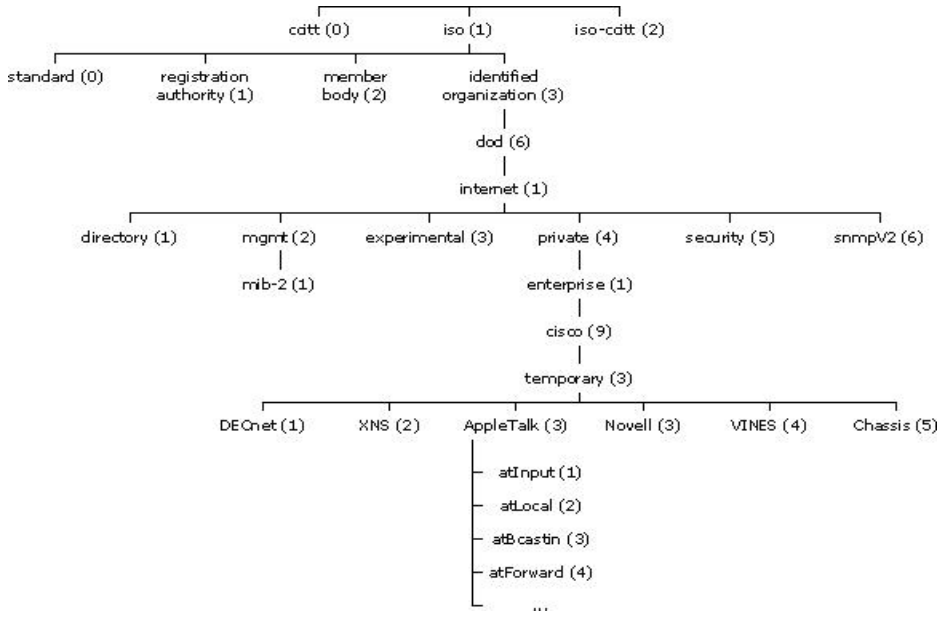
Una transacción completa de es por ejemplo la visualización de un video. Una sección típica de RTSP consiste en que un cliente envíe una petición de inicialización al mecanismo de transporte del flujo multimedia (SETUP), se comience el envío mediante PLAY o RECORD, y se finaliza mediante TEARDOWN.

El cliente usará el protocolo RTSP, que es con el que vlc maneja el VoD. El flujo proporcionado por el envío del video será a través de RTP sobre UDP y en un solo sentido, de servidor a cliente. Mientras que a través de TCP negocia puertos, origina sesiones y las libera y además el cliente maneja las acciones de play, pausa, rebobinado, etc.

B.2 SNMP (Simple Network Management Protocol)

SNMP [29] es un protocolo elaborado para el monitoreo y administración de distintos tipos de dispositivos de red.
Los dispositivos a ser administrados tendrán operando un agente SNMP que guarda la en variables u objetos, la información sobre su estado.

Dicha información se organiza en forma jerárquica, en forma de árbol y conforma una base de datos. Cada nodo del árbol corresponde a estructura MIB (Management Information Base) que alojara objetos o variables.



Provee un mecanismo para acceder a los objetos de una MIB para que puedan ser consultados y modificados. Además el protocolo ofrece la posibilidad de que los dispositivos de la red a ser monitoreados, envíen mensajes de notificación a una estación de gestión SNMP para indicar que se ha producido un cierto evento.

SNMP Define cinco tipos de mensajes de intercambio entre gestor y agente que se denominan PDUs (Unidad de datos de Protocolo):
1. Get-request. Utilizado por la estación de gestión para obtener el valor de una o más variables MIB del agente SNMP de la estación remota.
2. Get-next-request. Es similar a la anterior, con la diferencia que se obtiene el valor de una variable sin definir ésta explícitamente. De hecho se obtiene el valor de la variable que sigue a la especificada dentro de la ordenación de la MIB.
3. Response. Es la respuesta del agente a una petición del gestor devolviendo el valor de una o más variables.
4. Set-request. Constituye el mecanismo para que el gestor modifique los valores de las variables MIB de la estación remota.
5. Trap. Cuando se produce un determinado evento o condición en la estación remota, el agente envía un trap para notificarlo al gestor. Dado que el mensaje se envía de forma asíncrona y en cualquier momento, la estación de gestión debe monitorizar la red en todo instante.

SNMP utiliza UDP como protocolo de transporte por lo cual el gestor implementa un esquema de temporización y retransmisión para contemplar el hecho de la pérdida de los mensajes y solventar la falta de fiabilidad de UDP.
SNMP hace uso de dos puertos UDP, el 161 y el 162 para la recepción en el agente y el gestor
respectivamente, de esta forma es posible que en una estación operen simultáneamente el software de gestor y agente.

Pasos para la instalación

La herramienta net-snmp contiene el conjunto de aplicaciones necesarias para implementar snmpv1,snmpv2c y snmpv3.
Esta se puede descargar de los repositorios de ubuntu mediante el comando "sudo apt-get install snmp snmpd" el cual instalara todos los paquetes necesarios para implementar el cliente y servidor snmp respectivamente.

Para la configuración puede ser útil la aplicación "snmpconf" que construye automáticamente un archivo de configuración según las opciones seleccionadas.

Directivas usadas en el archivo de configuración (snmpd.conf) para lograr el comportamiento deseado del agente snmp:

agentaddress [<transport-specifier>:]<transport-address>[,...]
define la lista de direcciones en la que el agente escucha las solicitudes SNMP entrantes
Por defecto escucha en el puerto UDP 161 en todas las interfaces IPv4.

hostname[:port] o IPv4-address[:port]
Ejemplo:
Agentaddress 192.168.1.20:10000

Escuchara en el puerto udp 10000 en la dirección ip 192.168.1.20
En nuestro caso usamos la opción por defecto.

rocommunity COMMUNITY [SOURCE [OID | -V VIEW [CONTEXT]]]

Especifica una comunidad snmpv1 o snmpv2c a la que se le permitirá acceso de solo lectura.
Se puede restringir el acceso a la ip de origen (SOURCE) y al subárbol con raíz en la oid dada por OID.
Ejemplo: rocommunity test 87.65.43.21

Por defecto se permite acceso al árbol oid completo y a solicitudes de cualquier ip de origen.
Se configuro la opción por defecto y la comunidad "qp-qos" .

agentuser {USER|#UID}

Cambia al usuario definido por el nombre "USER" o por el user id "UID" luego de abrir los puertos en los que escucha solicitudes.

Este usuario debe tener permisos para poder ejecutar los scripts que implementan las respuestas a las consultas sobre los parámetros de calidad predichos para un usuario determinado.
En nuestro caso elegimos que el agente corra como usuario root.

exec [MIBOID] NAME PROG ARGS

Ejecuta el programa PROG (ruta absoluta al archivo ejecutable) con argumentos ARGS siendo accesibles, entre otros, los siguientes resultados bajo la rama ucdavis.extTable.extEntry :

- primera línea de la salida estándar (*ucdavis.extTable.extEntry.extOutput*)
- valor retornado (*ucdavis.extTable.extEntry.extResult*)
- comando ejecutado (*ucdavis.extTable.extEntry.extCommand*)

Si se especifica MIBOID los resultados serán accesibles debajo de dicha oid .

Ejemplo :

Si agregamos la siguiente línea al archivo de configuración:
exec foo /bin/sh /tmp/foo.sh -arg1

Al ejecutar en el terminal : snmpwalk -v 1 -c public localhost .1.3.6.1.4.1.2021.8.1 obtenemos :

```
UCD-SNMP-MIB::extIndex.1 = INTEGER: 1
UCD-SNMP-MIB::extNames.1 = STRING: foo
UCD-SNMP-MIB::extCommand.1 = STRING: /bin/sh /tmp/foo.sh -arg1
UCD-SNMP-MIB::extResult.1 = INTEGER: 0
UCD-SNMP-MIB::extOutput.1 = STRING: 123
UCD-SNMP-MIB::extErrFix.1 = INTEGER: 0
UCD-SNMP-MIB::extErrFixCmd.1 = STRING:
```

En el caso que hayan mas líneas exec, los índices en las oid's se irán correspondiendo según el orden en que están definidas en el archivo de configuración.
O sea, si por ejemplo quiero consultar el valor retornado a la salida estándar por el tercer script definido en el archivo de configuración, basta con ejecutar:
snmpget -v1 -c <comunidad> <ip> UCD-SNMP-MIB::extOutput.3

De esta manera, utilizamos esta funcionalidad para que el proxy pueda consultar al vía SNMP los valores de QoS predichos por el M.E. para cierto cliente.
La información de los índices asociados a cada cliente se extrae de un archivo.

Se definieron los scripts peticionCliente e ingresoCliente para avisar al programa principal del M.E. cuando un cliente está solicitando una conexión y cuando se acepta el acceso a algún cliente para que nuevamente se vuelvan a estimar los parámetros de QoS al resto de los clientes.

ANEXOS C

Contenido del CD

Adjunto a este documento se hace entrega de un CD con los códigos fuente de todos los programas implementados. También se incluye una guía de usuario para la instalación y uso del sistema.