Universidad Autónoma de Madrid
Escuela politécnica superior

Proyecto fin de carrera

EVALUACIÓN DE LA INFLUENCIA DEL MUESTREO Y DE LA PÉRDIDA DE PAQUETES SOBRE LA DETECCIÓN DE TRÁFICO SKYPE

Ingeniería de Telecomunicación

Diego Corral González
NOVIEMBRE 2012

EVALUACIÓN DE LA INFLUENCIA DEL MUESTREO Y DE LA PÉRDIDA DE PAQUETES SOBRE LA DETECCIÓN DE TRÁFICO SKYPE

AUTOR: Diego Corral González
TUTOR: Pedro María Santiago del Río

High Performance Computing and Networking Group
Escuela Politécnica Superior
Universidad Autónoma de Madrid
NOVIEMBRE 2012

Resumen

Resumen

En los últimos años ha ganado cada vez mayor importancia las aplicaciones de Voz sobre IP (VoIP). En especial, Skype, ha conseguido en pocos años una gran popularidad siendo utilizada por millones de usuarios de todo el mundo.

Este hecho ha provocado el interés tanto de la comunidad científica como de las operadoras de telecomunicaciones en analizar, clasificar y caracterizar el tráfico de dicha aplicación. Tales tareas son capitales para la gestión del tráfico en tiempo real, el cumplimiento de las restricciones de calidad de servicio (QoS) y por motivos legales (intercepción de las comunicaciones).

Skype utiliza un protocolo privado, ofuscado, donde los datos están cifrados, y funciona sobre puertos aleatorios, lo que dificulta su identificación. Tal dificultad de clasificar el tráfico Skype con las técnicas habituales, esto es, mediante el análisis de los puertos TCP o UDP, o analizando la carga útil de los paquetes mediante técnicas *Deep Packet Inspection* (DPI), ha hecho necesario la utilización de nuevas técnicas, que analizan las características estadísticas del tráfico y usan técnicas de aprendizaje automático.

Otro hecho importante es el aumento de la capacidad de los enlaces, llegando a existir redes con velocidades de hasta 100 Gb/s, lo que hace más complicado la clasificación del tráfico debido al gran número de información que se debe procesar.

Para poder soportar tales altas velocidades, se propone no procesar todos los paquetes que circulen por la red, aplicando políticas de muestreo o cuando aparecen pérdidas de paquetes en el enlace. Este proyecto tiene como objetivo el estudio de la precisión de detectores de tráfico Skype que se basan en técnicas de análisis estadístico del tráfico cuando se aplica muestreo de paquetes o hay pérdidas de paquetes.

En este proyecto, analizamos la precisión del detector de tráfico *Skypeness* al variar dos factores, a saber: por un lado, la política de muestreo o modelo de pérdida de paquetes y, por otro lado, la tasa de muestreo o probabilidad de pérdidas. En la experimentación, se han utilizado seis trazas públicas, que contienen tanto tráfico Skype como no Skype. Después de analizar los resultados, se observa una degradación inaceptable de la precisión del detector a medida que la cantidad de paquetes disponibles es menor (mayor tasa de muestreo). En particular, identificamos como factor relevante de esta degradación, la característica estadística del tiempo entre llegadas de los paquetes. Finalmente, se propone e implementa una modificación del algoritmo de detección que mitiga tal reducción de la precisión.

De este modo, este proyecto prueba que el muestreo puede ser una solución para la escalabilidad de la detección del tráfico Skype a tasas del orden de multi-10 Gb/s.

Palabras Clave

Clasificación de tráfico, Skype, calidad de servicio, tiempo real, aprendizaje automático, red de alta velocidad, muestreo de paquetes

Abstract

Last years Voice over IP (VoIP) applications have gained more and more relevance. Specially, Skype has become, in a few years, the most popular VoIP application, used by several million users.

This fact has awoken the interest of the research community and of telecom operators to analyze, classify and characterize Skype traffic. Such tasks are capital for traffic management in real time, for fulfillment the good quality of service (QoS) requirements and for legal purposes (lawful interception).

Skype uses a proprietary protocol, obfuscated, where data is encrypted and random port numbers are used, which difficults its identification. Such issue of classifying Skype traffic with traditional techniques, that is, making use of TCP or UDP ports or analyzing the packet payload using Deep Packet Insepection (DPI) techniques, has made it necessary the utilization of novel techniques, which analyze statistical characteristics of traffic and use machine learning techniques.

Another important fact is the ever increasing capacity of network links, reaching speeds up to 100 Gb/s, which makes it more difficult to classify Skype traffic due to the large amount of information to be processed.

In order to cope with such high speeds, we propose not processing every packet in the network, applying sampling policies or when there is packet loss in the link. This project aims to study the accuracy of Skype traffic detectors that are based on statistical analysis techniques traffic when packet sampling is applied or when there is packet loss in the link.

In this work, we analyze the accuracy of the traffic detector *Skypeness*, varying two factors, namely: on the one hand, the sampling policy or the packet loss model and, on the other hand, the sampling rate or the packet loss probability. In the experiments, we have used six public traces, which comprise both Skype and Non-Skype traffic. After analyzing the results, we observe an unacceptable degradation of the detector accuracy as the amount of available packets decreases (greater sampling rates). Particularly, we identify a key factor in such degradation: the packet interarrival times. Finally, we propose and implement a modification of the detection algorithm which mitigates such accuracy reduction.

Thus, this work shows that sampling may be a solution for the scalability of Skype traffic detection at multi-10 Gb/s rates.

Key words

Traffic classification, Skype, quality of service, real-time, machine learning, high-speed network, packet sampling

Agradecimientos

Quiero agradecer la ayuda y el apoyo a todas aquellas personas que han estado a mi lado tanto en los momentos fáciles como en los difíciles, durante estos años que ha durado mi etapa universitaria.

Muchas gracias a mi tutor, Pedro María Santiago, por toda la ayuda prestada en la realización de este proyecto, por todo el tiempo que me ha dedicado, que no ha sido poco, por todo lo que he aprendido a su lado y por hacer que tengas más ganas de aprender y mejorar.

Gracias a mis padres y a mi hermana que han estado a mi lado en todo momento, dándome fuerzas y ánimos en los momentos donde las cosas no iban tan bien.

En especial, a mi novia Marta, porque me ha dado la ilusión por acabar esta carrera, me ha soportado en los momentos de bajón y me ha apoyado en todo momento, animándome siempre con una sonrisa.

A mis amigos de toda la vida por estar cerca siempre que los he necesitado. Y a mis amigos con los que he estado desde el primer hasta el último día de la carrera: Nacho, Tomás y Dani, que han hecho que todo fuera más fácil, a los que deje por el camino, y a los que fui conociendo más tarde, que han conseguido que ir a la universidad fuera algo agradable y que han hecho que me lleve muy buenos amigos y recuerdos.

Muchas gracias a todos.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

1.1. Motivación del proyecto

La clasificación del tráfico IP es un campo de investigación de gran importancia, y cobra especial interés si se realiza a tiempo real ya que facilitaría en gran medida la gestión y mantenimiento de la red a los proveedores de servicio de Internet (ISPs) [1]. Uno de los principales objetivos de tales tareas es la mejora de la calidad de servicio (QoS).

Por otra parte, la imposición por parte de los gobiernos y autoridades legales de la intercepción de las comunicaciones (la llamada *lawful interception*), hace que los proveedores deban ser capaces de dar información acerca de un usuario en cualquier punto de la red y en cualquier momento. Lo cual hace más patente la necesidad de identificar y clasificar tráfico por aplicación.

En los últimos años, hemos visto como las aplicaciones Voz sobre IP (VoIP) incrementaban sustancialmente su número de usuarios, y, entre ellas, Skype es la que ha conseguido mayores registros. Este fenómeno ha provocado el interés por parte de las compañías telefónicas y de la comunidad científica de analizar, clasificar y caracterizar el tráfico Skype [2].

Anteriormente, resultaba sencillo la clasificación del tráfico gracias a los números de puertos TCP o UDP bien conocidos, o analizando la carga útil del paquete (DPI, *Deep Packet Inspection*). El uso de estas técnicas ya no es factible: gran parte de las aplicaciones usan puertos aleatorios, la carga útil está cifrada o el proceso necesario para la inspección de la carga útil no puede ser llevado a cabo en tiempo real en redes de alta velocidad. Por estas razones, recientemente se han propuesto algoritmos que se basan en características estadísticas del tráfico [1].

A parte de los problemas mencionados anteriormente, la caracterización del tráfico Skype plantea otros retos particulares. Esta aplicación utiliza un protocolo propietario, ofuscado, los datos están cifrados, puede usar como protocolo de transporte tanto UDP como TCP sobre puertos aleatorios y transporta distintos tipos de contenido (voz, video, texto y datos) con características distintas [2], [3].

En la actualidad existen diferentes algoritmos para la clasificación y detección de este tipo de tráfico con grandes resultados de precisión [3]. Sin embargo, el estudio de estas técnicas no se ha centrado en el rendimiento de las mismas desde un punto de vista computacional. En la actualidad, la velocidad de los enlaces se ha incrementando notablemente debido al gran número de aplicaciones que consumen gran ancho de banda [4]. De este modo, los clasificadores

de tráfico tienen que ser capaces de actuar en entornos de alta velocidad y, por tanto, además de ser precisos, tienen que ser eficientes desde un punto de vista de tiempo de proceso [5].

En un entorno de muy alta velocidad (10-40-100 Gb/s), las tasas de recepción de paquetes se encuentran en el orden de decenas o centenas de millones de paquetes por segundo. Procesar la totalidad de estos paquetes para extraer la información necesaria para la detección de los flujos Skype se presenta como una tarea ardua. Por ello, se plantea la posibilidad de procesar sólo una fracción de los paquetes, ya sea porque se produzcan pérdidas en la recepción o porque se apliquen técnicas de muestreo.

1.2. Objetivos y enfoque

El principal objetivo de este proyecto es evaluar cómo afecta la tasa de muestreo y la pérdida de paquetes a la detección del tráfico Skype.

Para poder llegar a realizar dicha evaluación se irán realizando diversas etapas y se deberá ir cumpliendo con una serie de objetivos intermedios, a saber:

- Estudio del estado del arte sobre clasificación de tráfico.
- Instalación de los métodos de clasificación de tráfico Skype en una maqueta de alta velocidad.
- Implementación de las técnicas de muestreo y emulación de pérdidas de paquetes.
- Evaluación de la precisión de los clasificadores en presencia de muestreo y pérdidas.
- Análisis de los resultados y extracción de conclusiones.

1.3. Metodología y plan de trabajo

En este apartado se habla del procedimiento que se ha seguido en la elaboración del proyecto y consecución de los objetivos, así como de las diferentes etapas en las que se ha dividido el proyecto.

- **Estudio del arte sobre métodos de clasificación del tráfico Skype**. Para la realización de esta etapa se realiza la lectura de diferentes documentos acerca del tema tratado. Primero, es importante entender la importancia de la clasificación del tráfico, los diferentes métodos que existen y los problemas que se pueden encontrar. Después nos centramos en la aplicación Skype, su funcionamiento, el por qué se desea caracterizar su tráfico, así como la complejidad para hacerlo, y el entendimiento de los diversos métodos de detección que se exponen en la literatura.
- **Implementación e instalación del detector de tráfico Skype**. Una vez conocidos y entendidos los algoritmos para la clasificación del tráfico Skype expuestos en la literatura, pasamos a instalar los ya implementados. Esta instalación se lleva a cabo en un entorno de alta velocidad de 10 Gb/s.
- **Estudio de la precisión de los detectores desarrollados**. Para comprobar el buen funcionamiento de los distintos métodos implementados e instalados en el punto anterior, se realiza un experimento que nos permite observar la precisión sin la presencia de pérdidas o muestreo, realizando una comparativa de los diferentes detectores.

- **Experimentación basada en la tasa de muestreo**. En este punto se somete a los detectores a un conjunto de tráfico (tanto de la aplicación Skype como de otro tipo de aplicaciones), en condiciones de pérdida de paquetes y muestreo para ver cómo afecta a la precisión del detector. Se estudian distintas situaciones de pérdida de paquetes (variando la tasa de pérdidas y la rafagosidad de las mismas) y de muestreo (variando la tasa de muestreo y el tipo de muestreo).

- **Análisis de los resultados y conclusiones finales**. Después de la realización del experimento se extraen unas conclusiones claras y acertadas acerca de lo sucedido. De esta forma, se evaluará si es posible la utilización de los detectores estudiados, con una pérdida razonable en la precisión, en redes de alta velocidad (10-40-100 Gb/s). Con estas conclusiones y recopilando los datos del resto de etapas anteriores, se redacta la memoria final del proyecto.

1.4. Medios a utilizar

En este apartado nombraremos las distintas herramientas que se han utilizado para la elaboración del proyecto:

- **Herramientas Hardware:**
  - Ordenador personal para programación y desarrollo.
  - Servidores para la ejecución sistemática de los programas desarrollados.
- **Herramientas Software:**
  - Sistema operativo *GNU/Linux*.
  - Entorno de desarrollo integrado.
  - Procesador de texto.
  - Compilador C *gcc*.
  - Gestor de compilaciones *make*.
  - Librería *pcap*.
  - Herramienta de monitorización de red *tstat*.
  - Clasificador de tráfico *Skypeness*.

1.5. Organización de la memoria

La memoria se divide en cinco capítulos, en el primero se explica en qué consiste el proyecto, cuál ha sido la motivación que ha llevado a su desarrollo, cuales son los objetivos que se persiguen, el procedimiento que se ha seguido para la elaboración del trabajo y las herramientas utilizadas.

En el siguiente capítulo se realiza un estudio del arte, acerca de la detección de tráfico Skype, de la utilización de técnicas de muestreo en este campo y de la simulación de un estado de pérdidas en aplicaciones de tiempo real. En el capítulo 3 se analiza tres herramientas de detección de tráfico Skype, como son Tstat, L7filter y Skypeness.

Después, en el capitulo cuatro, se explican los experimentos que se van a llevar a cabo, se realiza un análisis estadístico acerca de dos parámetros del trafico Skype y se exponen los resultados obtenidos.

En el capítulo cinco se explica una mejora llevada en el detector Skypeness, exponiendo los nuevos resultados que se han logrado y para finalizar, en el capítulo 6, se extraen conclusiones claras y concisas de todo lo evaluado anteriormente que muestran si se han alcanzado los objetivos que se propusieron al inicio del proyecto.

2

Estado del arte

2.1. Métodos de clasificación del tráfico Skype

La primera parte de este capítulo se basa en realizar el estudio del estado del arte referido a los métodos de clasificación y detección del tráfico Skype. Para ello empezaremos hablando de la importancia que tiene la clasificación del tráfico de Internet y los diferentes métodos para su realización. Después, hablaremos del funcionamiento de la aplicación Skype, esto nos servirá para poder entender con mayor facilidad sus diferentes clasificadores y detectores.

La clasificación del tráfico de Internet es un tema que ha suscitado el interés tanto de la comunidad científica como de los operadores de red [1]. A los primeros, a para analizar los cambios que se producen en Internet, para entender los mecanismos y el funcionamiento de las nuevas aplicaciones, y para estudiar el tráfico que generan. Para los operadores de red y proveedores de servicios de Internet también es un tema a tener en cuenta, y más si se realiza en tiempo real, ya que les permitirá gestionar las redes, corregir errores, o detectar intrusos en el menor tiempo posible, en definitiva, para poder dar una mejor calidad del servicio a sus usuarios.

También ha cobrado mayor importancia dicha clasificación debido a la llamada *lawful interception*. Esto es, los proveedores deben poder dar información sobre algún usuario de interés de la red a los gobiernos, capaces de capturar el tráfico de esa persona, y saber qué aplicación está utilizando o con quién está hablando.

Para poder comparar diferentes técnicas para la captura de tráfico presentamos una serie de términos que nos servirán para poder medir la precisión del detector [1], [6]. Algunos de estos términos son:

- **Falsos negativos (FN):** porcentaje de miembros de la clase que queremos detectar clasificados como otra.
- **Falsos positivos (FP):** porcentaje de miembros de otra clase clasificados como la clase a detectar.
- **Verdaderos positivos (TP):** porcentaje de miembros de la clase a analizar clasificados correctamente.

- **Verdaderos negativos (TN):** porcentaje de miembros de otra clase correctamente clasificados.
- **Recall:** porcentaje de miembros de la clase a detectar correctamente clasificados.
- **Precision:** porcentaje de miembros que verdaderamente tienen la clase buscada, de entre todas las clasificadas de ese tipo.

Un buen clasificador será aquel que tenga menor porcentaje de falsos positivos y negativos.

También es importante saber en qué medida realizamos la comparación, ya sea en bytes o en flujos. Ambas opciones son válidas, y dependerá de la finalidad del detector el utilizar una u otra [1]. Por ejemplo, si hay aplicaciones que utilizan un número pequeños de flujos, en ese caso nos interesa más analizar la precisión en términos de bytes.

Anteriormente, los clasificadores se basaban en estudiar la quíntupla de cada paquete, es decir, en las direcciones IP origen y destino, en los número de puerto TCP o UDP origen y destino, y en el tipo de protocolo de transporte utilizado. Pero esto tiene sus limitaciones, ya que las nuevas aplicaciones utilizan puertos impredecibles y aleatorios, y cada vez hay menos aplicaciones que utilicen puertos conocidos [1]. Al ocurrir esto, se pasó a analizar el la carga útil (*payload*) de todos los paquetes. Los inconvenientes de esta técnica son el alto coste computacional de analizar grandes cantidades de flujos de datos y que algunas aplicaciones utilizan protocolos privativos o mecanismos de cifrado de datos, además se puede violar leyes de privacidad, haciendo muy difícil utilizar estas técnicas de clasificación de tráfico.

Por estas razones se empezó a buscar nuevos métodos de clasificación. En estos momentos, se utilizan características estadísticas del tráfico, tales como la duración del flujo, el tiempo de llegadas entre paquetes o el tamaño de los paquetes [2], [5], [6], [7]. El hecho de tratar con un gran número de datos, patrones de tráfico y espacios multidimensionales hizo necesario la utilización de técnicas *Machine Learning*.

Éstas técnicas son utilizadas para buscar y describir patrones en un conjunto de datos. Tienen la habilidad de aprender automáticamente a partir de la experiencia y mejorar sus conocimientos con un mayor número de muestras. A partir de unos datos entrantes es capaz de obtener características de diferentes tipos, es decir, modelos de clasificación. Debido a esto, se utilizó en muchos campos como el de la medicina, marketing, y más tarde en las telecomunicaciones.

Se definen cuatro tipos de aprendizajes: clasificación (o aprendizaje supervisado), *clustering* (o aprendizaje no supervisado), asociación y predicciones numéricas [1]. Los más utilizados para la clasificación del tráfico son el aprendizaje supervisado y no supervisado.

El primer tipo se basa en dos etapas, una de entrenamiento que se ocupa de crear un modelo para la clasificación, y una segunda de test, donde a partir de los datos de la primera etapa clasifica instancias no vistas hasta entonces. Este método es eficiente cuando se busca identificar una aplicación de interés.

El aprendizaje no supervisado no utiliza clases predefinidas como lo visto anteriormente, si no que descubre patrones sin un conocimiento previo y crea grupos de instancias con propiedades similares, *clusters*.

Uno de los grandes desafíos es conseguir realizar la clasificación de tráfico IP en tiempo real, pero existen inconvenientes, como son el clasificar el flujo sin la información completa del mismo y las capacidades computaciones necesarias para hacerlo. Por esto, se investiga cómo resolver este problema, ya sea utilizando solo los primeros paquetes del flujo, y realizando un uso eficiente de la memoria y el procesador. Todo para conseguir reducir el coste computacional y poder realizar una clasificación eficiente y de gran precisión.

Después de haber presentado la importancia de los clasificadores de tráfico y explicado los puntos claves de ellos, así como su funcionamiento, ahora nos centraremos en la aplicación Skype y sus detectores de tráfico. En estos últimos años, hemos visto como las aplicaciones VoIP multiplicaban su número de usuarios en poco tiempo, destacando entre ellas, Skype [2]. Este fenómeno ha hecho que la comunidad científica y los operadores de telecomunicaciones hayan realizado diversos estudios para analizar y caracterizar el tráfico de dicha aplicación, ya sea para investigar los mecanismos y su funcionamiento, como para entender el comportamiento seguido por los usuarios Skype, pero, en cualquier caso, esto no es una tarea sencilla.

Skype cuenta con diferentes servicios: comunicación de voz y video, chat y transmisión de archivos; pudiendo realizar llamadas entre diferentes usuarios de Skype (End-to-End) o entre un usuario y un terminal de telefonía convencional (End-to-Out). En lo que se refiere a la comunicación de voz puede utilizar diferentes códecs, que se han conseguido descubrir y analizar sus características, siendo el más popular el código *Sinusoidal Voice Over Packet Coder* (SVOPC) [2], [6].

La aplicación utiliza un protocolo privado, con técnicas de cifrado y ofuscación de datos. Puede utilizar TCP o UDP en la capa de transporte, siendo más normal la utilización de UDP [5]. TCP solo es usado cuando existen restricciones de *firewall* sobre UDP. También decir que tiene diferentes versiones de software con características y comportamientos diferentes.

Skype a diferencia de otras aplicaciones de VoIP se basa en una arquitectura de red *peer to peer* (P2P), en vez de usar el modelo cliente-servidor [3]. La red P2P Skype está compuesta por nodos ordinarios, supernodos y servidores dedicados [3], [8]; donde los nodos ordinarios serían los clientes y los supernodos funcionarían como *switches*. Dentro de los servidores, los podemos dividir en tres tipos: *login servers* para recoger la información del usuario y responsables de su autentificación, *update servers* para revisar si hay disponible una nueva versión de la aplicación e indicar la dirección IP del servidor Skype, y *buddy-list servers* utilizado para guardar la información referida a los contactos de un usuario.

Una vez un cliente lanza Skype se establece una conexión con un supernodo que será el responsable de él, para ello primeramente investiga mediante mensajes intercambiados con otros supernodos, cual está en funcionamiento y cual le acepta como cliente. Al principio también contacta con el login server para la identificación. Durante el tiempo que el usuario está online hay un intercambio de mensajes periódicos entre el host del usuario y el supernodo seleccionado [3]. La figura 2.1 muestra la arquitectura de la red Skype.

Gracias a diferentes estudios se ha conseguido caracterizar los diferentes mensajes que se intercambian entre clientes Skype, y entre el cliente y los supernodos [8], [9]. Nosotros presentaremos varios de estos mensajes:

- **Skype UDP Ping:** mensaje intercambiado, lanzado por todos los clientes, que se realiza periódicamente. Hay flujos UDP que solo transportan este tipo de mensajes.
- **Skype UDP Probe:** mensaje intercambiado cuando se lanza Skype, para descubrir supernodos y características de la red como, presencia de *Network Address Translations* (NATs) o *firewalls*. Este mensaje se repite periódicamente hasta estar seguro de haberse conectado a un supernodo disponible.
- **Skype TCP Handshake:** después de haber sido seleccionado un supernodo, el cliente inicia una conexión TCP con él para saber si la red de acceso Skype está disponible. Si no se pude establecer dicha conexión, el cliente usará el puerto 80 o 443.
- **Skype TCP Authetication:** mensaje intercambiado entre el cliente y el login server para realizar la fase de autentificación.

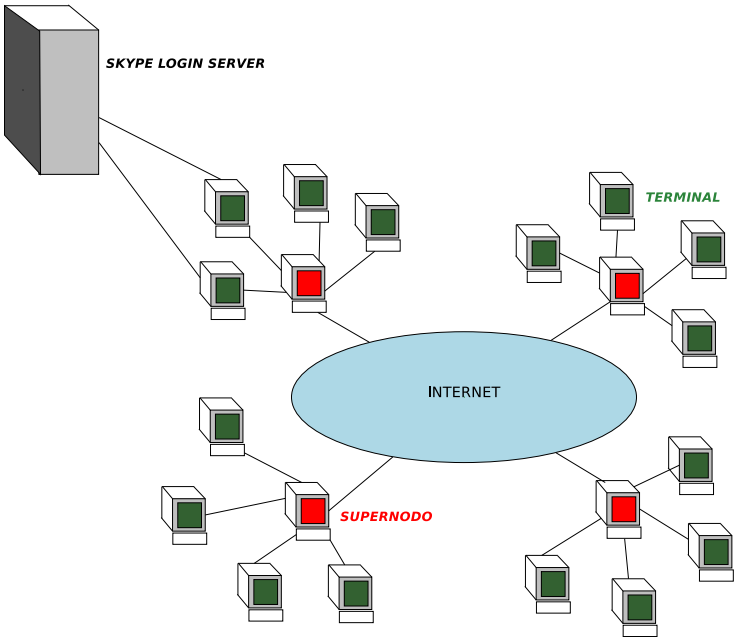


Figura 2.1: Arquitectura de red Skype

- **Skype http Update:** mensaje para comprobar si hay actualizaciones del software. El mensaje está cifrado y depende de la versión utilizada de Skype.

A este tipo de mensajes se les conoce como tráfico de señalización, y tiene especial importancia para la detección del tráfico Skype por sus características y por una serie de patrones que se repiten, como el tiempo entre mensajes, el número de bytes, etc. Este tipo de flujos de datos tiene como función el mantenimiento, gestión y descubrimiento de la red y de nuevos nodos.

También es importante hablar del modelo de fuente de Skype, es decir, de cómo se generan los datos desde el origen. La información se genera en bloques que se multiplexan en tramas, que después se comprimen y se le aplica el cifrado, y por último se le añade una cabecera no cifrada llamada *Start of Message* (SoM) [8], [9]. Para finalizar el mensaje es encapsulado en segmentos UDP o TCP.

Debido a los problemas tales como la utilización de un protocolo privado, ofuscado y cifrado que usa puertos aleatorios, o el diferente comportamiento de las diferentes versiones de la aplicación, hace que utilizar algoritmos para la clasificación del tráfico Skype basados en los puertos tradicionales y el payload de los paquetes no sea suficiente. Este hecho ha provocado que haya que buscar otros métodos. La comunidad científica propone nuevos enfoques basados en las características estadísticas del tráfico y utilización de técnicas de Machine Learning, para la clasificación y detección.

Por lo explicado anteriormente, se han hecho varias investigaciones acerca de las características del tráfico Skype y la importancia de ciertos parámetros del flujo de datos [2], [5], [6], [7], como pueden ser:

- *Bitrate*: tasa de bits generados en la capa de aplicación en un intervalo de un segundo.
- Tiempo entre paquetes: tiempo entre dos paquetes pertenecientes al mismo flujo.
- Longitud de los paquetes: número de bits que se encuentran en el payload TCP o UDP.

Estos parámetros pueden variar dependiendo de las características de la red.

En el artículo [3] se nos presenta un algoritmo para la identificación del tráfico Skype basado en los flujos de datos y en la extracción de información a nivel de paquete, y no basada en los puertos conocidos o en el payload de los paquetes. Su método se divide en dos pasos: primero descubrir candidatos host Skype y luego buscar llamadas de voz. Para la búsqueda de candidatos hay tres técnicas diferentes: buscar conexiones específicas Skype, buscar flujos de señalización entre clientes y supernodos, y buscar relaciones UDP, que son los mensajes periódicos entre el cliente y sus contactos. Se utilizará el tercer método por proveer mayor éxito en el proceso de búsqueda. Cuando un usuario realiza una llamada con otro se le asigna un puerto aleatoriamente antes de que realice la llamada y se mantiene después de ella. Habría que saber cuándo un usuario está inactivo o cuando está realizando una llamada, esto se puede conseguir gracias al tamaño de los paquetes y el número de ellos. Una vez identificada la relación UDP podemos determinar la dirección IP del host y el puerto de comunicación del cliente.

Por último, para buscar llamadas de voz y poder clasificar correctamente el tráfico Skype debemos fijarnos en usa serie de características de él: el ancho de banda, el número de paquetes transmitidos y el tiempo entre llegadas de los paquetes. De ellas se sacan unos valores umbrales que nos servirán para identificar las llamadas Skype. Una dificultad adicional es la clasificación de dicho tráfico en tiempo real.

Hay otros autores que se han centrado solo en el flujo de datos, y han hecho un estudio sobre cómo ha ido evolucionando la red Skype, como en el artículo [8]. En este artículo se presenta un algoritmo para identificar clientes y supernodos Skype basado en la detección de cierto tráfico de control. La identificación a nivel de flujo permite realizar un mejor estudio a gran escala sobre la red que los enfoques basados en paquetes. Se realiza el estudio sobre una serie de datos obtenidos desde 2004 a 2009, para observar los cambios sufrido por la red Skype, este es el primer estudio exitoso sobre el comportamiento de la red Skype. El algoritmo desarrollado analiza ciertos mensajes intercambiados entre clientes Skype y supernodos. Los dos mensajes analizados son UDP probe y TCP handshake, explicados en profundidad anteriormente. Gracias a las características de este tipo de mensajes el algoritmo es capaz de detectar clientes y supernodos. Este algoritmo tiene algunas limitaciones, ya que no puede identificar conexiones cuando se bloquee el protocolo UDP o usa otro método de handshake. Se implementa este algoritmo en un detector, llamado *snack*, y se examina después con diferentes métodos de evaluación: sobre la observación de una red y los cambios que se producen en ellas, y se calcula las tasas de falsos positivos y negativos. También se evalúa los falsos positivos y negativos mediante la identificación de supernodos activos. Después se compara el método basado en flujos frente a otro basado en paquetes. Mediante el primer método se consigue reducir el número de datos y se obtienen resultados más eficientes en grandes redes, como por ejemplo en la red Skype. Para medir el tamaño de una red se evalúa cuantos supernodos se detectan dentro de ella durante un espacio de tiempo. Y por último, se llega a la conclusión del mejor funcionamiento del detector basado en flujos que en paquetes en grandes redes.

Los autores de [6] nos presentan un detector cuya finalidad es realizar una rápida clasificación del tráfico Skype, solo observando los primeros cinco segundos del flujo de datos, y con una precisión alta sabe si pertenece a Skype o a otra aplicación. Gracias a técnicas de aprendizaje automático se busca identificar Skype casi a tiempo real. Para ello se entrena al clasificador con un tamaño de ventana de 1 a 10 segundos y después se evalúan los resultados. Para capturar el tráfico Skype utilizan el comando *tcpdump*, y gracias a un software llamado *Netmate* separa el tráfico determinando su quíntupla. Para la clasificación de tráfico se utiliza un clasificador de árbol de decisión ya que obtuvo mejores resultados. Las características más efectivas para realizar la clasificación son la longitud del paquete, estadísticas de la longitud del paquete como el máximo, mínimo, media, desviación estándar, y tiempo entre llegadas de paquetes. Después se realizan varios experimentos usando solo una clase de características, luego dos y por último

las tres, para obtener conclusiones para realizar una rápida clasificación. Siendo la característica más efectiva es el tiempo de llegadas entre paquetes. Se descubre que usando las tres clases se obtenían unos porcentajes de precisión cercanos al 100 % usando una duración de ventana de cinco segundos o más. Con esto se muestra que es posible una rápida clasificación con gran exactitud.

Uno de los puntos importantes en los detectores es la capacidad de realizar en tiempo real la clasificación. El clasificador que analizaremos ahora es capaz de rellenar ese hueco, utilizando diferentes enfoques a los vistos hasta hora, y es el que se presenta en el artículo [7]. En este paper se presenta dos técnicas complementarias para la detección de tráfico Skype en tiempo real. Después para analizar la precisión de estas herramientas se las ha comparado con otra técnica que los autores han desarrollado basada en DPI. Esta última se ha probado en diferentes redes obteniendo una gran efectividad en la identificación del tráfico Skype. Primero se nos explica el funcionamiento acerca de cómo se origina el tráfico Skype, explicado anteriormente, y una serie de parámetros que determinan las características del tráfico generado por el origen: la tasa binaria usada por la fuente, el tiempo de llegada entre dos paquetes del mismo flujo y el factor de redundancia, que es el número de los últimos bloques que Skype retransmite junto con el bloque actual. Dichos parámetros pueden cambiar dependiendo de las condiciones de la red, pero tienes unos valores similares dependiendo del códec utilizado. Cuando se utiliza TCP en la segmentación el contenido de todos los mensajes está cifrado siendo muy complicada su identificación, en cambio cuando se usa UDP hay una porción del mensaje que se puede identificar observando el payload, a esto se le llama Start of Message. También se han conseguido identificar varios campos en los mensajes E2E sobre UDP: ID, FUN y FRAME, una serie de bytes que se encuentran en los segmentos UDP. Y en las llamadas E2O después de un número inicial de mensajes, los primeros cuatro bytes del mensaje se repiten, *Conection IDentifier*.

A partir de estas características y aspectos se basan los tres clasificadores desarrollados. El primer clasificador referenciado como *Chi-Square* se enfoca en los mecanismos de cifrado, más concretamente analiza el contenido del mensaje aleatorizado por el cifrado. Esta técnica nos permite distinguir el tráfico generado por los clientes Skype y darnos diferentes características y comportamientos del contenido del mensaje después del cifrado, es decir, obtener patrones del mensaje. En el caso de UDP, detecta si es una llamada E2E o E2O. El segundo clasificador, *Naive Bayes Classifier*, se basa en la caracterización estocástica del tráfico y en las propiedades del códec de voz usado y del framer. Las principales características utilizadas son la longitud del paquete, el tiempo entre llegadas de los paquetes y del tamaño de los mensajes, el valor de éstas dependerán del códec utilizado y el framer. Y por último el clasificador basado en el payload, que explota el conocimiento del formato de las cabeceras del protocolo, pero con la problemática del cifrado y la ofuscación del mensaje. Por ello, con ésta técnica analizaremos la parte del flujo que no está cifrado y del que se conoce ciertas características, tanto de los mensajes de datos como de los mensajes de señalización. Se llega a la conclusión que Naive Bayes es muy efectivo identificando todo el tráfico de voz de cualquier aplicación, el Chi-Square para identificar el tráfico Skype sobre UDP y los dos en conjunto para detectar tráfico de voz Skype tanto sobre UDP como sobre TCP. Es de gran importancia el clasificador Chi-Square ya que resuelve el problema de la aleotorización introducida en el tráfico Skype y ayudará a entender mejor dicha problemática.

Otro artículo interesante [9], presenta un herramienta de detección que consigue mejorar las prestaciones del último clasificador visto. Se desarrolla un algoritmo de clasificación de tráfico Skype que también trabaja en tiempo real y que es capaz de distinguir distintas actividades de Skype, como llamadas E2E o E2O, transferencias de archivos, o tráfico de señalización. Este algoritmo se basa en enfoques estadísticos y en patrones de tráfico. Es importante remarcar el estudio realizado sobre el tráfico de señalización, el que se nos muestran diferentes tipos de mensajes, que hemos visto antes, como UDP ping, UDP probe, TCP handshake, etc. Gracias

a diferentes características de este tipo de tráfico, como el número de mensajes intercambiados o el número de bytes de cada uno de ellos, se ha conseguido desarrollar este algoritmo tanto para flujos UDP como TCP. También hay otros factores importantes como los campos del Start of Message, tiempo entre llegadas de los paquetes, la quíntupla del flujo o el número de bytes enviados por la origen y el destino. En el caso de TCP como todo el tráfico está cifrado se ha basado en estadísticas del tráfico. Por último se realiza un experimento para analizar la precisión del algoritmo desarrollado y se compara con otras técnicas que aparecen en el estado del arte. Se observa que los mejores resultados acerca de falsos positivos y negativos los obtiene dicha técnica tanto sobre UDP como TCP, consiguiendo mejorar los resultados de todos los anteriores detectores. Después estos escritores desarrollaron la herramienta *Skype-Hunter* y explicada con más detalle en el paper [10], basada en el algoritmo anteriormente contado, además realizaron un serie de experimentos para analizar en mayor profundidad los resultados obtenidos con dicha herramienta de detección.

Hemos visto que existen algoritmos para la detección de tráfico Skype que consiguen grandes resultados. Pero también hemos observado que requieren un alto coste computacional, y esto haría muy dificultoso la clasificación del tráfico en redes de alta velocidad, como las que existen en la actualidad. El artículo [5] se centra en analizar hasta que velocidad funcionaría correctamente un algoritmo que los autores han modificado, y como se conseguiría llegar a velocidades mayores. El detector llamado *Skypeness* utilizado se basa en técnicas estadísticas. Observa y estudia el comportamiento seguido por el flujo Skype, es decir, el tamaño de cada paquete, el tiempo de llegada entre ellos o el bitrate. De estos datos se sacan unas medias y unos valores umbrales, de manera muy exhaustiva, que nos servirán para poder saber si los datos pertenecen a Skype o a otra aplicación. Este detector solo funciona para flujos UDP Skype y no para TCP, ya que TCP se utiliza en casos muy particulares. El hardware donde se utiliza el algoritmo es sobre un servidor de propósito general que consta de 4 AMD Opteron 6128 procesadores trabajando a 2 GHz. Cada procesador cuenta con 8 núcleos y la memoria total está compuesta por 32x4 GB DDR3. Se proporciona una arquitectura NUMA, donde la memoria se divide en grupos, uno por CPU, llamado nodos NUMA. El software utilizado es Ubuntu 10.04 Linux Server (64 bits). Por tanto, el detector Skype trabaja a 1 y 3.7 Gbps leyendo de la tarjeta de red y de memoria. Y se ha conseguido llegar a 45 Gbps con 4 de los 8 núcleos, obteniendo un porcentaje de falsos negativos del 6 por ciento en el peor caso y 0 por ciento de falsos positivos. Estos resultados muestran que se puede realizar una correcta clasificación en redes de alta velocidad utilizando el hardware adecuado. Los autores esperan poder aplicar esta metodología sobre otras clases de tráfico, como P2P, y con otras técnicas de detección como *Deep Packet Inspection* (DPI).

Sobre esta última técnica, DPI, se habla en el artículo [4] y su aplicación en redes de alta velocidad como las habladas en el estudio anterior. La técnica Deep Packet Inspection (DPI) se ha considerado de un alto coste computacional, por eso, este documento realiza un estudio para reducir este coste sin afectar a la precisión y pudiendo utilizar esta técnica en redes de alta velocidad. Este método de clasificar tráfico analiza los datos en la capa de aplicación, teniendo gran efectividad, pero se necesita grandes requisitos tanto de procesamiento como de memoria. El gran problema de la técnica DPI es la dificultad que tiene para tratar con tráfico cifrado, por esto el alto coste computacional. Pero se puede se puede optimizar sin necesidad de tener una perfecta precisión realizando una serie de mejoras. El clasificador DPI se basa en las diferencias de las cabeceras de los protocolos de la capa de aplicación usadas por cada aplicación para iniciar y controlar el intercambio de datos. Se puede asociar cada aplicación con un conjunto de expresiones regulares, que representan la firma de cada protocolo de aplicación. Gracias a esto podemos clasificar los paquetes entorno a su firma. Una vez se identifica la dirección destino y fuente, el protocolo utilizado en la capa de aplicación y los puertos origen y destino, se inserta en una tabla de sesión y se clasifican según su quíntupla. A parte de esto, también existen y es posible realizar una serie de implementaciones basándose en el análisis de paquetes (PBFS)

o de mensajes (MBFS). Nos interesa evitar la normalización TCP/IP, que tiene como finalidad solucionar el problema de la fragmentación IP y el resemblado TCP, ya que deseamos utilizar un número pequeño de bytes, por ejemplo, con el primer paquete de la sesión puede ser suficiente para realizar una correcta clasificación. Por esto la elección del enfoque de utilizar PBFS. El algoritmo utilizado para analizar las expresiones regulares es el *Determinist Finite Automata* (DFA, máquina de estados finita), ya que pude reducir el coste computacional, ya que éste solo depende la longitud de la secuencia de entrada, independientemente del número de expresiones regulares analizadas. Y aunque depende de las características de las expresiones que puede requerir una gran cantidad de memoria, es la mejor elección porque no hay una gran cantidad de diferentes estados.

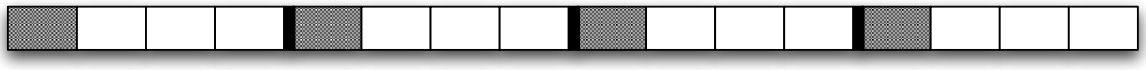
Después, otras formas de de reducir dicho coste será reducir el número de bytes analizados en la carga útil, ya que hay muchos bytes que son inútiles en la clasificación, o reducir el número de paquetes, siendo necesarios solo los primeros del flujo. Ahora se pasará a realizar una evaluación de la metodología utilizada a partir de tráfico capturado en routers de borde, donde el tráfico es limitado, con diferentes características. De dicho tráfico se conoce su contenido para analizar la precisión, el tráfico mal clasificado y como varía la velocidad en la que analiza y clasifica el trabajo. Se observa al realizar el experimento que se consiguen mayores mejoras utilizando PBFS cuando se puede tolerar algunas imprecisiones. Después se comprueba la elección del algoritmo DFA, siendo éste el más óptimo y factible en nuestro escenario. También se realiza un experimento para saber cuántos bytes son suficientes para realizar una buena clasificación, llegando a la conclusión que con 128 se obtienen grandes resultados. Por otro lado, se llega a la conclusión de que reducir el número de paquetes a analizar es una buena estrategia para mejorar el coste computacional y no perder demasiada precisión, eligiendo 10 paquetes como una buena cifra. Por último se observa si es posible utilizar conjuntamente las dos mejoras anteriores, donde no se consiguen unas conclusiones claras, ya que ambos requerimientos nos proporcionan mejores resultados por separado. Como conclusiones finales podemos decir que es posible la utilización del algoritmo DPI para la clasificación de tráfico en redes de alta velocidad. Las principales mejoras se obtienen basándose en el algoritmo PBFS, con DFA. También se llega a la conclusión que para tráfico TCP es mejor limitar el número de paquetes y para UDP limitar el número de bytes. Por tanto, con estas mejoras se obtienen grandes resultados de procesamiento sin pérdida de precisión. Este artículo puede cambiar el punto de vista sobre esta técnica en redes de alta velocidad.

2.2. Muestreo

En la actualidad, las velocidades en las redes de acceso son cada vez mayores y las nuevas aplicaciones generan mayores cantidades de tráfico, esto ha hecho que cada vez sea más complicado caracterizar o analizar las estadísticas de dicho tráfico. Por estas razones, la comunidad científica y los proveedores de servicios de Internet han realizado diversos estudios para intentar reducir la cantidad de datos a analizar y reducir el coste computacional, y gracias a diferentes técnicas de muestreo se ha conseguido resolver en cierta medida este tipo de problemas.

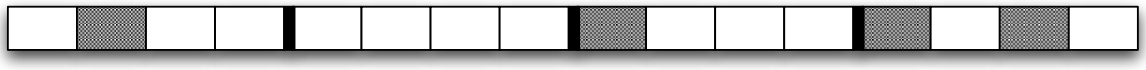
Gracias al muestreo no es necesario tratar con todo el tráfico que se quiere analizar, sino que se van cogiendo muestras cada cierto intervalo de tiempo o cada cierto número de paquetes o de flujos. El muestreo puede ser realizado durante la captura o después de la clasificación del tráfico. El principal problema del muestreo será la precisión, ya que no analizas todo el tráfico y solo se observa una parte de él. Pero, se han conseguido técnicas con las que se pierde muy poca precisión pudiendo reducir en gran medida el número de datos a analizar.

En la literatura se nos muestran diferentes técnicas de muestreo. La figura 2.2 muestra los tres principales métodos, a saber:



(a) Sistemático.

(b) Aleatorio estratificado.



(c) Aleatorio simple.

Figura 2.2: Políticas de muestreo de paquetes

- **Determinista**: se divide las muestras en ciclos de partes iguales y se toma el primer elemento de cada ciclo.
- **Aleatorio estratificado**: igual que en la técnica determinista, se divide toda la muestra en ciclos del mismo tamaño pero ahora se toma un elemento de cada ciclo de forma aleatoria.
- **Aleatorio simple**: se toman muestras de forma aleatoria.

Para realizar la división de ciclos se puede realizar mediante intervalos de tiempos, haciendo que cada cierto tiempo se pase a nueva división, o también mediante un contador de paquetes o de flujos. La selección de la muestra también se puede realizar mediante el tiempo o el evento. Otro parámetro a tener en cuenta será la frecuencia de muestreo o la granularidad, con lo que se definiría las fracciones de muestreo, pudiendo variar esta tasa dependiendo del experimento a realizar o del tipo de tráfico a analizar.

Dependiendo de la distribución del tipo de población, serán unos métodos más eficientes que otros, por ejemplo, si la varianza dentro del muestreo determinista es mayor que el de la población será más eficiente el muestreo determinista que el aleatorio simple, o a si la población tiene una tendencia lineal será más eficiente el método aleatorio estratificado que el determinista [11]. También es importante decir que se obtienen mejores resultados, aunque no en gran medida, realizando el muestreo mediante un contador de paquetes que mediante el tiempo [11] y [12].

En el artículo [13] se propone un nuevo método de muestreo adaptativo que depende de la tasa de tráfico. Con ello se intenta ser más preciso en el muestreo adaptándolo al tipo de tráfico o los parámetros de tráfico que se desean obtener, todo a partir de analizar el tráfico con anterioridad. Este mecanismo se puede dividir en tres partes: en una primera parte, estaría el estimador de la tasa de la ventana de tiempo deslizante, donde se calcula la tasa de tráfico y donde también, se permite suavizar el efecto del ruido. Después tenemos el algoritmo de estimación del tamaño de muestreo, y por último los paquetes integrados de monitorización que son insertados cada cierto tiempo, pudiendo conocer el número de paquetes recibidos. Todo ello para conseguir reducir la tasa de muestreo, y por tanto analizar un número menor de datos con gran precisión. Además se muestra que para analizar tráfico de voz funciona mejor que con los tres métodos anteriores.

También existe otras dos nuevas técnicas de muestreo que son utilizadas para detectar y clasificar anomalías en la red, dicha información se expone en el artículo [14]. Se explica que se ha demostrado que el muestreo sobre flujos es más idóneo y más preciso para analizar estadísticas de flujo. Por ello, se utiliza este tipo de método en las técnicas de muestreo oportunista, que

se nos explica, para la detección de anomalías. El primer método, muestreo selectivo, se utiliza para flujos con un número bajo de paquetes, y el método de muestreo inteligente para flujos de mayor tamaño. Aprovechan el hecho de que grandes cantidades de información se encuentran en fracciones pequeñas del flujo, por tanto se puede reducir bastante el tamaño de tráfico a analizar, y de que la mayoría de los flujos pequeños son el origen de ataques. Dependiendo de si se utiliza un método u otro se utiliza diferentes fórmulas para calcular la probabilidad de realizar el muestreo del tráfico, con el fin de conseguir una alta precisión. Todo en función del tamaño del flujo. Se demuestra, gracias a diversos experimentos, como eligiendo bien el método de muestreo a emplear se consigue una gran efectividad para detectar anomalías e incluso de revelar ataques que de otra forma no se podrían encontrar.

Otros investigadores [15] han estudiado el impacto que se obtendría en la clasificación de tráfico mediante los datos que se desprenden con la herramienta *Sampled Netflow*, ya que es un sistema de monitorización de tráfico muy usada actualmente por operadores de red. Se ha trabajado con *Netflow* debido a que se reduce el coste computacional y la complejidad del hardware necesario en comparación con técnicas de aprendizaje automático. Además se evita el tiempo dedicado a fase de entrenamiento necesario en este tipo de técnicas. Netflow es un protocolo de Cisco que recoge información acerca de los flujos IP en routers y switches. Esta técnica limita la información acerca de ciertas características que luego son usadas por métodos de clasificación de aprendizaje automático. Además para reducir más dichos costes se desarrolló Sampled Netflow que se encarga de realizar un muestreo de la información recolectada. Por todo esto, se ha querido estudiar cómo afecta a la clasificación de tráfico el uso de este sistema. En este artículo se ha evaluado como afecta a la precisión de una técnica de aprendizaje automático, en este se ha utilizado el árbol de decisión C4.5, que utiliza para clasificar tráfico la información extraída de Sampled Netflow (número de puerto origen y destino, protocolo IP, type of service, etc), más algunas características que se han añadido como la media del tamaño de los paquetes y el interarrival. Primeramente, se ha mostrado la precisión de éste método sin muestreo, donde se obtienen buenos resultados. Y después se realizado los mismos experimentos con muestreo, observando como a medida que se aumenta la tasa de muestreo la precisión cae rápidamente, todo esto debido al error introducido en la estimación de las características del flujo, a cambios en la distribución del tamaño del flujo y a la partición de los flujos durante el muestreo. Debido a la caída de la precisión en el paper se propone una mejora para poder solucionar los errores que aparecen por el muestreo. Esta mejora consiste en utilizar las trazas muestreadas en el proceso de entrenamiento, es decir, utilizar la misma tasa de muestreo en esta fase y en el proceso de clasificación. Este cambio consigue que la precisión se ha bastante buena incluso con la utilización de Sampled Netflow, es decir, con el muestreo de paquetes. Se espera que con estos resultados los operadores empiecen a utilizar esta técnica en vez técnicas clasificación DPI o basada en los puertos conocidos.

En el paper [16] se analiza cómo afecta a la caracterización y a la clasificación de tráfico el muestreo de paquetes, por una parte midiendo el efecto que realizan diferentes políticas de muestreo y diferentes tasas, y por otra parte la pérdida de precisión de un detector, *tstat*, después del muestreo de los datos. Las técnicas de muestreo son las tres contadas anteriormente (sistemático, aleatorio estratificado y aleatorio simple), más una nueva no contada hasta ahora, *systematic SYN*, por un lado toma paquetes como un muestreo determinista y por otra parte toma todos los paquete TCP con la bandera SYN activa. Para analizar la distorsión entre los flujos muestreados y sin muestrear se toman dos métricas con diferentes características, *Fleiss Chi-Square* y *Hellinger distance*. Después se realiza un estudio viendo como varía estas dos métricas y ciertas características del tráfico en función de la política de muestreo y la tasa tomada. Se observa en dicho estudio como la mayoría de las características se distorsionan incluso con tasas bajas de muestreo independientemente de la técnica utilizada. Otras, en cambio, son robustas al muestreo. Se observar también que no hay ventajas de usar una política de muestreo

u otra. La clasificación del tráfico se realiza mediante una técnica de aprendizaje automático, *arboles de decisión C4.5*. La clasificación muestra que para bajas tasas de muestreo algunas características se distorsionan poco haciendo que la precisión del detector no se degrade en demasía. Por tanto eligiendo cuidadosamente la tasa de muestreo y las características a utilizar para la clasificación del tráfico se pueden obtener grandes resultados de precisión realizando un muestreo de la información, es decir, reduciendo el número de operaciones a realizar y el número de paquetes y bytes a analizar.

En los artículos anteriores se ha analizado el impacto del muestreo de paquetes sobre la clasificación de tráfico, pero en cambio en el siguiente estudio se ha querido evaluar el muestreo de flujos [17], mediante técnicas acerca del comportamiento del tráfico que requieren un bajo coste computacional. Normalmente la clasificación del tráfico se realiza cerca de la red de acceso del usuario, en este trabajo se ha querido observar el comportamiento si se realiza en el interior de la red donde solo existe una parte de todo el tráfico intercambiado entre usuarios. En este caso el encargado de clasificar el tráfico es el detector *Abacus*, que mediante patrones y firmas del tráfico intercambiado entre usuarios es capaz de determinar la aplicación P2P utilizada. El detector es entrenado con tráfico sin muestrear. Se realizan dos técnicas diferentes para el muestreo de flujos, una más realista, mediante el uso de las tablas de reenvío de router reales, y otra más idealizada, con un muestreo aleatorio dependiendo de la red de origen. En el estudio se observa como cuando la tasa de muestreo es baja, hasta 1/8, las firmas de las aplicaciones P2P no sufren grandes cambios y a medida que se aumenta la tasa de muestreo las diferencias entre la firma muestreada y sin muestrear van siendo mayores. Por último, se evalúa como cambia la precisión en relación con la tasa de muestreo, obteniendo una caída de precisión de alrededor del 20 % cuando solo el 10 % de la red es muestreada.

2.3. Pérdidas

En las aplicaciones en tiempo-real sabemos que muchas veces se producen pérdidas y retrasos en la llegada de los paquetes, afectando a la Quality of Experience (QoE), es decir, las aplicaciones tendrán una mala calidad si se producen muchos errores. Sobre todo, este tipo de problemas ocurren en aplicaciones de VoIP, haciendo que se degrade la interactividad, no pudiendo tener una conversación coherente. Cuando se produce una pérdida o un retraso hay una gran probabilidad de que el siguiente paquete también falle, esto es debido a que existe una dependencia temporal. Se producen ráfagas de errores haciendo que con el *Forward Error Correction*(FEC) no pueda corregir los fallos correctamente.

Para caracterizar este tipo de fallos en el paper [18] se nos habla de utilizar la cadena de Markov, y más concretamente, el modelo de Gilbert o segundo estado del modelo de Markov. La cadena de Markov tiene tantos estados como eventos diferentes, donde la probabilidad de pasar a un nuevo estado depende del evento inmediatamente anterior. El modelo de Gilbert, mostrado en la figura 2.3, se basa en el segundo estado del modelo de Markov, es decir, tenemos dos estados, pérdidas y no pérdidas, donde hay una probabilidad p de que si estamos en el estado de no pérdidas pasemos al estado de pérdidas y una probabilidad q de que estando en pérdidas se pase a no pérdidas. También tenemos una probabilidad de $1-p$ y $1-q$ de seguir en el mismo estado. Si $p+q=1$ se reduce en el modelo de Bernoulli.

La distribución de probabilidad de pérdidas será: $P_k = (1-q)^{k-1} \times q$, siendo k la longitud de pérdidas. También se nos expone el modelo de cuatro estados de Markov, en el que tendremos dos estados buenos y dos malos para generar la distribución de la duración de las fases buenas y malas con transiciones especificas. Se puede considerar como una extensión del modelo de Gilbert con diferentes duraciones de estados. Con este modelo se caracteriza mejor algunos tipos de comunicaciones.

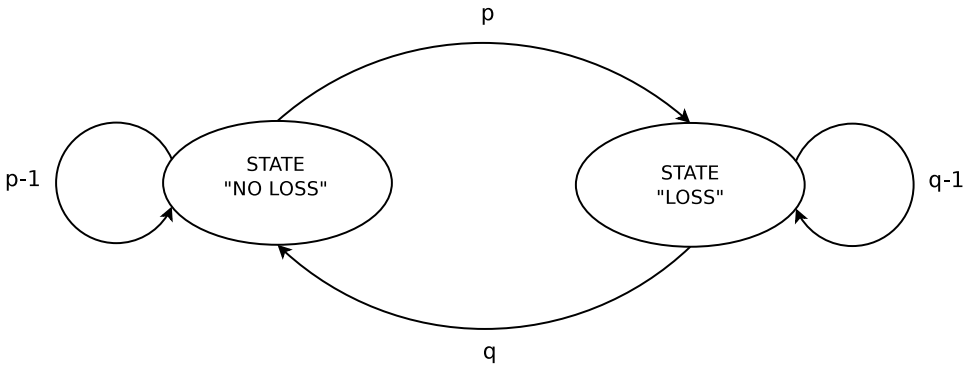


Figura 2.3: Modelo de Gilbert

En el artículo [19] se nos habla de utilizar el modelo extendido de Gilbert junto con la medida de distancia entre pérdidas. En el modelo extendido de Gilbert se nos habla de n+1 estados para recordar n eventos. En la cadena de Markov se asume que todos los eventos anteriores afectan al futuro, pero en el modelo extendido no pasa esto, solo los n eventos consecutivos de pérdidas afectan al futuro. Ahora tendremos probabilidades diferentes dependiendo del estado en el que estamos y del número consecutivo de pérdidas, es decir, no hay la misma probabilidad de estar en la quinta pérdida consecutiva y pasar a la sexta que si estamos en la primera. El problema es que este modelo tampoco captura la rafagosidad y el agrupamiento de las pérdidas. Para solucionar este problema nos ayudamos de la medida de la distancia entre pérdidas. Con ello se describe la distancia entre paquetes perdidos como una secuencia de números. Con ello se conseguirá mejorar la modelización de las pérdidas en aplicaciones en tiempo real en las diferentes redes.

3

Herramientas de detección

3.1. Tstat

Clasificador de tráfico desarrollado a partir del algoritmo explicado en el artículo [7], desarrollado por el Grupo de Redes de Telecomunicación de la Universidad Politécnica de Torino. Esta herramienta [20] es capaz de detectar diferentes aplicaciones de un conjunto de flujos en una red, generando una serie de medidas y estadísticas referidas a los paquetes clasificados. Puede clasificar tráfico tanto en tiempo real, como trazas que han sido capturadas previamente.

3.1.1. Instalación

La herramienta ha de instalarse en un sistema operativo Linux, con núcleos de una versión mayor de la 2.2. Se necesitará tener una serie de librerías instaladas, entre las que destacamos:

- Libpcap: para poder capturar y procesar datos en una red LAN.
- Libpthread: para poder trabajar con hilos. Esto mejorará el rendimiento del detector cuando se realice en entornos multi-corel.
- Librrd: para poder utilizar la herramienta round robin database.
- Libz: para compresión de librerías.

Descargamos la aplicación de la página [21]. La última versión de tstat es la 2.2. Abrimos la terminal y ejecutamos una serie de comandos para instalarla:

1. ***./configure [–enable-libtstat]***
2. ***make***
3. ***sudo make install***

Una vez hemos realizado todos estos pasos, si no se ha informado de ningún error por la terminal, ya podríamos empezar a utilizar el detector.

3.1.2. Funcionamiento

Tstat captura y caracteriza el tráfico de la red y genera una serie de medidas de gran utilidad. La herramienta puede analizar los paquetes que se generan en una red, a través de una interfaz en tiempo real, o también puede analizar trazas obtenidas con anterioridad, en un archivo comprimido o descomprimido. Los archivos que genera son los siguientes:

- *Archivos logs*: se crean archivos de texto con diferentes medidas acerca de la traza detectada a nivel de flujos.
- *Histogramas*: diferentes archivos con la distribución de una cantidad dada, durante un intervalo de tiempo.

Los archivos de texto que se generan, *logs*, son los siguientes:

- log_tcp_complete, log_tcp_nocomplete: estadísticas de los flujos TCP.
- log_udp_complete: estadísticas de los flujos UDP.
- log_mm_complete: estadísticas de los flujos RTP y RTCP.
- log_chat_complete, log chat messages: estadísticas de los flujos y mensajes generados por aplicaciones chat.
- log_skype_complete: estadísticas de los flujos generados por la aplicación Skype.
- log_video_complete: estadísticas de los flujos TCP de aplicaciones de video.

Los que más nos interesan serán los archivos de UDP y de Skype. En el archivo UDP se muestra información como la dirección IP origen y destino, los puertos utilizados, el tiempo entre el primer y el último paquete del flujo. Y en el log de Skype se presenta el número de paquetes E2E, E2O o de señalización, el número de paquetes de audio o de vídeo, si la conexión es TCP o UDP, etc.

A cerca de los histogramas, se genera una serie de subdirectorios con medidas en diferentes intervalos de tiempos (por defecto 5 minutos), donde aparece un valor máximo, mínimo y el tamaño del contenido. Será de gran utilidad para crear gráficos. En cada subdirectorio se generan ocho archivos con dichas medidas:

- Ip_bitrate_loc.
- Ip_len_loc.
- L7_UDP_num_in.
- L7_UDP_num_loc.
- L7_UDP_num_out.
- Udp_bitrate_in.
- Udp_bitrate_loc.
- Udp_bitrate_out.

Los archivos in son de comunicaciones de un host externo a uno interno, los out es al contrario y los loc entre dos hosts internos. Existe la posibilidad de utilizar la opción ***-H ?***, para sacar una serie de valores de histogramas por pantalla de diferentes campos.

3.1.3. Ejecución

Para poder utilizar esta herramienta debemos ejecutarla con privilegios de superusuario:

sudo tstat [opciones]

Las opciones que debemos utilizar y cómo utilizarlas podemos verlas ejecutando la instrucción siguiente en la terminal:

tstat -h

Entre las más interesantes, serán las opciones necesarias para detectar tráfico y ver información de las tramas acerca de un flujo ya clasificado y que conocemos. Para ello debemos ejecutar el siguiente comando:

tstat -N [archivo con direcciones IP de host de la red que queremos analizar] -H [archivo de configuración de histogramas] [archivo con la traza a analizar]

En el caso que la traza sea Skype debemos añadir la opción -B junto con el directorio donde se encuentra el archivo de configuración para la clasificación bayesiana. Como hemos dicho antes, también tenemos la opción de detectar y clasificar el tráfico que circula por una determinada red privada y generar sus estadísticas, para ello debemos escribir esta instrucción:

tstat -N [archivo con direcciones IP de host de la red que queremos analizar] -H [archivo de configuración de histogramas] -li [nombre de la interfaz donde se realizará la captura de tráfico] -B [directorio donde se encuentra el archivo de configuración para la clasificación bayesiana]

Después de haber utilizado la herramienta tstat, ésta generara una serie de archivos y documentos de gran interés. Se genera una carpeta llamada stdin o en caso de utilizar la opción -s podemos cambiar el nombre del directorio a crear, dentro de esta carpeta se crea otro directorio con el nombre de la fecha y la hora del primer flujo detectado, y en ella varios archivos logs e histogramas mencionados anteriormente.

3.2. L7filter

Esta herramienta es utilizada para detectar y clasificar paquetes basado en la capa de aplicación en tiempo real, a partir de patrones y diferentes firmas [22]. L7-fliter debe instalarse en un sistema Linux y es capaz de determinar el protocolo que se está utilizando. Existe una versión para el kernel y otra para espacio de usuario.

Se apoya en *Netfilter* [23], framework capaz de manipular paquetes de red en el kernel de Linux, en diferentes etapas de procesamiento. Los paquetes pueden ser aceptados o rechazados, y una vez aceptados pasan a introducirse en una cola para ser usada en el espacio de usuario. Otro elemento importante es la herramienta de espacio de usuario IPTABLES, que permite el filtrado de los paquetes que circulan por la red, es decir, permite definir reglas para decir que hacer con los paquetes.

3.2.1. Instalación

La herramienta descargada en nuestro caso es el l7-filter versión 0.12 beta para espacio de usuario. La página de la cual nos descargamos el detector es la siguiente [24]. Antes de realizar la instalación debemos tener en nuestro sistema una serie de archivos de netfilter:

- Libnetfilter conntrack.
- Libnetfilter queue.
- Protocolos soportados por l7-filter.

No debemos descargarnos las últimas versiones de estas librerías porque da errores con nuestra versión del l7-filter. Una vez realizado estos pasos previos pasamos a la instalación de la herramienta. Primero debemos descomprimir el archivo y después escribir en la terminal los siguientes comandos:

1. ***./configure***
2. ***make***
3. ***make install (como super-usuario)***

Una vez hemos realizado todos estos pasos, si no se ha informado de ningún error por la terminal, ya podríamos empezar a utilizar el detector.

3.2.2. Funcionamiento

El l7-filter analiza todo el tráfico que se le envía desde la cola de paquetes creada con la herramienta iptables. Con esta herramienta realizaremos el filtrado para ver solo los flujos que nos interesa analizar. También debemos crear un archivo de configuración para el l7-filter, donde especificaremos los protocolos que queremos que detecte la herramienta seguido de un número, a partir del 3, para referenciar cada protocolo o aplicación con una etiqueta.

3.2.3. Ejecución

Para poder ejecutar la herramienta debemos hacerlo como superusuario y debemos cargar el módulo ip_conntrack_netlink mediante el comando:

sudo modprobe ip_conntrack_netlink

También debemos cargar:

sudo modprobe nf_conntrack_ipv4

Después debemos re-enviar el tráfico de red mediante el comando iptables:

sudo iptables -A FORWARD -j NFQUEUE –queue-num 0

Con esto estamos creando una nueva regla, para ver todos los paquetes que pasan por el sistema e introducirlos en la cola 0 para después ser utilizado por l7-filter en el espacio de usuario. Ahora ya estamos preparados para analizar el tráfico de nuestra red. Para ello ejecutamos el siguiente comando en la terminal:

l7-filter -f [nombre del archivo de configuración del l7-filter] -q [número de la cola donde hemos enviado los paquetes al espacio de usuario]

Se puede añadir otras opciones al comando l7-filter, para verlas podemos ejecutar el comando:

man l7-filter

Cabe destacar la opción -vv y -vvv para poder ver los paquetes que se están analizando y estadísticas sobre ellos, como el número de IP origen y destino, el número de puerto o el protocolo utilizado. El problema de esta herramienta es que no puede analizar trazas que han sido capturadas previamente.

3.3. Skypeness

Herramienta desarrollada en la Universidad Autónoma de Madrid por el grupo High Performance Computing and Networking y explicado en detalle en el artículo [5]. Se basa en detectar tráfico Skype a partir de características estadísticas de los flujos, como el tamaño de los paquetes, el tiempo entre llegadas y el bitrate.

3.3.1. Instalación

Para instalar esta herramienta debemos descargarnos los scripts donde se desarrolla la herramienta, con su correspondiente Makefile que nos permitirá construir fácilmente el programa. Por tanto, lo único que debemos hacer es, mediante la consola, introducirnos en la carpeta donde están los archivos y realizar un make. Con esto se generará el programa Skypeness para su ejecución.

3.3.2. Funcionamiento

El detector toma una traza con su correspondientes flujos para analizar y ver si dicho flujo es o no Skype. Para ello, como hemos dicho se basa en ver si ciertas características están en unos ciertos intervalos. Estos datos no se comparan a nivel de paquete, si no que se toman en ventanas de 10 paquetes para suavizar los datos.

Esta herramienta solo tiene en cuenta tráfico UDP, ya que el tráfico TCP suele ser escaso como hemos visto anteriormente, y flujos con más de 30 paquetes para no tener en cuenta el tráfico de señalización Skype dedicado a tareas de control de la aplicación, y solo centrarnos en flujos correspondientes a conversaciones, videoconferencia o transferencia de archivos. Si los estadísticos del flujo está entre los intervalos dados en la tabla 3.1 y supera un cierto umbral, el flujo se clasifica como Skype.

Una vez aplicado el detector a una traza se genera tres archivos con diferente información:

- flows.dat. Información a nivel de flujo (IP, puerto, paquetes, bytes, Skype o no,...)
- log.dat. Información a nivel de paquete (número de paquetes TCP, UDP, ICMP,...)
- stats.dat. Información general de la traza.

Tipo de servicio	Característica	Intervalo	Umbral
Audio	Tamaño del paquete [Bytes]	[30, 200]	0.75
	Interarrival [ms]	$[i_{n-1} \pm 15]$	0.6
	Bitrate [Kbps]	[0, 150]	0.75
Video	Tamaño del paquete [Bytes]	[150, 1200]	0.19
	Interarrival [ms]	$[i_{n-1} \pm 15]$	0.6
Transferencia de archivo	Tamaño del paquete [Bytes]	[480, 540] [950, 1050] [1310, 1380]	0.44

Tabla 3.1: Intervalos y umbrales de Skypeness

3.3.3. Ejecución

Para ejecutar el programa Skypness, primero debemos situarnos mediante la terminal en la carpeta donde se encuentra el ejecutable creado con el Makefile, y después ejecutar el siguiente comando:

sudo ./skypeness [ruta donde se encuentra la traza] [nombre de la traza a analizar] f 15 [ruta donde quieres que se genere los archivos de información estadística]

4

Análisis del impacto del muestreo y la pérdida de paquetes en la detección de tráfico Skype

En esta parte del proyecto se han realizado diversos experimentos para evaluar el impacto del muestreo y de la pérdida de paquetes en la precisión del detector Skypeness.

4.1. Metodología

Para realizar el muestreo hemos creado un algoritmo donde se elige:

- el tipo de muestreo que queremos aplicar a nuestra traza: sistemático, aleatorio estratificado y aleatorio simple;
- la tasa de muestreo: 2, 4, 8, 16, 32, 64, 128, 256, 512 y 1024.

Y para simular las pérdidas hemos creado otro algoritmo donde se escoge:

- la probabilidad de estar en el estado de no pérdidas: 0.99, 0.95, 0.9, 0.5 y 0.1;
- la probabilidad de pasar del estado de pérdidas a no pérdidas: 0.5 y 0.6.

Para realizar dichos algoritmos hemos utilizado la librería *libpcap*, que es capaz de descartar o almacenar en una nueva traza los paquetes que se desee. Esta herramienta se ha hecho pública para la utilización de cualquier persona que lo desee, pudiéndose descargar desde [25]. En el apéndice C se encuentra el código en C de dichos algoritmos.

La finalidad de este estudio es observar como varía la precisión del detector cuando variamos la política de muestreo, la tasa de muestreo o la probabilidad de perder paquetes. Para el tráfico Skype y no Skype mostramos el resultado en gráficas donde pintamos la precisión del detector, a nivel de flujo, paquete y bytes, frente a la tasa de muestreo para las tres técnicas en escala logarítmica; y para las pérdidas representamos la precisión frente a la probabilidad de estar en no pérdidas.

La precisión se mide como:

$$\text{Precisión} = \frac{TP+TN}{TP+FP+TN+FN}$$

Para realizar dichas pruebas utilizamos diversas trazas de tráfico capturadas previamente, que solo contengan tráfico Skype o que no contengan nada de Skype. De dicho tráfico solo tenemos en cuenta el tráfico UDP y flujos que contengan más de 30 paquetes para no tener en cuenta el tráfico de control.

En la tabla 4.1 mostramos las características de las seis trazas que utilizamos en nuestros experimentos. Las trazas 4 y 5 fueron capturadas en la Universidad Politécnica de Torino [26] y las demás en la Universidad Autónoma de Madrid. Las trazas de la UAM se han hecho públicas y anonimizadas, por tanto, se pueden descargar para ser usadas por la comunidad científica desde [27].

Traza	Tipo de llamada		**Skype**	**No Skype**	**Skype Media**
Traza 1	E2E	Bytes	30950000	0	Audio
		Paquetes	230100	0	
		Flujos	44	0	
Traza 2	E2E	Bytes	108700000	0	Video
		Paquetes	217300	0	
		Flujos	46	0	
Traza 3	E2E	Bytes	162800000	0	Envío archivo
		Paquetes	254300	0	
		Flujos	46	0	
Traza 4	E2E	Bytes	8381658970	0	Audio y video
		Paquetes	39458562	0	
		Flujos	1059	0	
Traza 5	E2O	Bytes	231257652	0	Audio
		Paquetes	3049148	0	
		Flujos	1059	0	
Traza 6	–	Bytes	0	1098935	–
		Paquetes	0	5312	
		Flujos	0	52	

Tabla 4.1: Trazas utilizadas

La traza que no contiene tráfico Skype está compuesta por tráfico de diversas aplicaciones P2P como BitTorrent o Emule.

Antes de pasar a analizar el impacto del muestreo en la precisión del detector, hemos llevado a cabo un profundo análisis estadístico de las distintas trazas. Posteriormente, mostraremos los resultados de cada traza e intentaremos explicar cómo varía la precisión del detector y el por qué actúa así.

4.2. Análisis estadístico

Para entender cómo funciona el detector a medida que se pierden paquetes hemos querido analizar como varían dos de las características estadísticas por las que se rige Skypeness: tamaño del paquete y el tiempo entre llegadas de los paquetes, respecto a la tasa de muestreo o de pérdidas. El bitrate no lo analizaremos puesto que está directamente relacionado con estos dos parámetros.

Debido a que Skypeness utiliza ventanas de 10 paquetes para comprobar que los estadísticos están dentro de ciertos intervalos (mostrados en la tabla: 3.1), aquí se muestra los resultados también en ventanas del mismo tamaño.

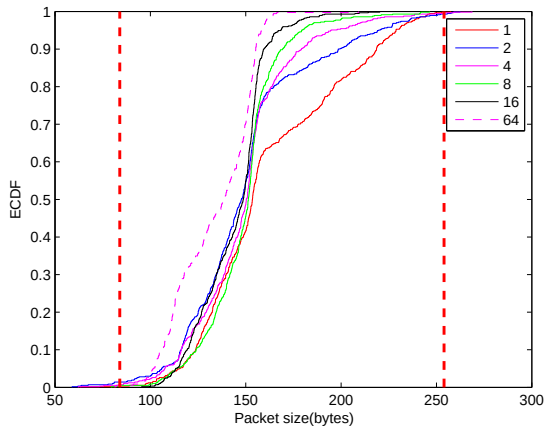
Los resultados se muestran en gráficas representando la función de distribución empírica acumulada (ECDF) para las diversas trazas y diferentes algoritmos (muestreo y pérdidas) con sus diferentes parámetros (tasa de muestreo y la probabilidad de estar en el estado de no pérdidas y de pasar del estado de pérdidas a no pérdidas). Hemos escogido los casos más representativos dependiendo del tipo de traza. Las rayas verticales punteadas que aparecen en las gráficas son los intervalos que analiza Skypeness para decidir si el flujo es Skype o no teniendo el cuenta el tipo de tráfico (audio, vídeo o transferencia de archivo). En las leyendas de las gráficas para el muestreo se muestra el inverso de la tasa de muestreo y para las pérdidas se muestra la probabilidad de estar en el estado de no pérdidas y la probabilidad de pasar del estado de pérdidas a no pérdidas.

Para calcular el tamaño de los paquetes de cada flujo se ha tenido en cuenta las cabeceras, es decir, los 54 bytes que suman a los bytes del payload.

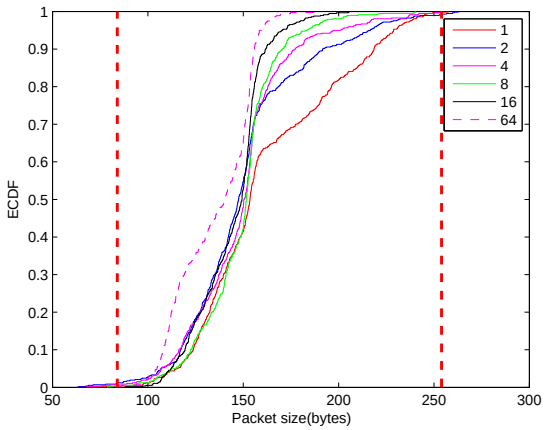
En el caso de la ECDF del interarrival debe estar comprendida entre 0 ms y 15 ms para ser considerado tráfico Skype, tanto para audio como para vídeo. El interarrival de la traza 3 no se ha puesto, ya que la transferencia de archivo solo se decide observando el tamaño del paquete.

4.2.1. Tamaño del paquete

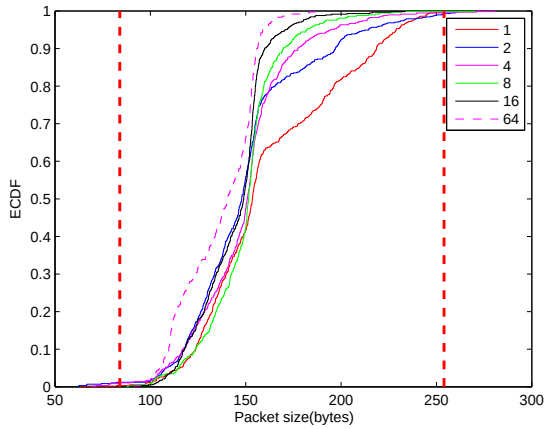
Como podemos apreciar en la figura 4.1, la variación de la ECDF del tamaño del paquete es muy pequeña, independientemente del tipo de muestreo y de las pérdidas introducidas, y se encuentra en todo momento en los rangos donde se detecta como tráfico Skype, ya sea audio o vídeo. Esto mismo ocurre en las demás figuras: 4.2, 4.3, 4.4, 4.5 y 4.6.



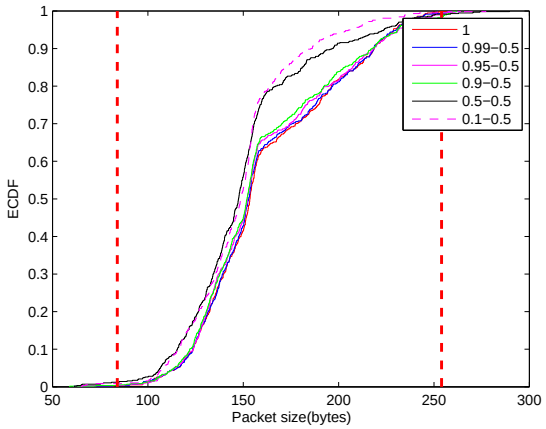
Muestreo sistemático



Muestreo aleatorio estratificado

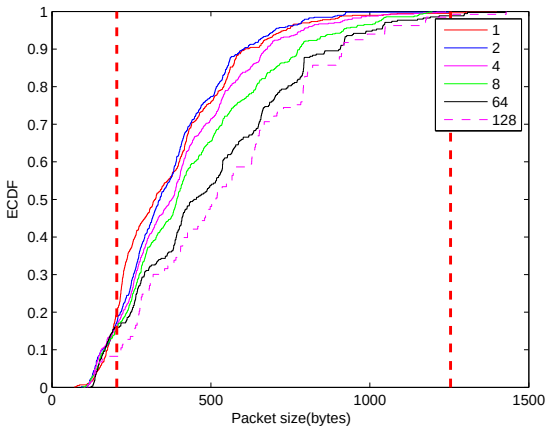


Muestreo aleatorio simple

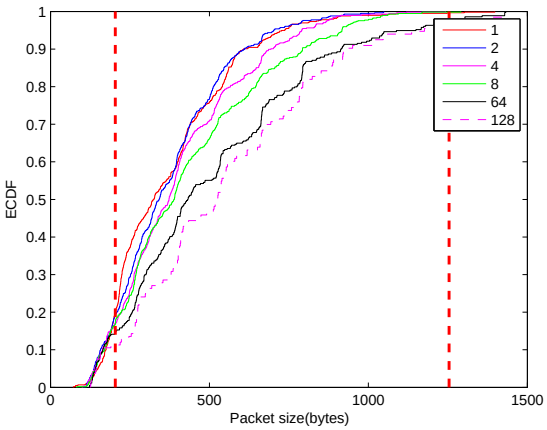


Pérdidas

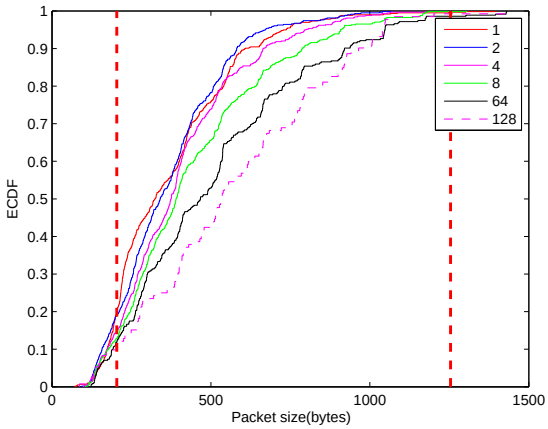
Figura 4.1: ECDF Tamaño del paquete: Traza 1



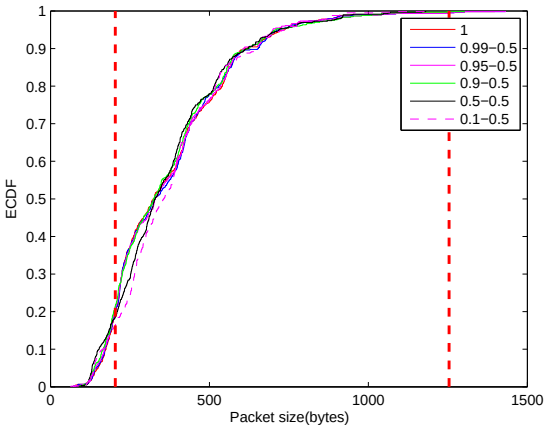
Muestreo sistemático



Muestreo aleatorio estratificado



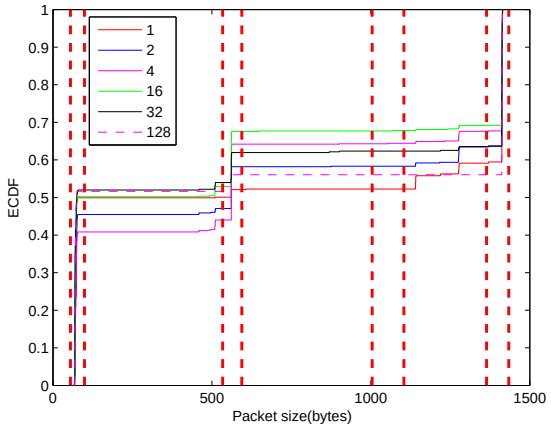
Muestreo aleatorio simple



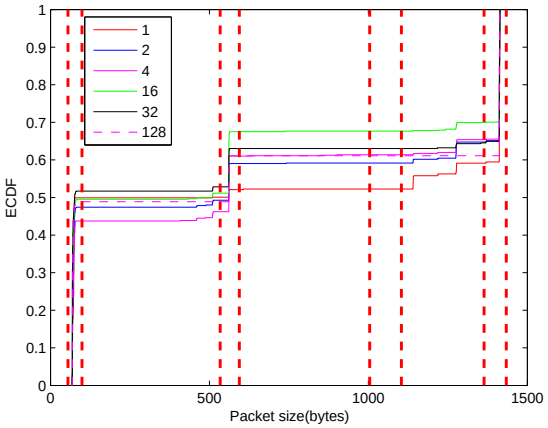
Pérdidas

Figura 4.2: ECDF Tamaño del paquete: Traza 2

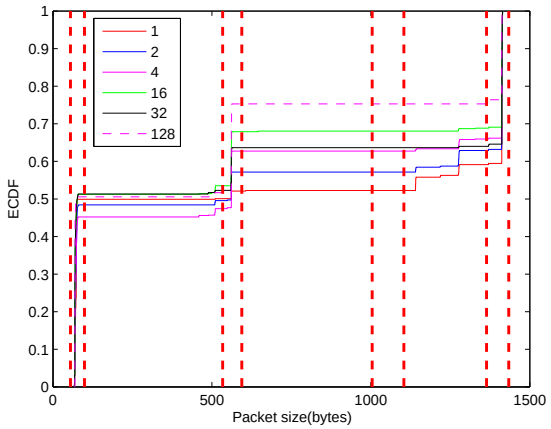
En el caso de la figura 4.3 vemos que hay cuatro posibles rangos de decisión. El primero de ellos corresponde al envío de asentimientos (ACK) de confirmación y los demás a diferentes tipos de envíos de archivos.



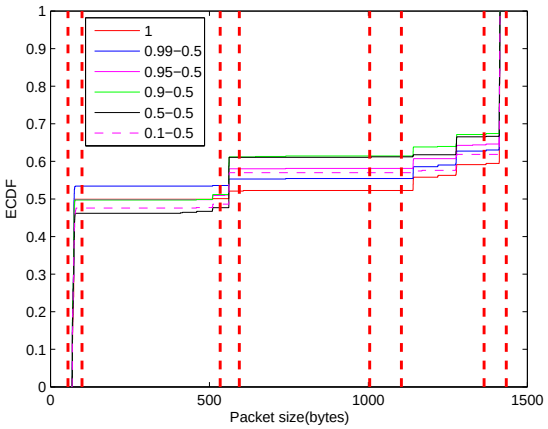
Muestreo sistemático



Muestreo aleatorio estratificado



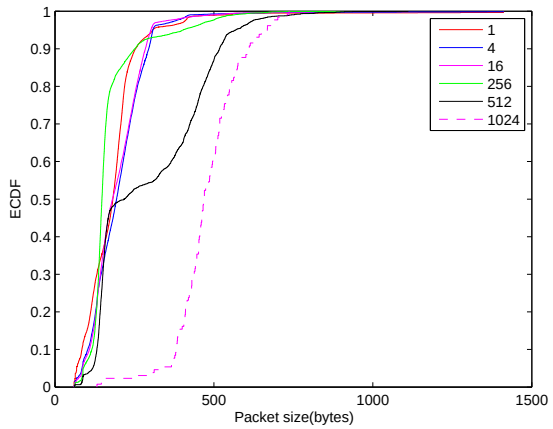
Muestreo aleatorio simple



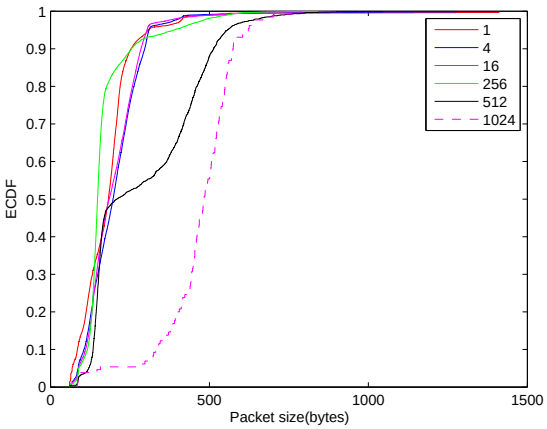
Pérdidas

Figura 4.3: ECDF Tamaño del paquete: Traza 3

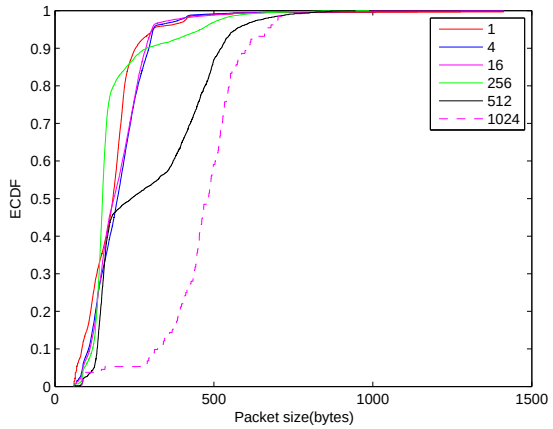
Para la traza 4 (llamada E2E) no hemos pintado las líneas punteadas como vemos en las gráficas de la figura 4.4 puesto que esta traza está compuesta tanto de audio como de vídeo. Apreciamos como todos los paquetes están en el rango de audio (84-254) o de vídeo (204-1254).



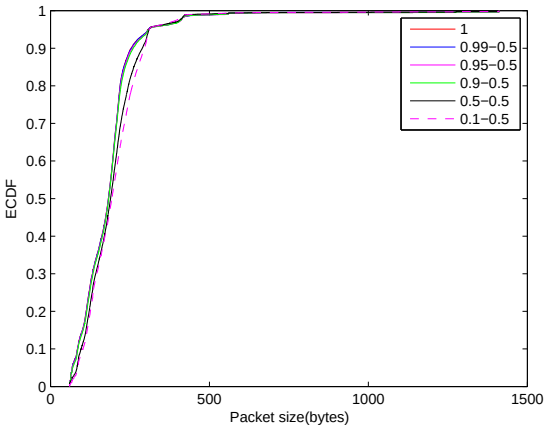
Muestreo sistemático



Muestreo aleatorio estratificado



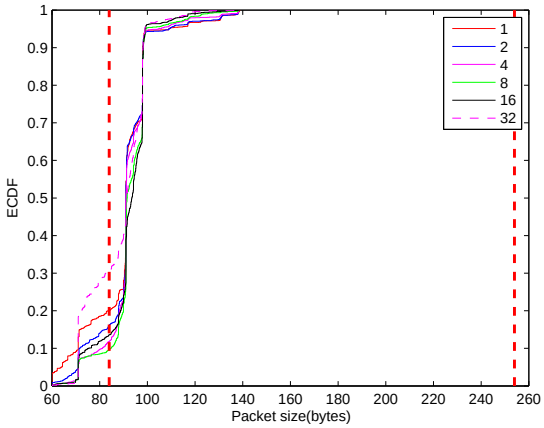
Muestreo aleatorio simple



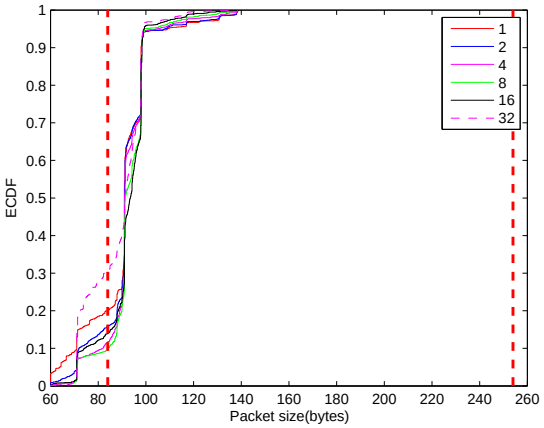
Pérdidas

Figura 4.4: ECDF Tamaño del paquete: Traza 4

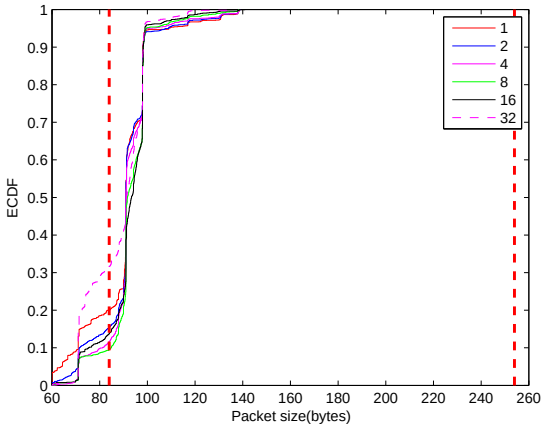
En esta traza se han capturados llamadas E2O, es decir, llamadas entre un usuario Skype y un terminal de telefonía convencional, por tanto, todo el tráfico será audio, como se observa en la figura 4.5, puesto que desde un teléfono convencional no se puede realizar videollamadas o envíos de archivos.



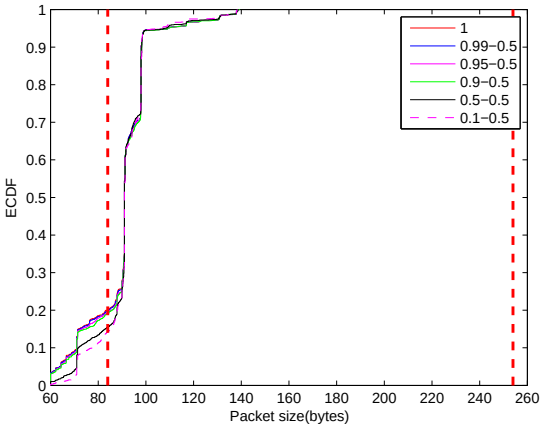
Muestreo sistemático



Muestreo aleatorio estratificado

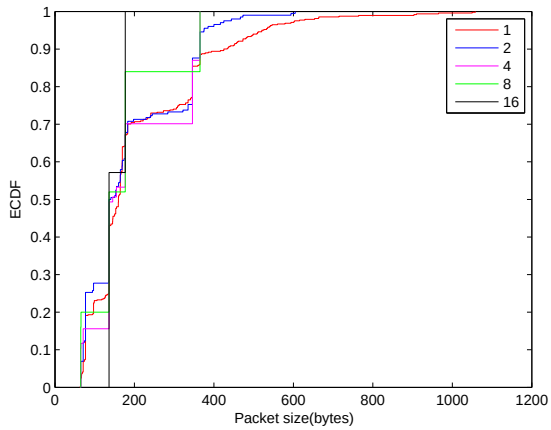


Muestreo aleatorio simple

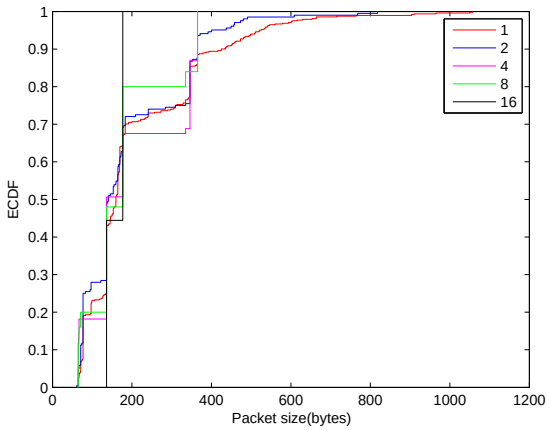


Pérdidas

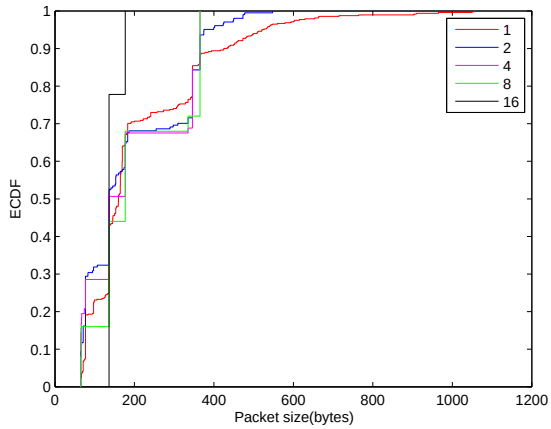
Figura 4.5: ECDF Tamaño del paquete: Traza 5



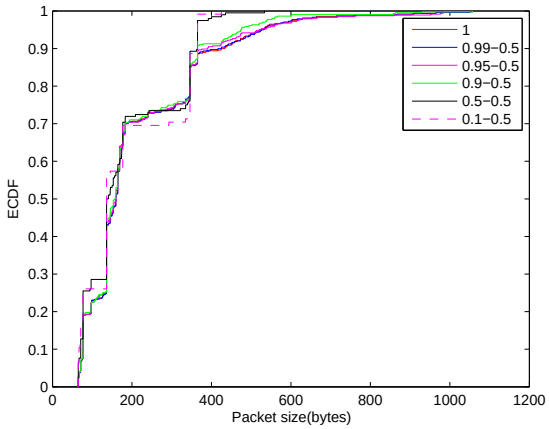
Muestreo sistemático



Muestreo aleatorio estratificado



Muestreo aleatorio simple



Pérdidas

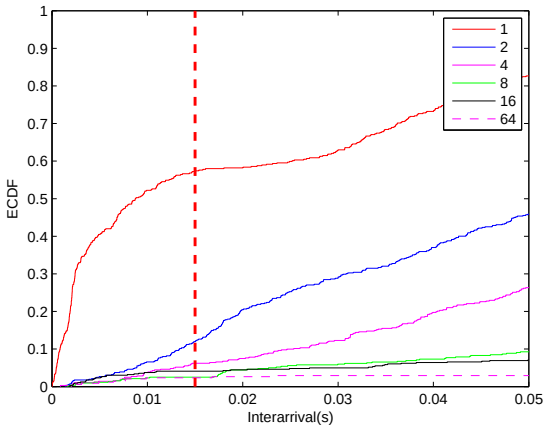
Figura 4.6: ECDF Tamaño del paquete: Traza 6

Una vez realizado el análisis estadístico se puede llegar a la conclusión de que si se realiza el algoritmo de muestreo o de pérdidas, independientemente de la tasa de muestreo que se aplique, del tipo de muestreo o de la probabilidad de pérdidas tomada, no afecta al tamaño del paquete en un factor relevante y, por tanto, no influirá en demasía a la precisión del detector, como veremos más adelante. Esto es algo que se podía haber intuido, puesto que aunque se tomen menos paquetes la media de sus tamaños tenderá al mismo número, aunque para tasas de muestreo altas se empieza a observar mayores diferencias en la ECDF, como por ejemplo, en el caso de la figura 4.4 para una tasa de 1/1024, al contar con un número mucho menor de paquetes.

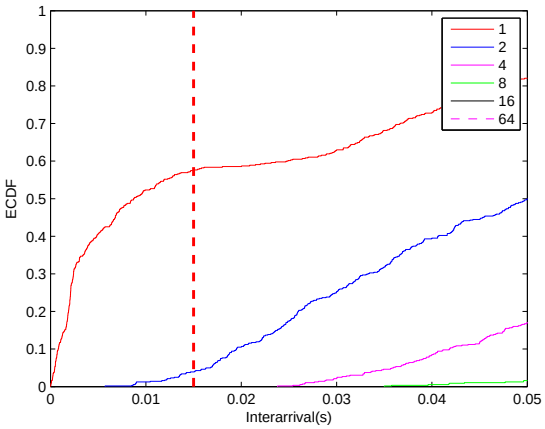
También observamos como el tamaño del paquete se mueve entre los intervalos y umbrales en los que el tráfico se considera Skype, tanto para audio, vídeo y transferencia de archivos, esto también ocurre con el tráfico No-Skype, figura 4.6.

4.2.2. Interarrival

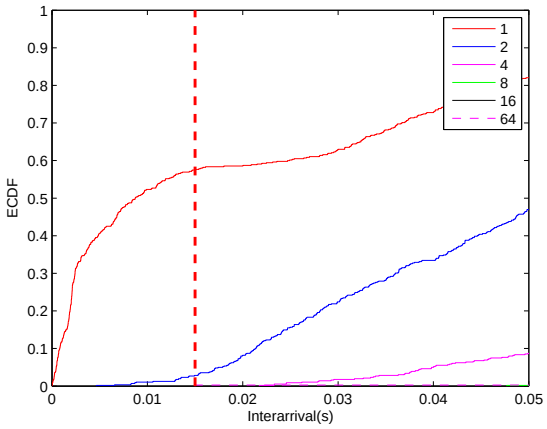
Viendo la figura 4.7 nos damos cuenta rápidamente como ahora si se observan mayores cambios en la distribución del interarrival, este hecho se explicara en mayor profundidad más adelante. Este suceso se repite para todas las trazas, como se puede observar en las figuras: 4.7, 4.8, 4.9, 4.10 y 4.11; salvo algunas excepciones que comentaremos en las siguientes páginas.



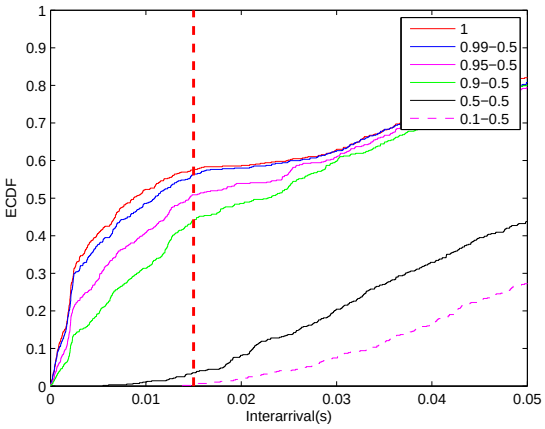
Muestreo sistemático



Muestreo aleatorio estratificado

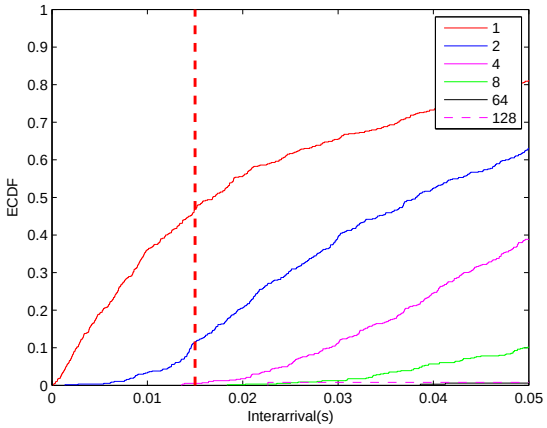


Muestreo aleatorio simple

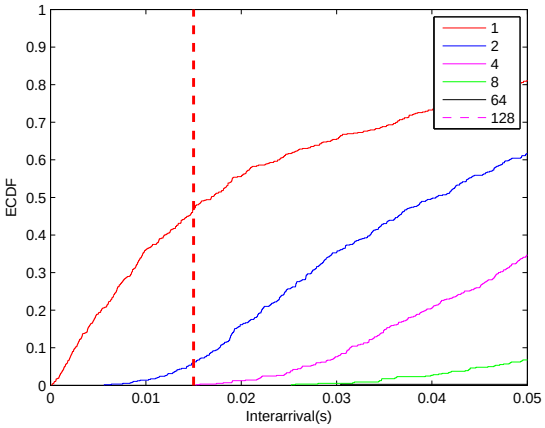


Pérdidas

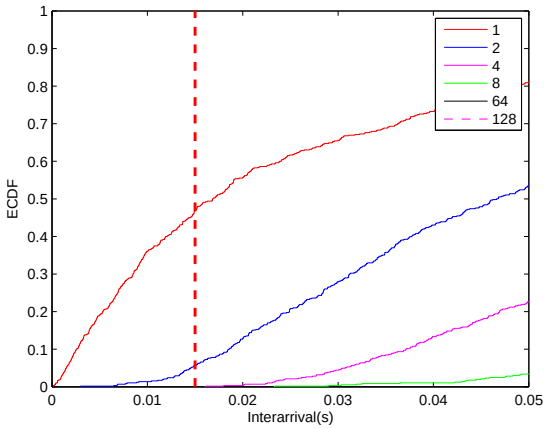
Figura 4.7: ECDF Interarrival: Traza 1



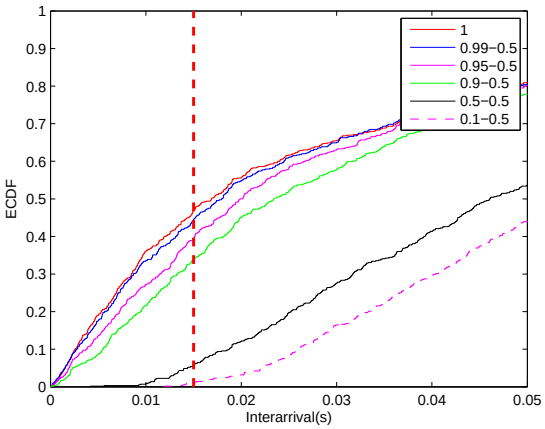
Muestreo sistemático



Muestreo aleatorio estratificado

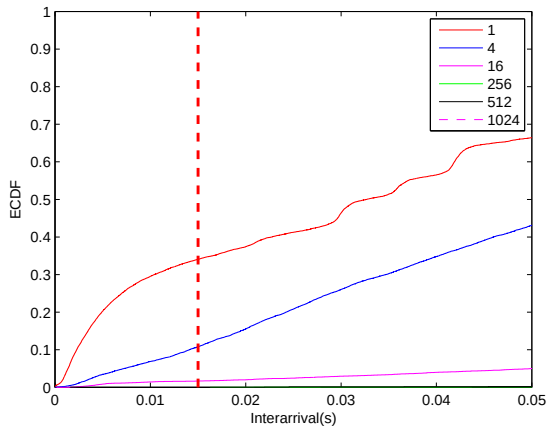


Muestreo aleatorio simple

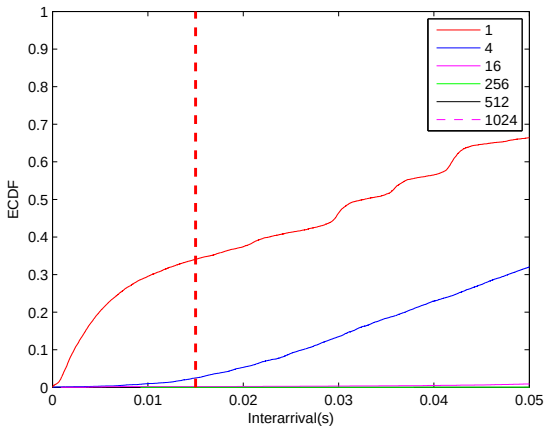


Pérdidas

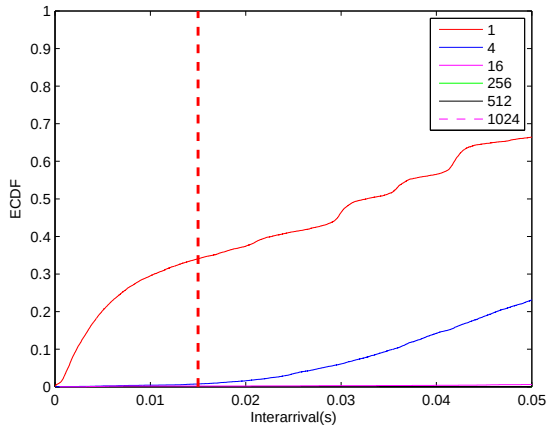
Figura 4.8: ECDF Interarrival: Traza 2



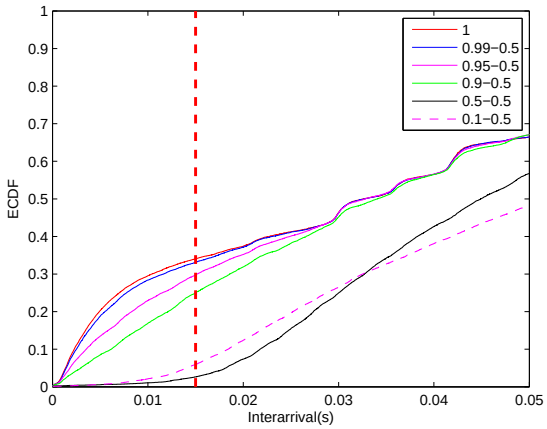
Muestreo sistemático



Muestreo aleatorio estratificado

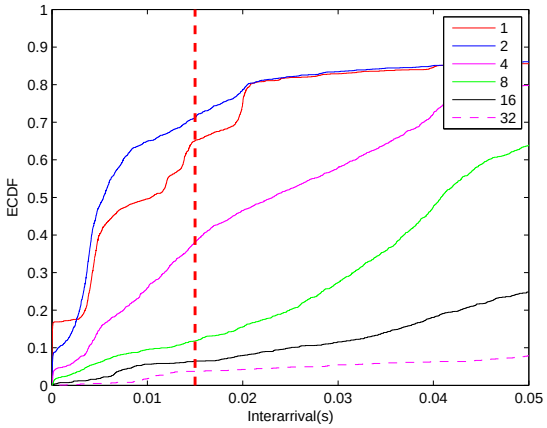


Muestreo aleatorio simple

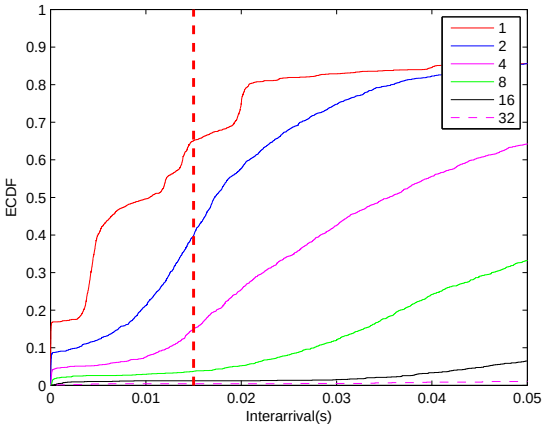


Pérdidas

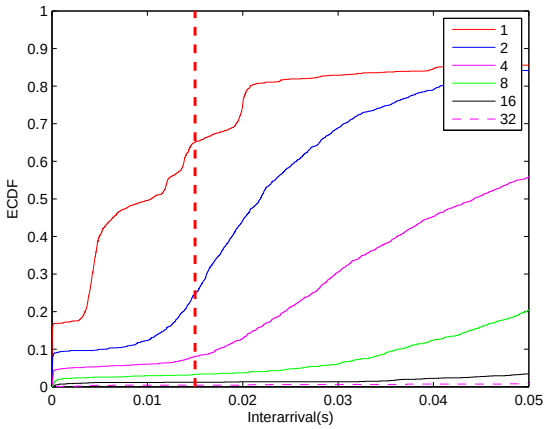
Figura 4.9: ECDF Interarrival: Traza 4



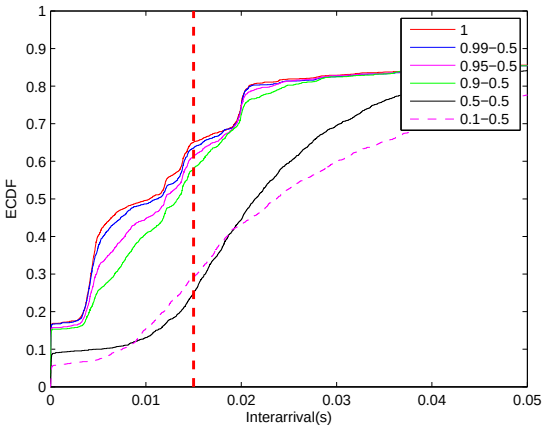
Muestreo sistemático



Muestreo aleatorio estratificado

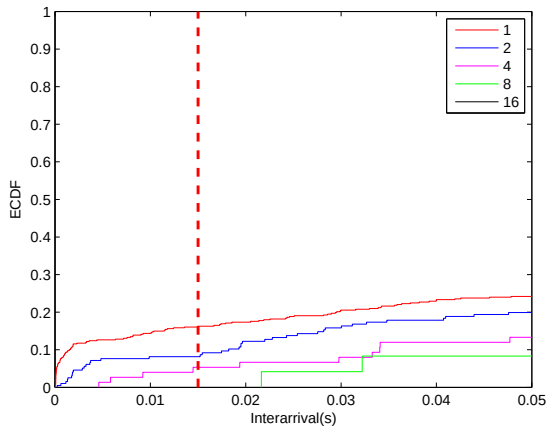


Muestreo aleatorio simple

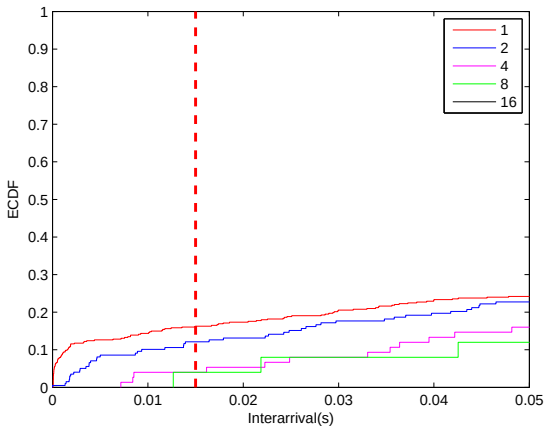


Pérdidas

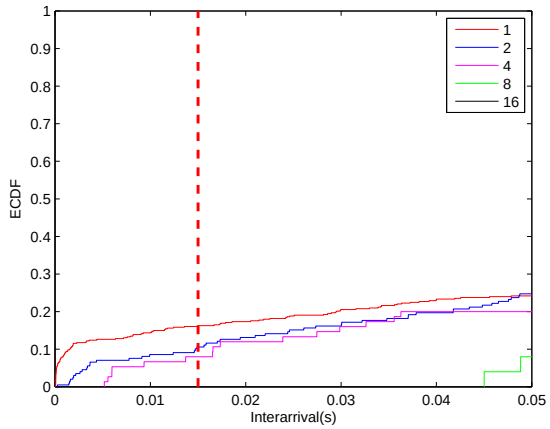
Figura 4.10: ECDF Interarrival: Traza 5



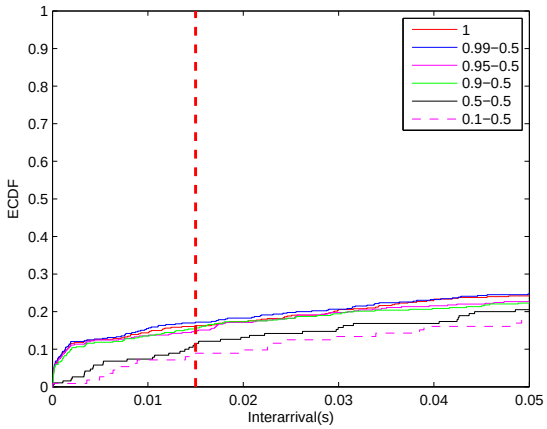
Muestreo sistemático



Muestreo aleatorio estratificado



Muestreo aleatorio simple



Pérdidas

Figura 4.11: ECDF Interarrival: Traza 6

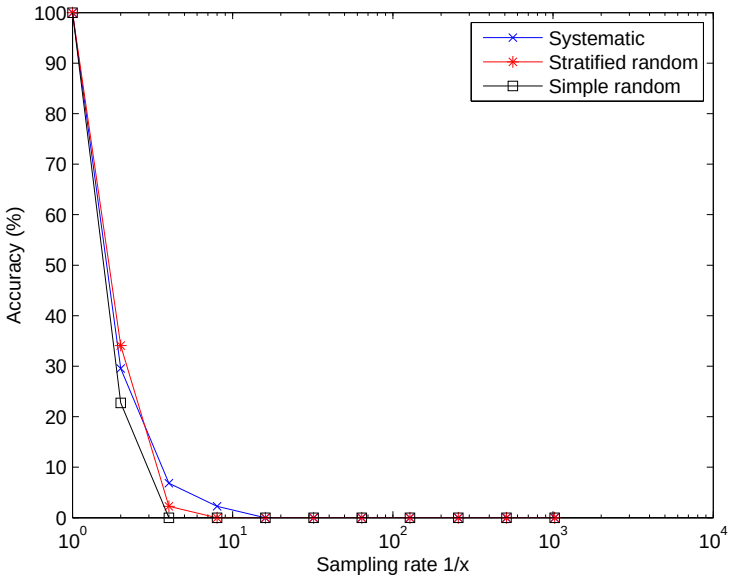
A diferencia de lo que ocurría en el análisis del tamaño del paquete, ahora observamos como el interarrival si que varía bastante, ya que a medida que tomamos menos paquetes nos vamos alejando del umbral deseado para detectar un flujo Skype, aumentando el interarrival, salvo en el caso de la figura 4.10 al aplicar el muestreo sistemático para la tasa de 1/2, donde se mejora. El tipo de muestreo utilizado no influye demasiado, aunque se observa mejores resultados por regla general para el muestro sistemático ya que se toman paquetes con la misma frecuencia. El interarrival será un factor a tener en cuenta en el cambio de precisión de Skypeness.

En el caso de la traza No-Skype se ve claramente como el interarrival está lejos del umbral necesario para clasificar el tráfico como Skype. Veremos si se confirma el comportamiento observado hasta ahora, en el siguiente apartado, donde se estudia el grado de acierto del detector.

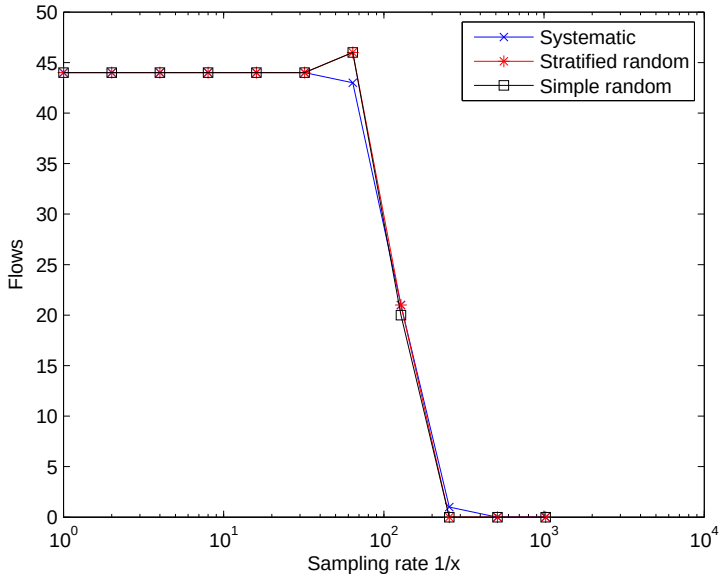
4.3. Impacto del muestreo de paquetes

A continuación mostraremos los resultados de cada traza e intentaremos explicar cómo varía la precisión del detector y el por qué actúa así, apoyándonos en el análisis estadístico y en otras características del tráfico. También observaremos como varía el número de flujos, paquetes y bytes según aplicamos los diferentes algoritmos.

4.3.1. Traza 1: audio

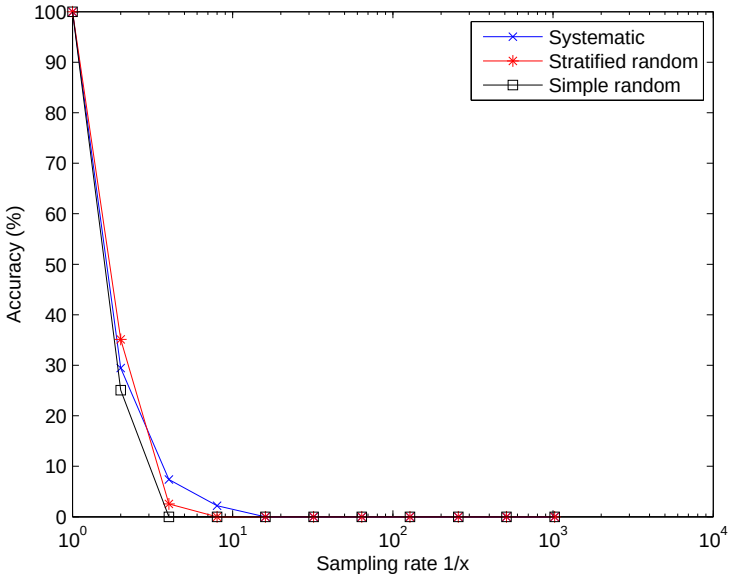


Precisión a nivel de flujo

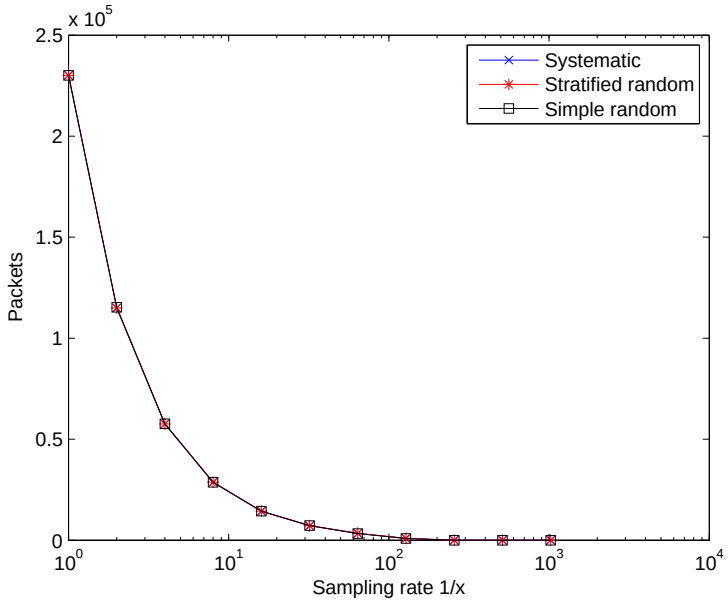


Número de flujos

Figura 4.12: Precisión(flujos): Muestreo Traza 1

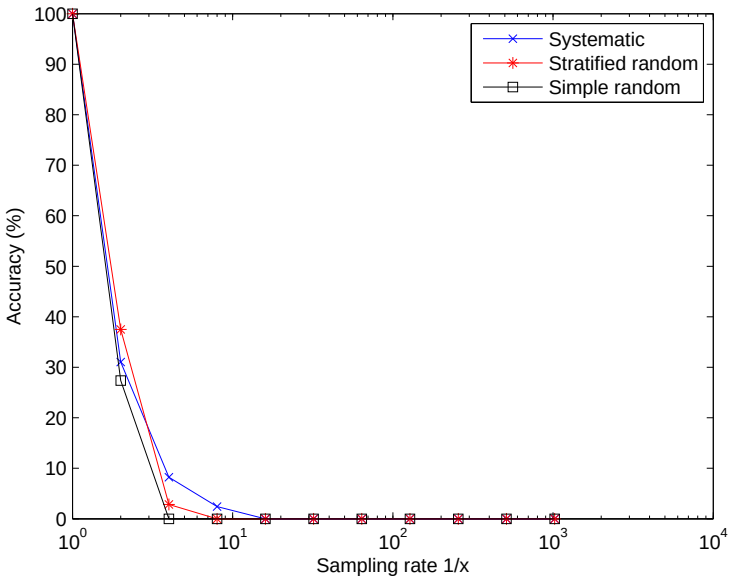


Precisión a nivel de paquete

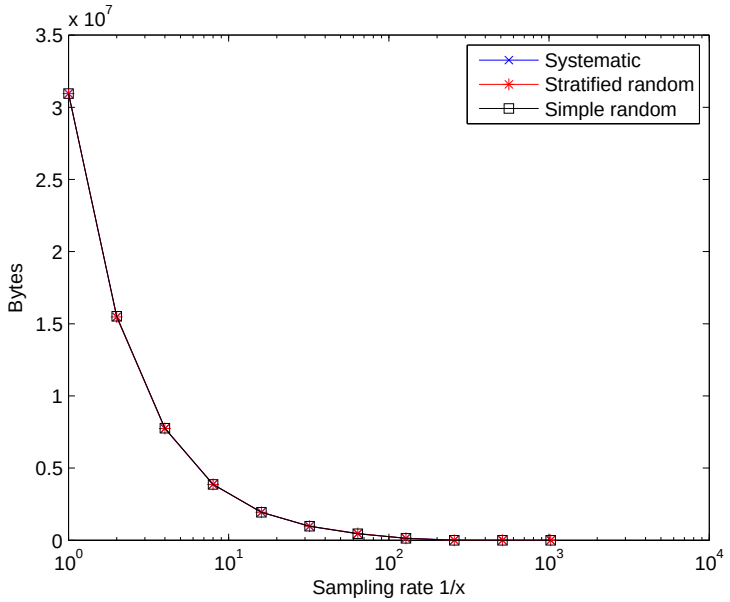


Número de paquetes

Figura 4.13: Precisión (paquetes): Muestreo Traza 1



Precisión a nivel de bytes



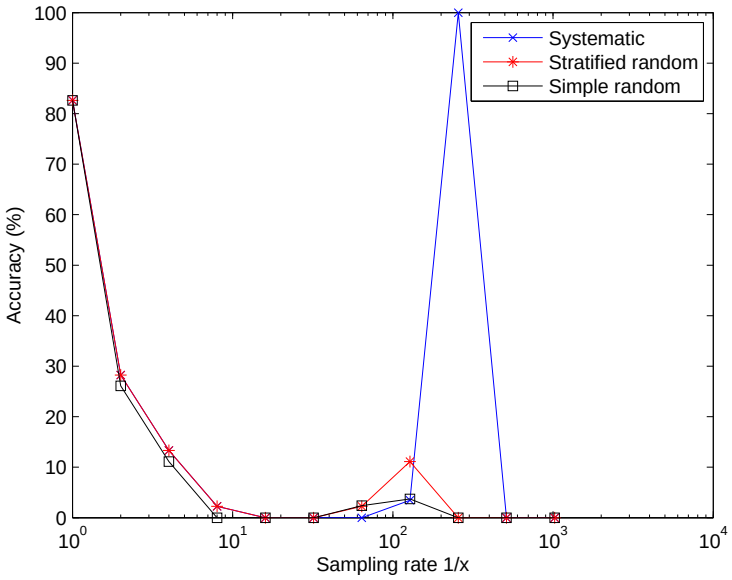
Número de bytes

Figura 4.14: Precisión (bytes): Muestreo Traza 1

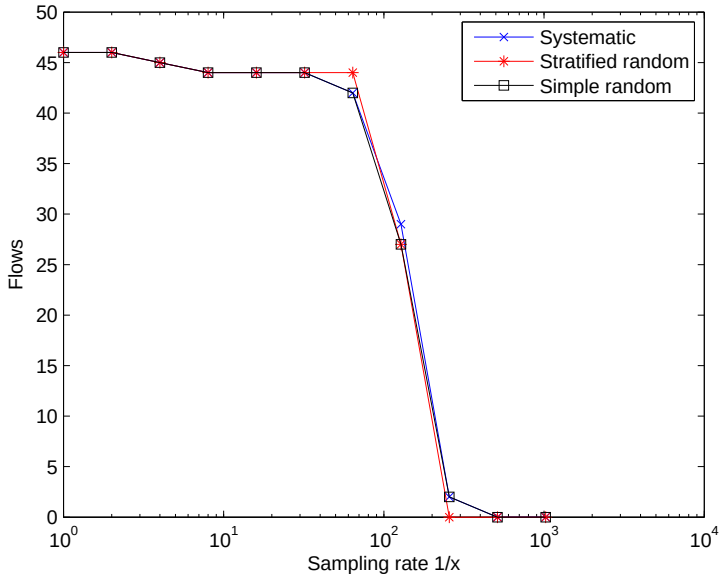
Una vez aplicado el algoritmo de muestreo, con los diferentes tipos y las diferentes tasas de muestreo, vemos que la precisión a nivel de flujo, paquete y byte, decae fuertemente según aumentamos la tasa de muestreo, llegando a una precisión de 0 % para una tasa de 1/16, figuras 4.12, 4.13 y 4.14 .

Este comportamiento se puede explicar si observamos la ECDF del interarrival de la traza 1, figura 4.7. A medida que se aumenta la tasa de muestreo, el interarrival cada vez es mayor alejándose del umbral para decidir que un flujo es Skype.

4.3.2. Traza 2: video

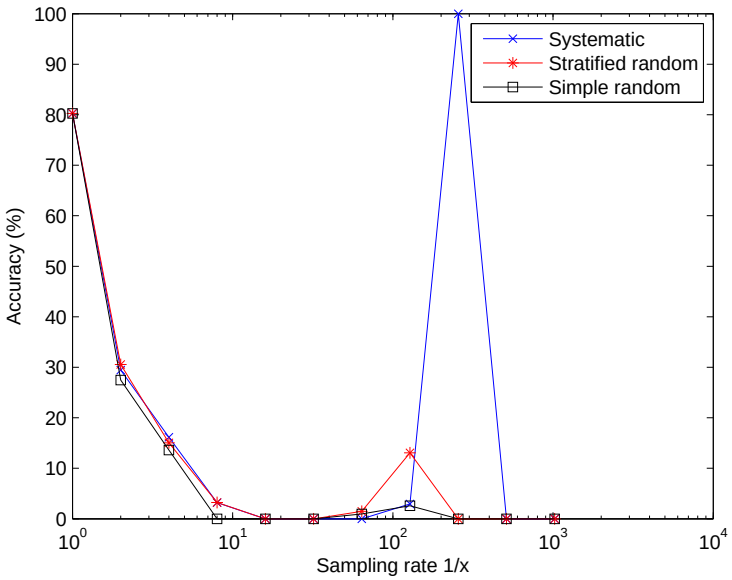


Precisión a nivel de flujo

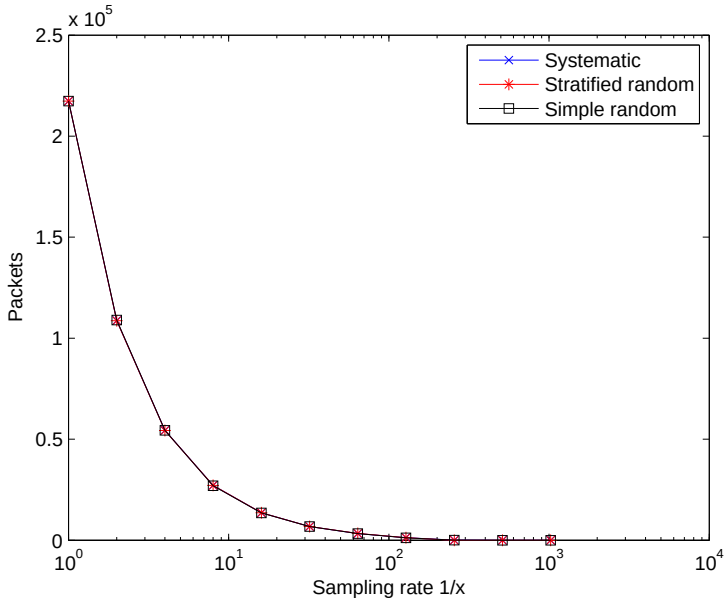


Número de flujos

Figura 4.15: Precisión (flujos): Muestreo Traza 2

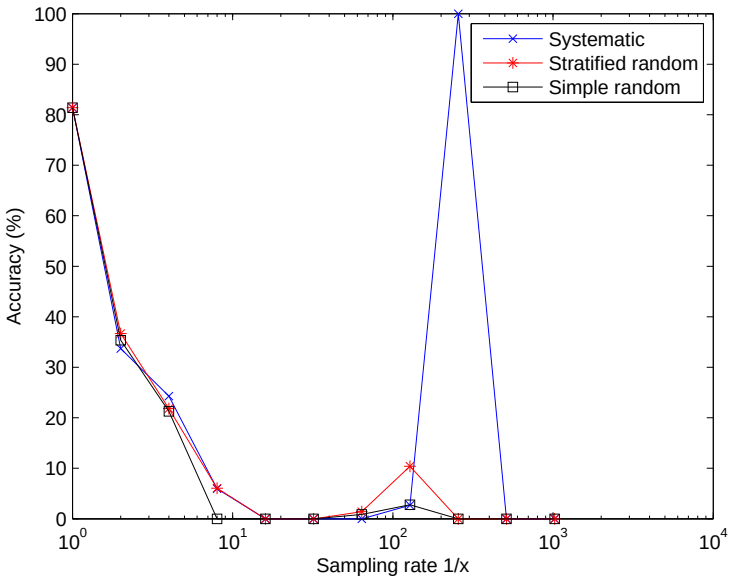


Precisión a nivel de paquete

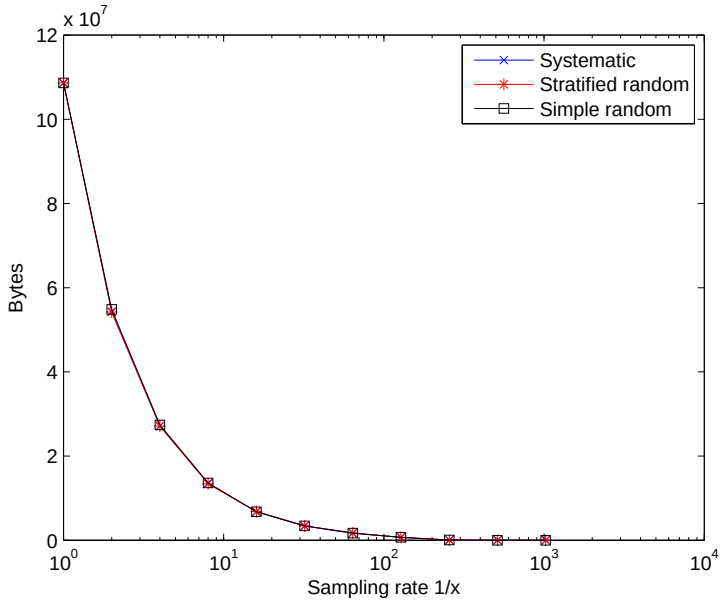


Número de paquetes

Figura 4.16: Precisión (paquetes): Muestreo Traza 2



Precisión a nivel de bytes



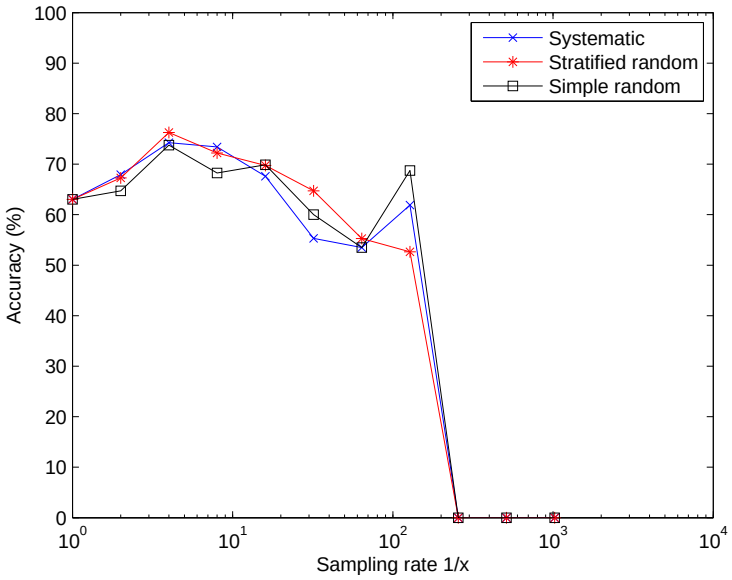
Número de bytes

Figura 4.17: Precisión (bytes): Muestreo Traza 2

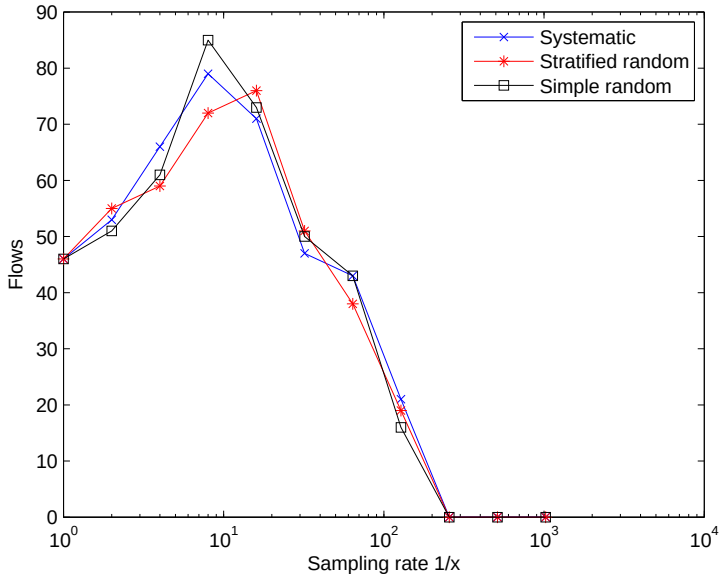
Para esta traza que contiene solo tráfico de vídeo la precisión, gráficas 4.15, 4.16 y 4.17, tiene un comportamiento similar que el de la traza de audio, ya que el factor que hace que falle Skypeness es el interarrival, figura 4.8.

La única diferencia es que para unas tasa de muestreo de 1/128 y 1/256 la precisión ha aumentado, y más en concreto para 1/256 donde se aumenta hasta alcanzar un 100 %, esto se debe a que el número de flujos se ha reducido considerablemente, como se puede observar en la gráfica 4.15, teniendo solo dos flujos y haciendo que si esos dos se clasifican como Skype la precisión será del 100 %, por tanto es un hecho bastante aleatorio que la precisión haya aumentado; además intuimos que al descartar paquetes nos quedamos con los flujos más largos, y que posiblemente sean los más estables y con mayor posibilidades de clasificarlos como Skype.

4.3.3. Traza 3: transferencia de archivo

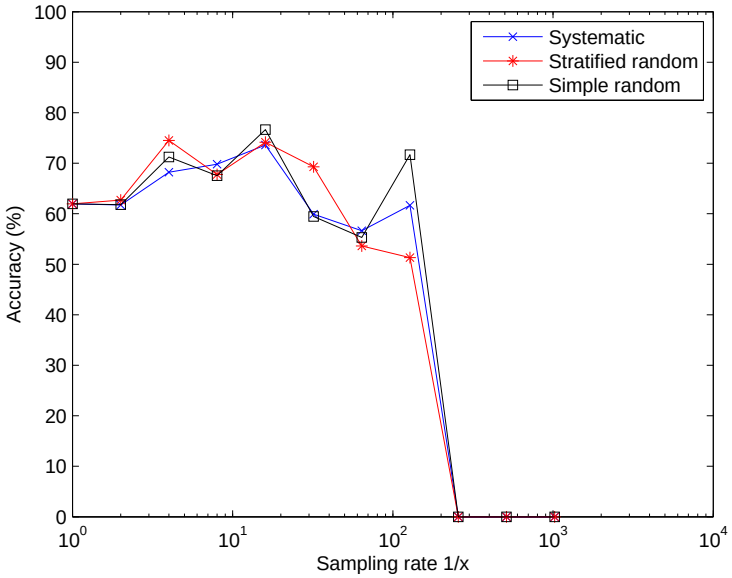


Precisión a nivel de flujo

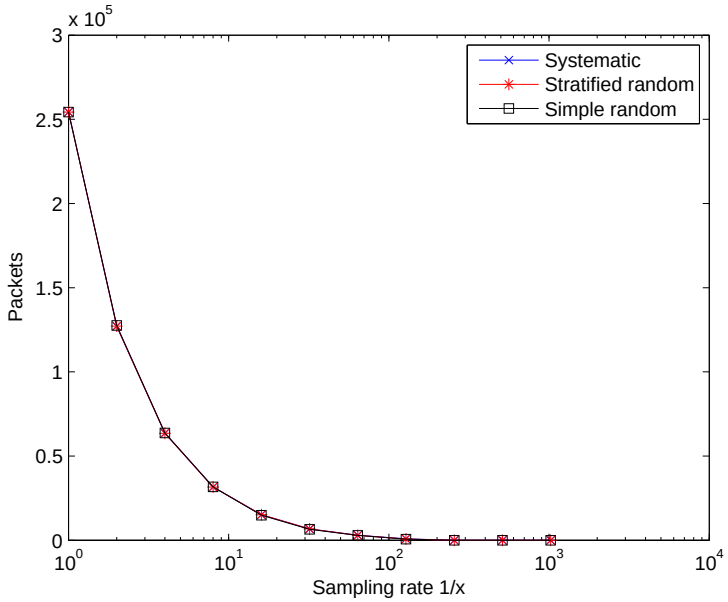


Número de flujos

Figura 4.18: Precisión (flujos): Muestreo Traza 3

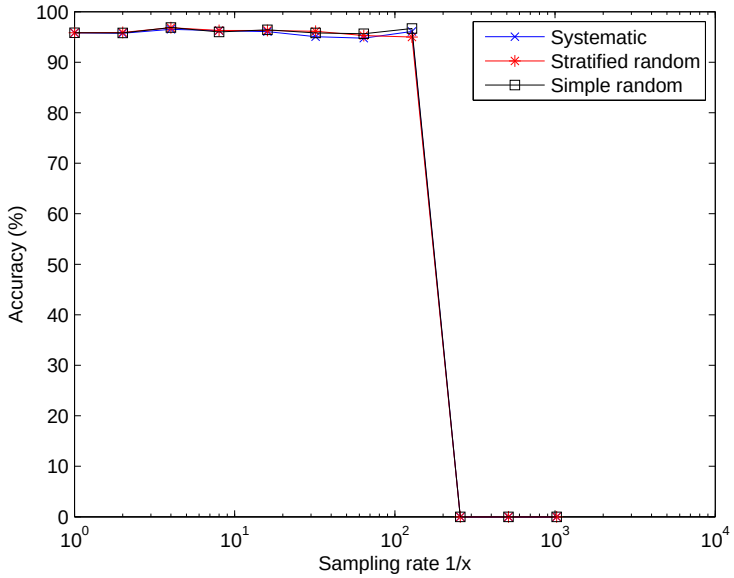


Precisión a nivel de paquete

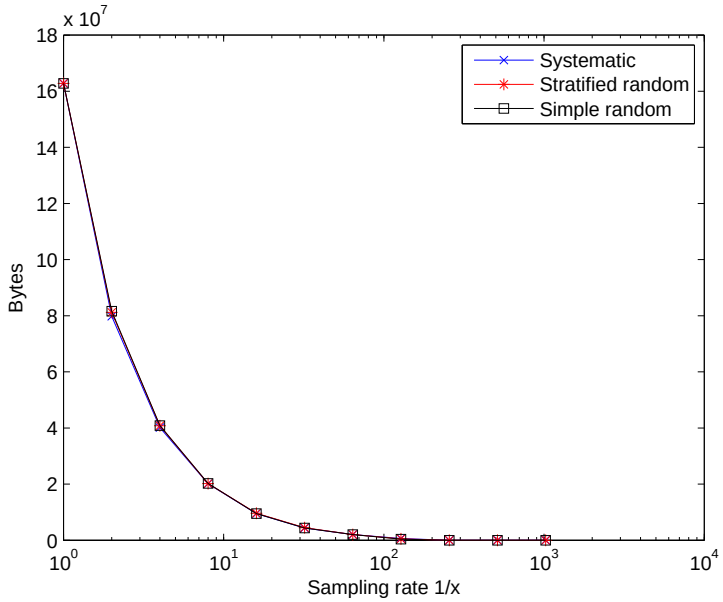


Número de paquetes

Figura 4.19: Precisión (paquetes): Muestreo Traza 3



Precisión a nivel de bytes



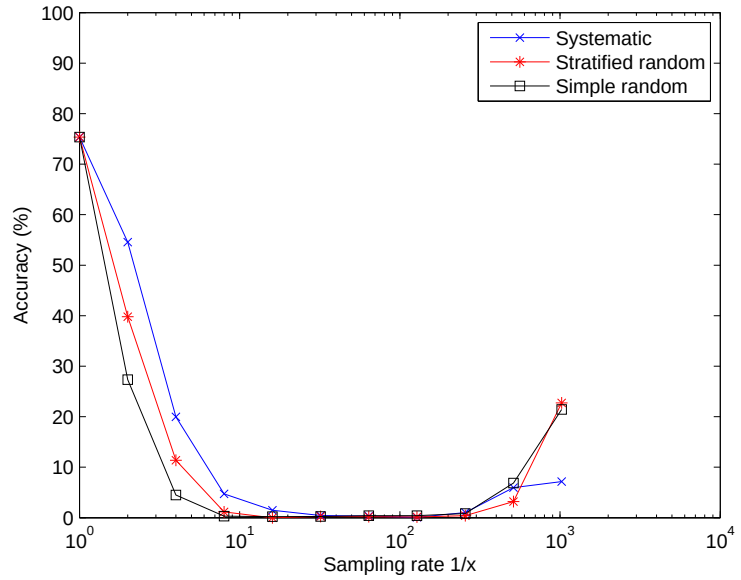
Número de bytes

Figura 4.20: Precisión (bytes): Muestreo Traza 3

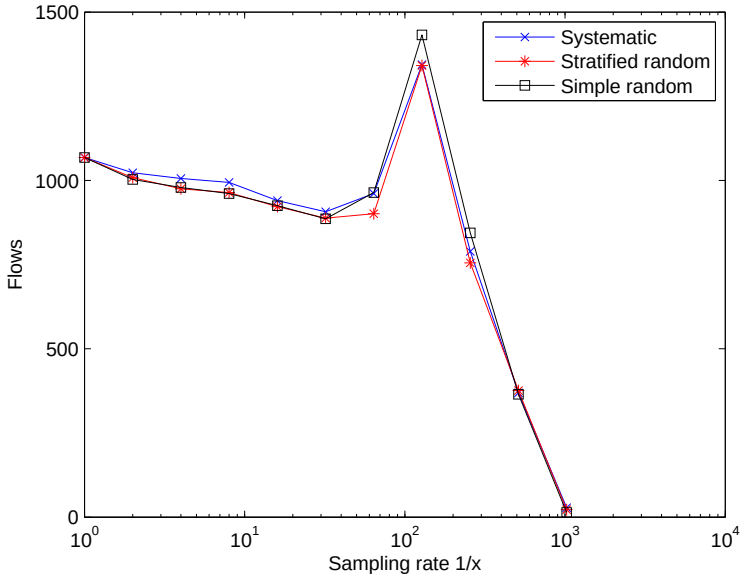
Ahora tenemos una traza Skype que se corresponde con el envío de archivos, en este caso el detector solo evalúa el tamaño del paquete, fig. 4.3, por tanto, como hemos explicado en el análisis estadístico, el muestreo y la pérdida de paquetes no afecta a la media del tamaño del paquete, y por esta razón, la precisión se mantiene más o menos en los mismos niveles, gráficas 4.18, 4.19 y 4.20, salvo para las tasas mayores 1/256, donde ya no se detectan flujos y la precisión es nula. Donde mejor se ve esto es en la precisión a nivel de byte, fig. 4.20, donde apenas varía la tasa de acierto de Skypeness.

Un hecho significativo es que el número de flujos aumenta a medida que tenemos menos paquetes, fig. 4.20. Esto puede ser debido a que esta traza contiene flujos mayores debido al envío de archivos y haga que se partan generando nuevos flujos.

4.3.4. Traza 4: llamadas E2E

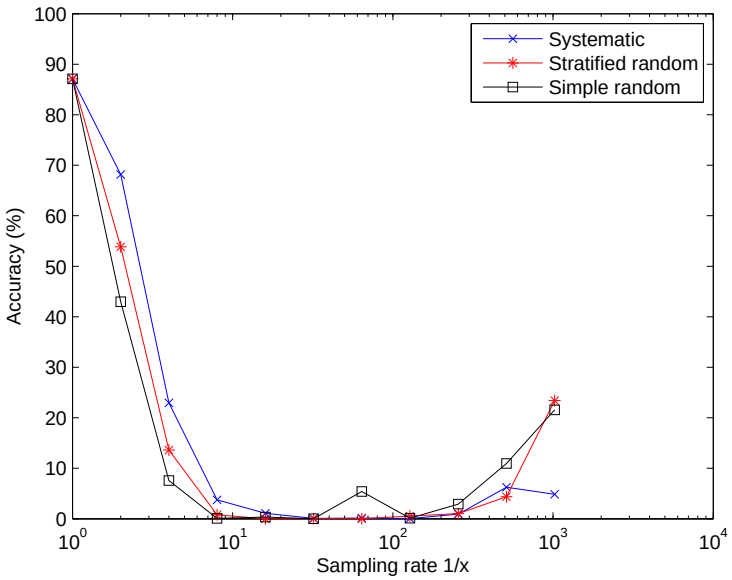


Precisión a nivel de flujo

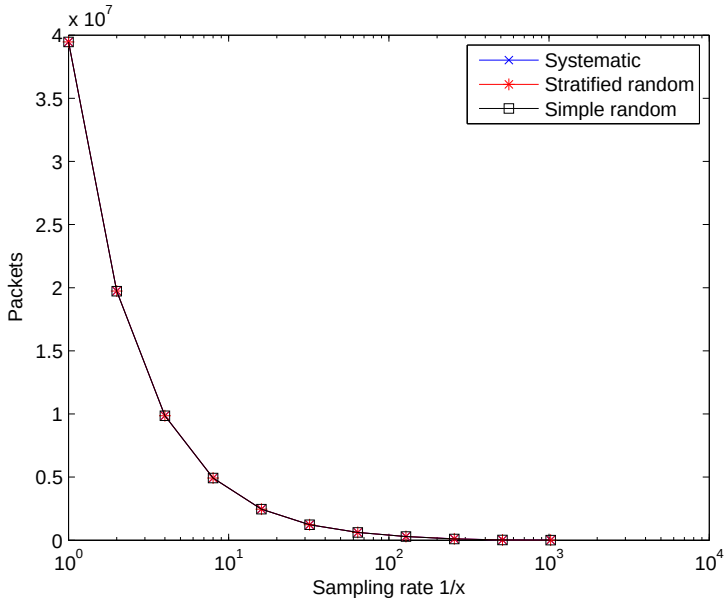


Número de flujos

Figura 4.21: Precisión (flujos): Muestreo Traza 4

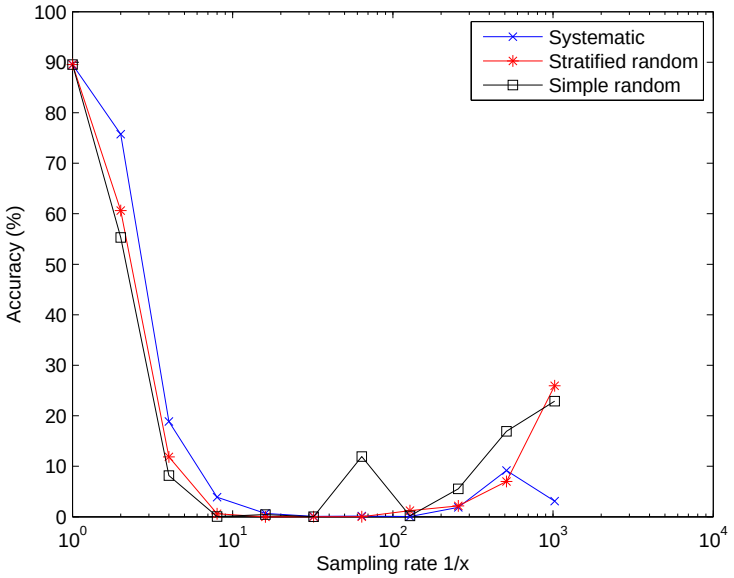


Precisión a nivel de paquete

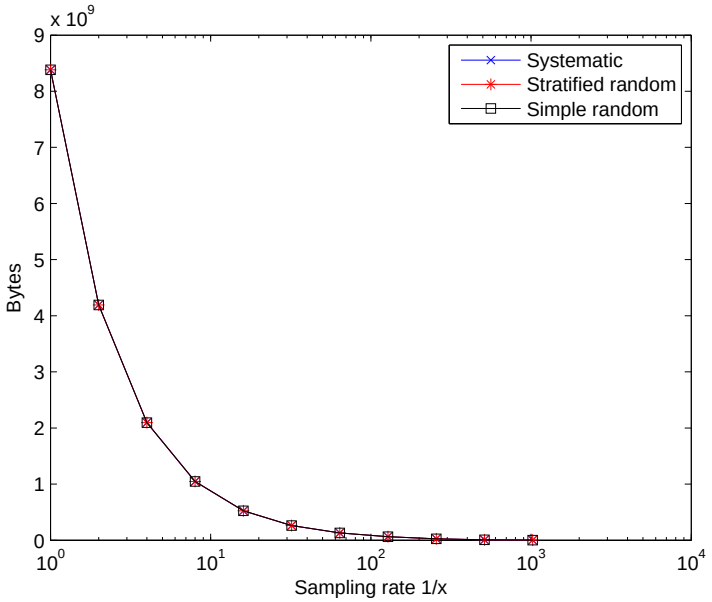


Número de paquetes

Figura 4.22: Precisión (paquetes): Muestreo Traza 4



Precisión a nivel de bytes



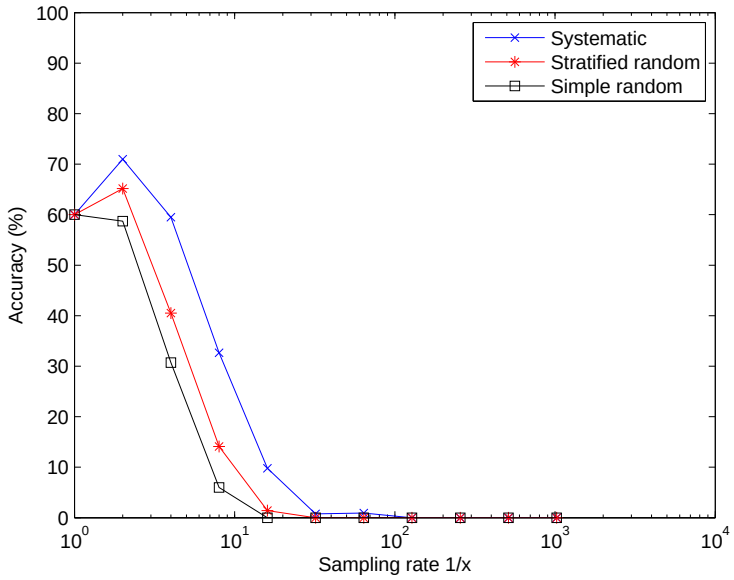
Número de bytes

Figura 4.23: Precisión (bytes): Muestreo Traza 4

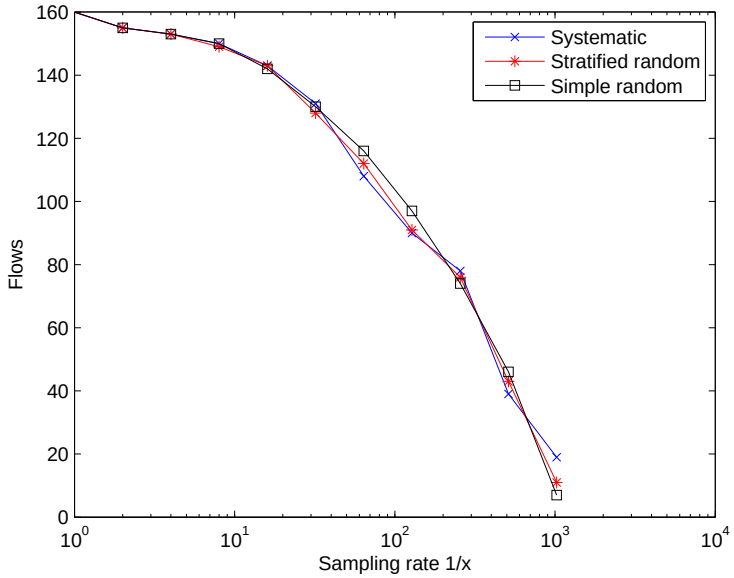
La siguiente traza está compuesta por audio y vídeo entre ordenadores, E2E. Observamos como la precisión decae según se van descartando paquetes, figuras 4.21, 4.22 y 4.23, igual que pasaba con las trazas anteriores de audio y vídeo, debido al interarrival, fig. 4.9.

Pero también vemos como para tasas altas de muestreo la precisión aumenta ligeramente, esto sucede como pasaba con la traza 2, por el bajo número de flujos que ahora se reconocen.

4.3.5. Traza 5: llamadas E2O

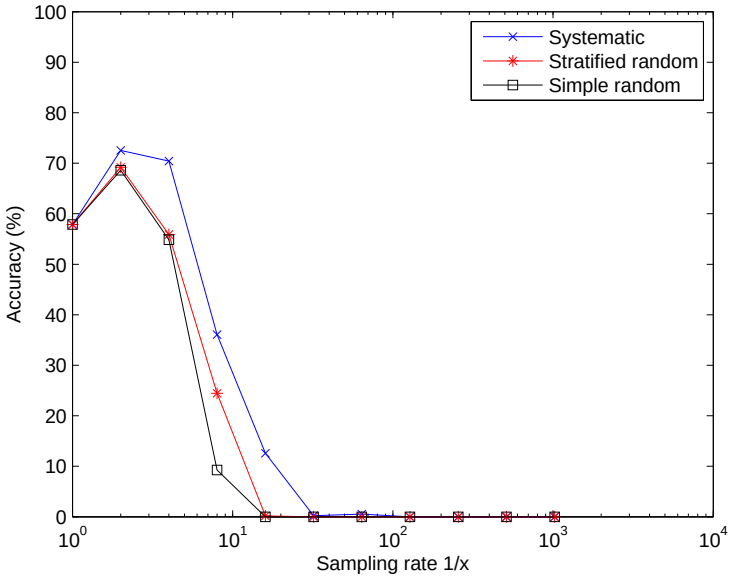


Precisión a nivel de flujo

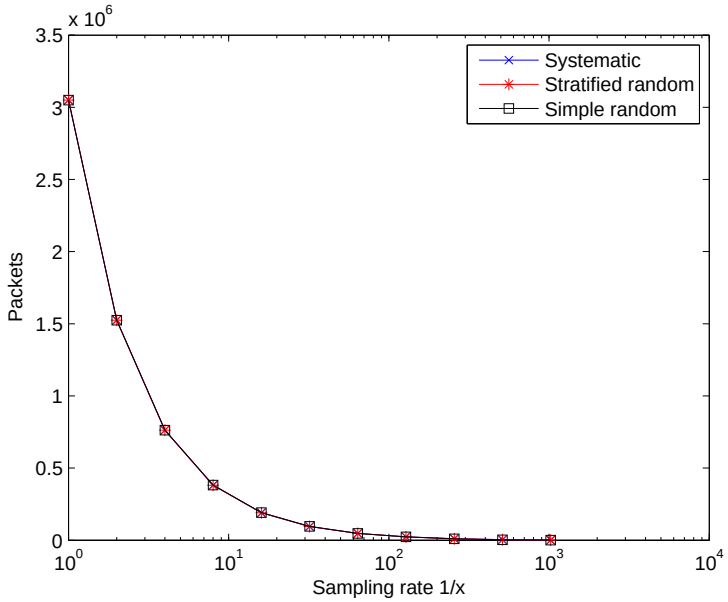


Número de flujos

Figura 4.24: Precisión (flujos): Muestreo Traza 5

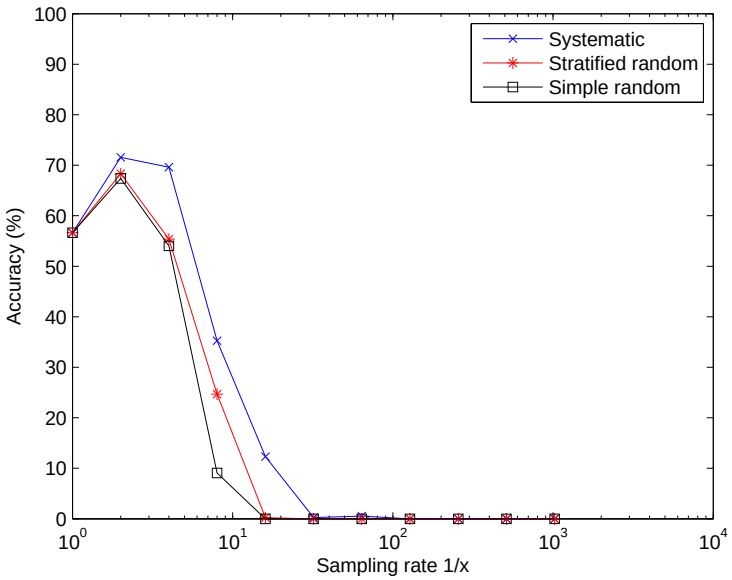


Precisión a nivel de paquete

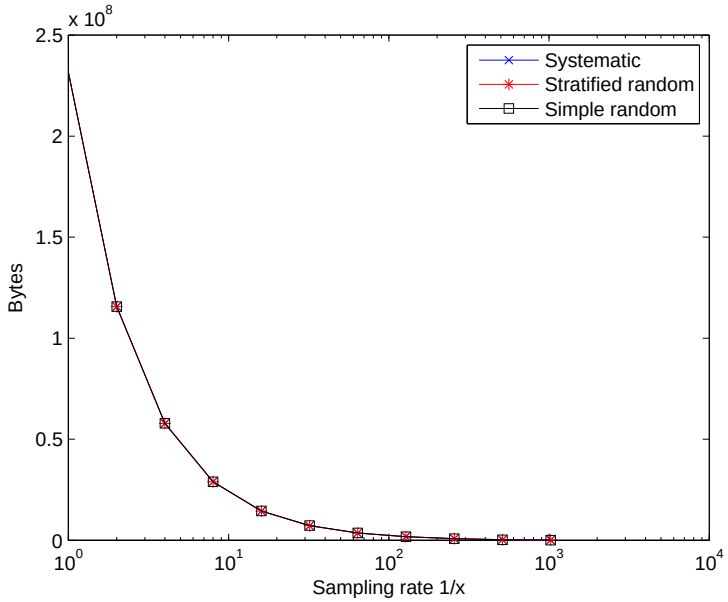


Número de paquetes

Figura 4.25: Precisión (paquetes): Muestreo Traza 5



Precisión a nivel de bytes

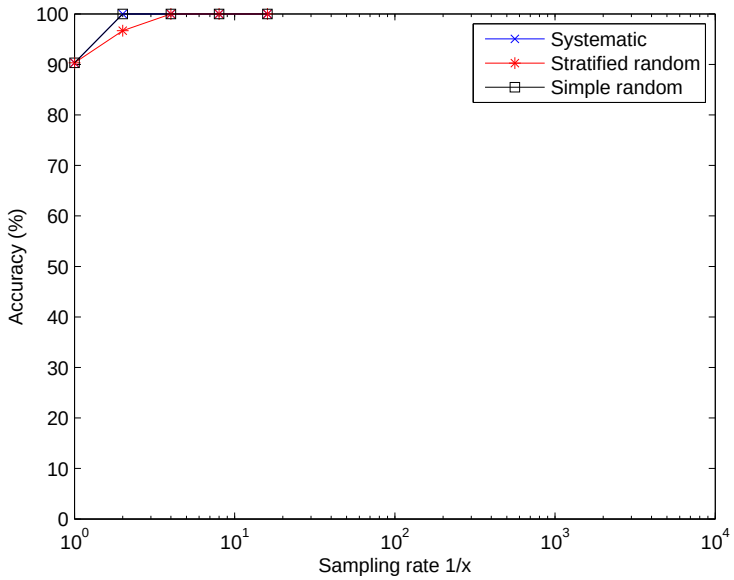


Número de bytes

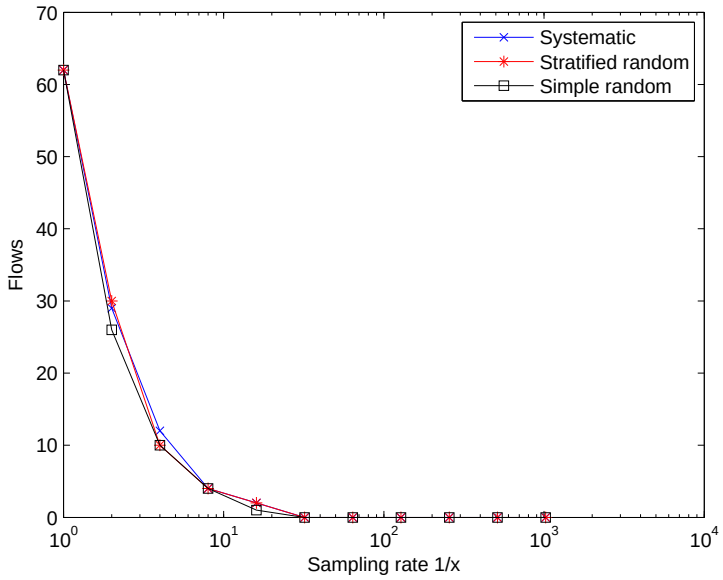
Figura 4.26: Precisión (bytes): Muestreo Traza 5

Con esta traza sucede algo que no había ocurrido antes, y es que al muestrear de forma determinista la traza con una tasa de 1/2 la precisión aumenta, figuras 4.21, 4.22 y 4.23, esto es debido a que estamos descartando paquetes donde el interarrival era mayor, es decir eliminamos los flujos peores, y de esta forma su media es menor como podemos observar en la gráfica 4.10. Para los demás casos todo sucede como pasaba anteriormente, la precisión disminuye al perder paquetes.

4.3.6. Traza 6: tráfico No-Skype

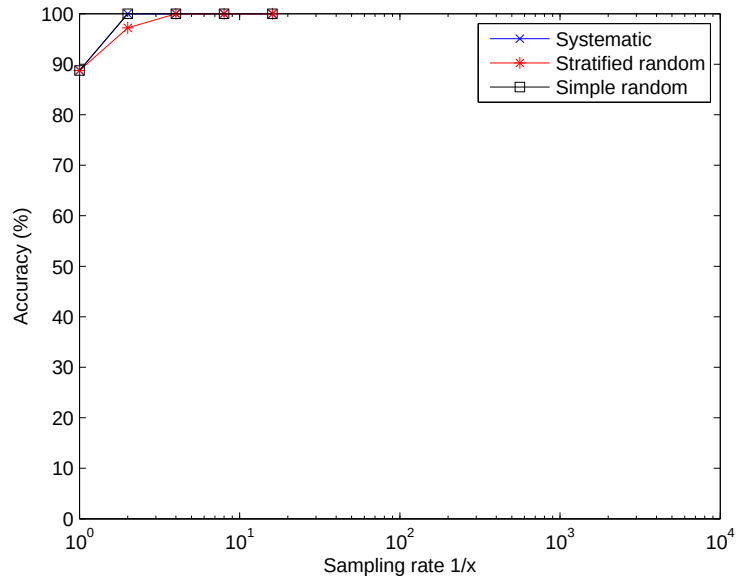


Precisión a nivel de flujo

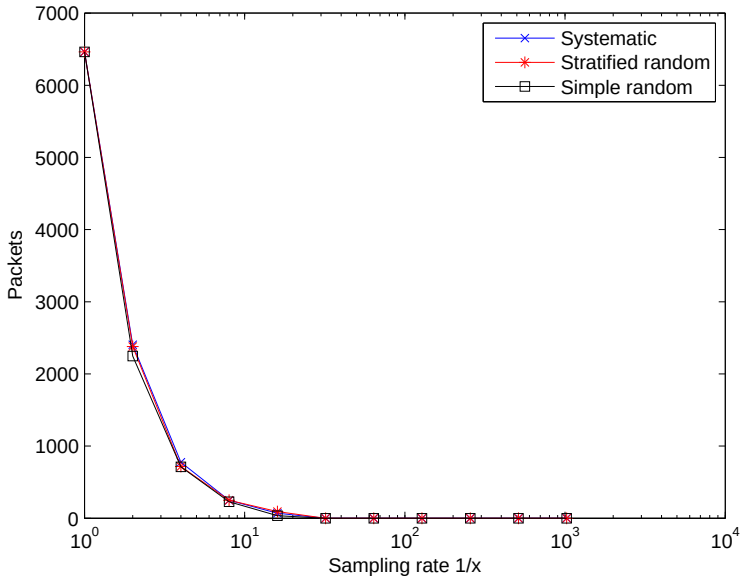


Número de flujos

Figura 4.27: Precisión (flujos): Muestreo Traza 6

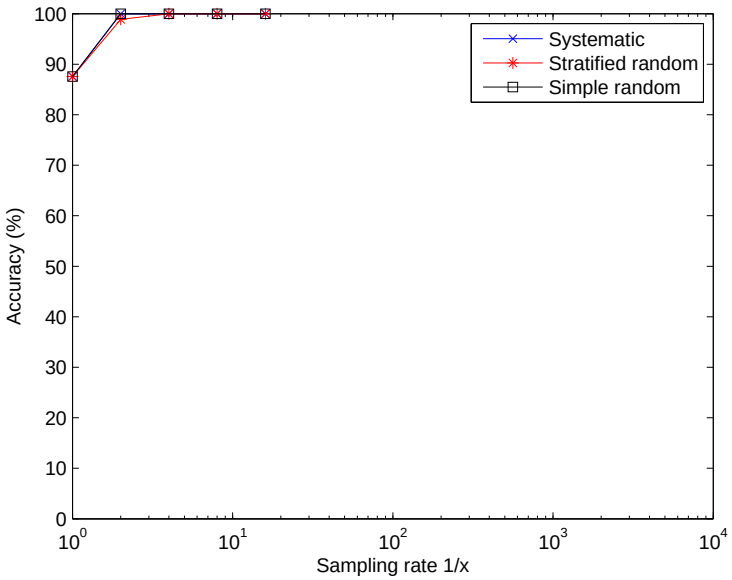


Precisión a nivel de paquete

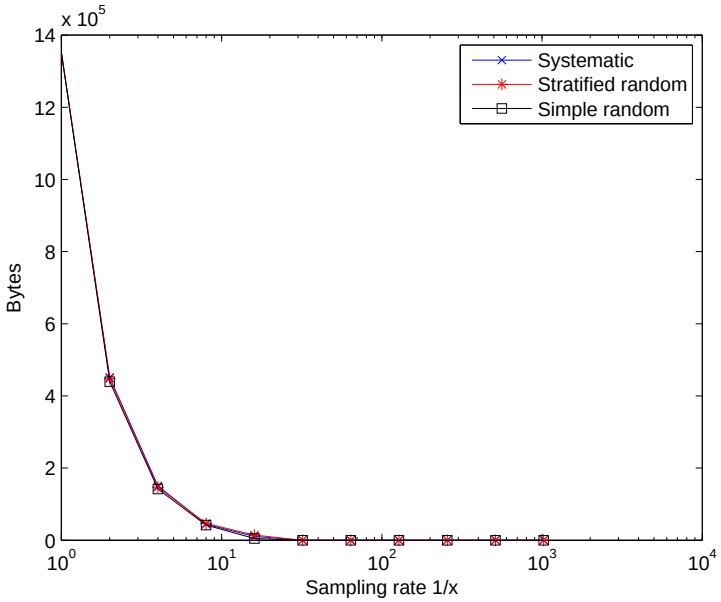


Número de paquetes

Figura 4.28: Precisión (paquetes): Muestreo Traza 6



Precisión a nivel de bytes



Número de bytes

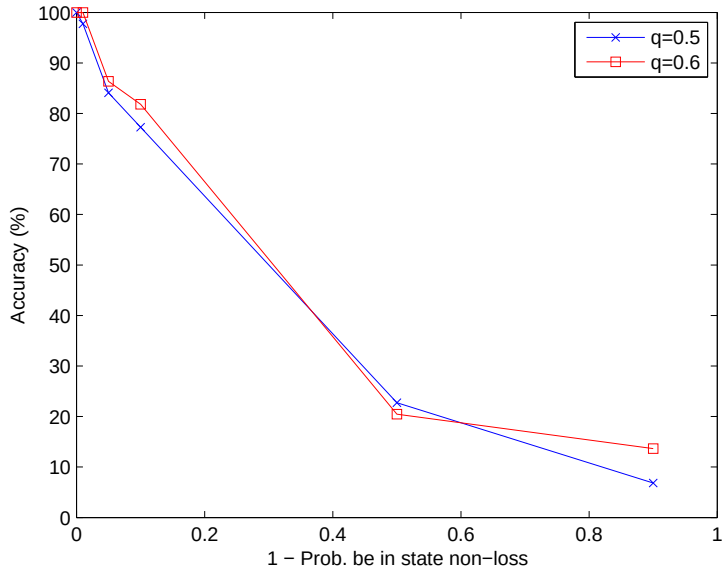
Figura 4.29: Precisión (bytes): Muestreo Traza 6

En el caso de tener una traza sin tráfico Skype, el detector es bastante preciso, la tasa de falsos positivos vemos que es muy baja y a medida que aumentamos la tasa de muestreo la precisión mejora llegando rápidamente a 100 % como vemos en las figuras 4.27, 4.28 y 4.29.

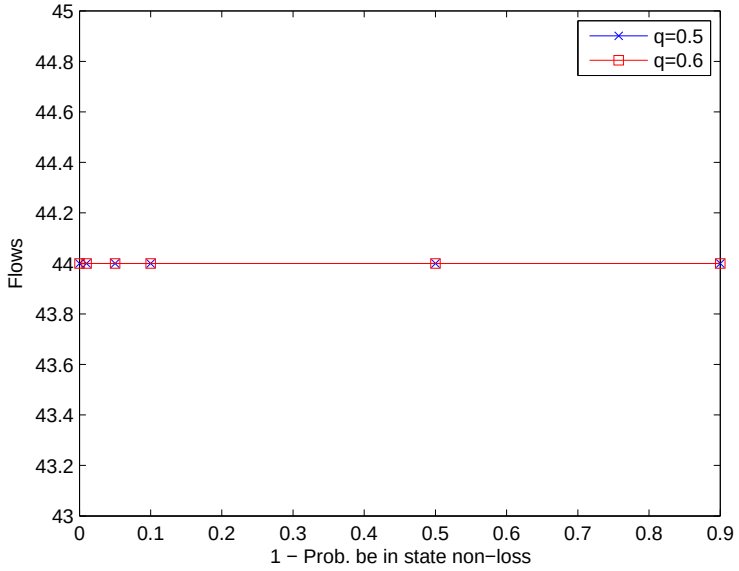
Estos resultados se deben principalmente al interarrival, fig. 4.11, ya que son diferentes a los intervalos definidos para Skype, mientras que los tamaños de los paquetes si podrían estar en los umbrales de Skype, fig. 4.6.

4.4. Impacto de la pérdida de paquetes

4.4.1. Traza 1: audio

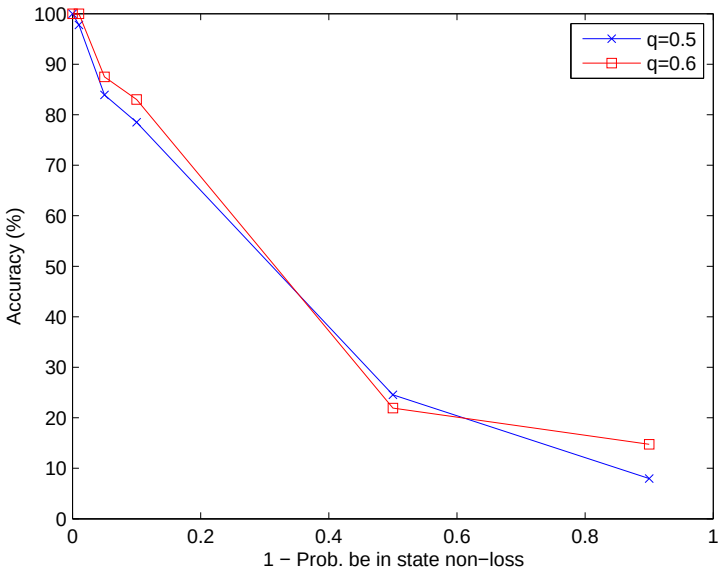


Precisión a nivel de flujo

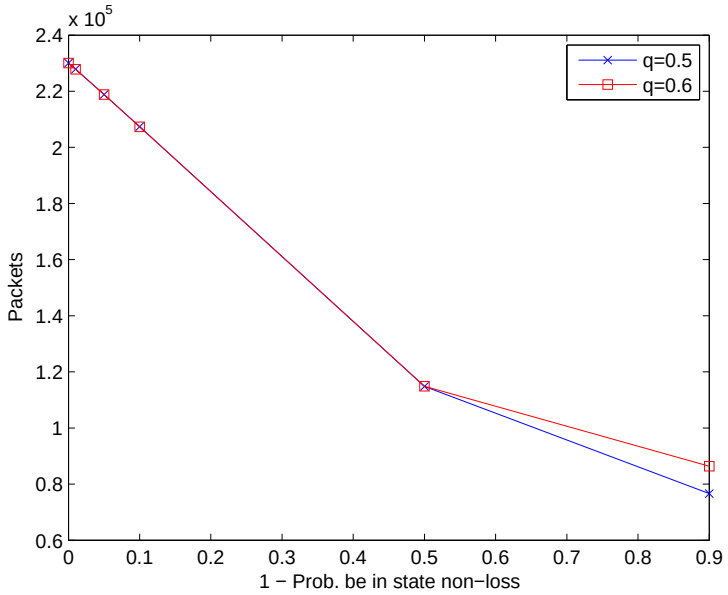


Número de flujos

Figura 4.30: Precisión (flujos): Pérdidas Traza 1

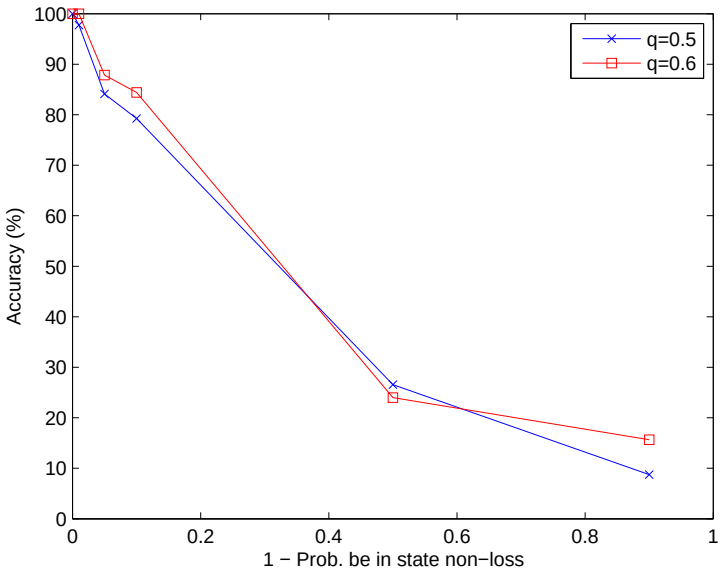


Precisión a nivel de paquete

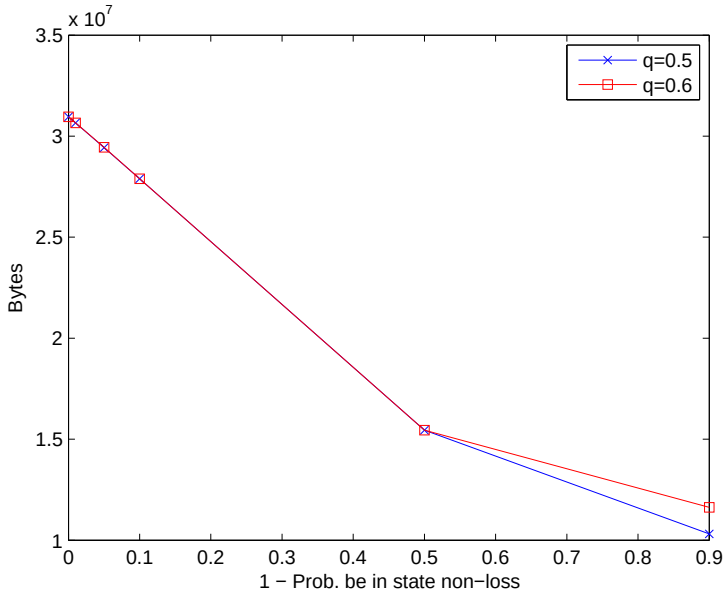


Número de paquetes

Figura 4.31: Precisión (paquetes): Pérdidas Traza 1



Precisión a nivel de bytes

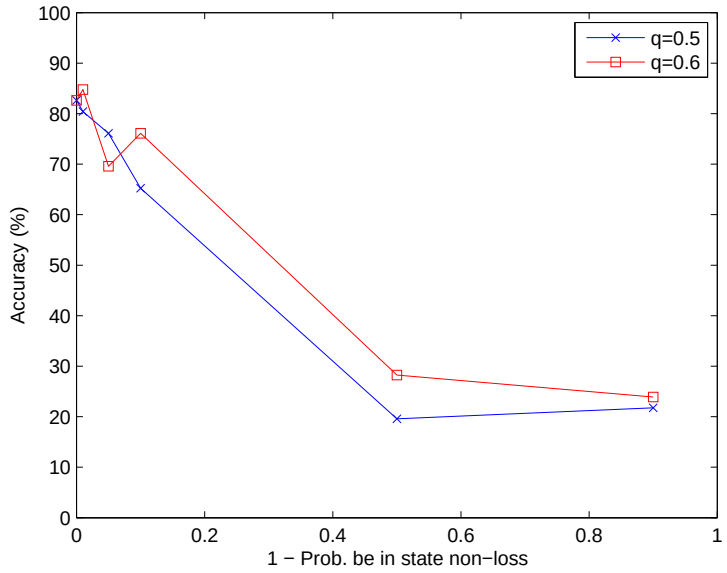


Número de bytes

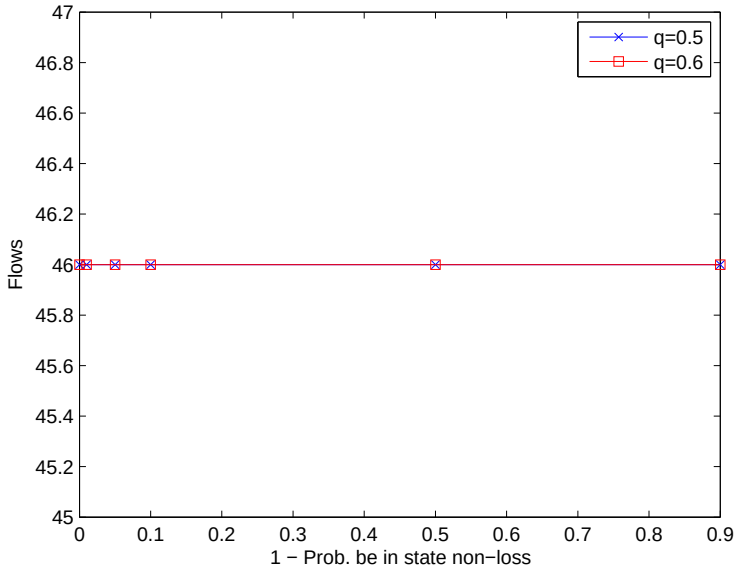
Figura 4.32: Precisión (bytes): Pérdidas Traza 1

En el caso de simular pérdidas vemos que ocurre algo similar a lo ocurrido con el muestreo, puesto que según aumenta la probabilidad de perder paquetes, el interarrival aumenta, como vemos en la fig. 4.7, y la precisión de Skypeness va disminuyendo clasificando el tráfico como no Skype, gráficas 4.30, 4.31 y 4.32. La precisión del detector se comporta de manera similar, tanto a nivel de flujo, de paquete y de byte.

4.4.2. Traza 2: video

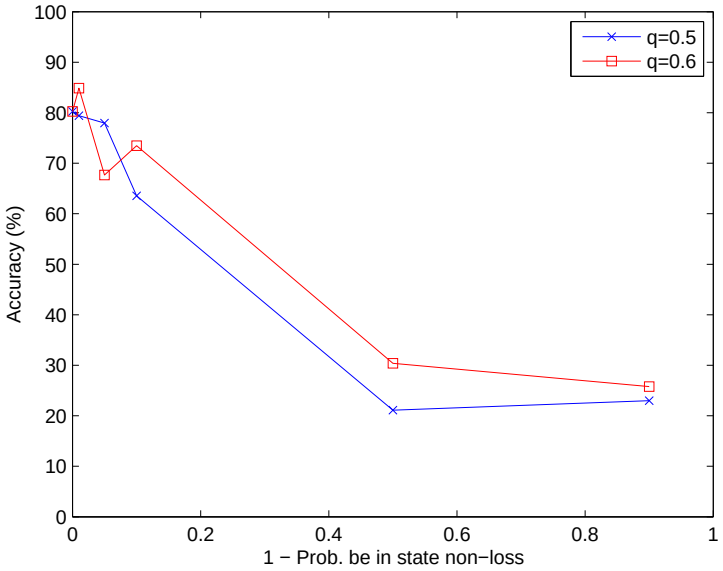


Precisión a nivel de flujo

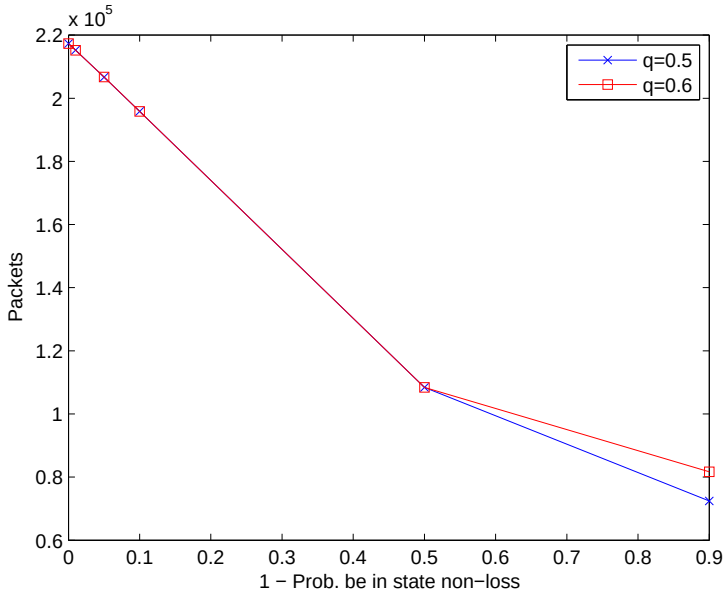


Número de flujos

Figura 4.33: Precisión (flujos): Pérdidas Traza 2

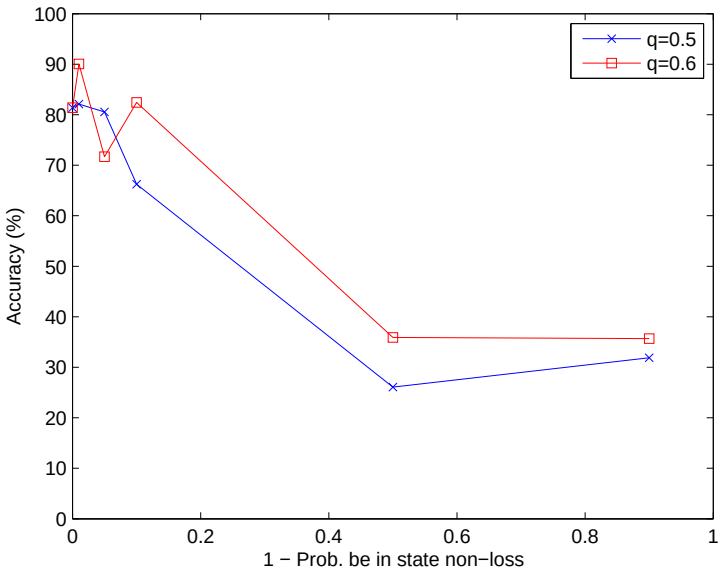


Precisión a nivel de paquete

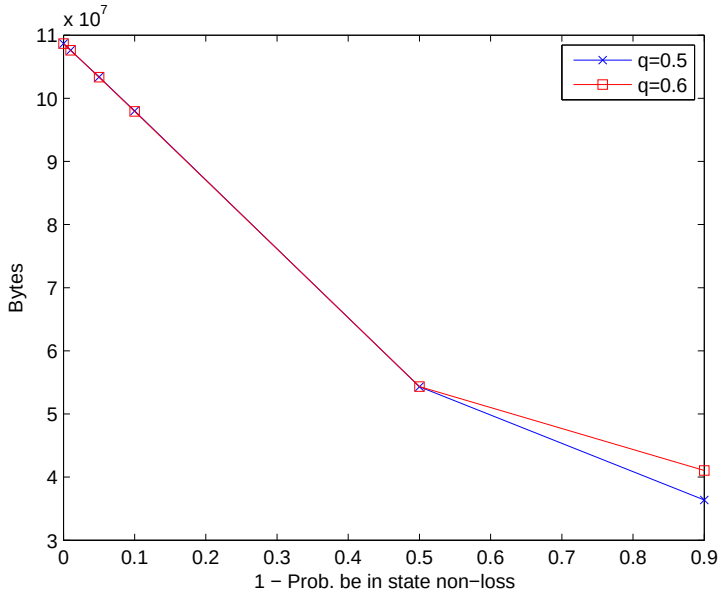


Número de paquetes

Figura 4.34: Precisión (paquetes): Pérdidas Traza 2



Precisión a nivel de bytes

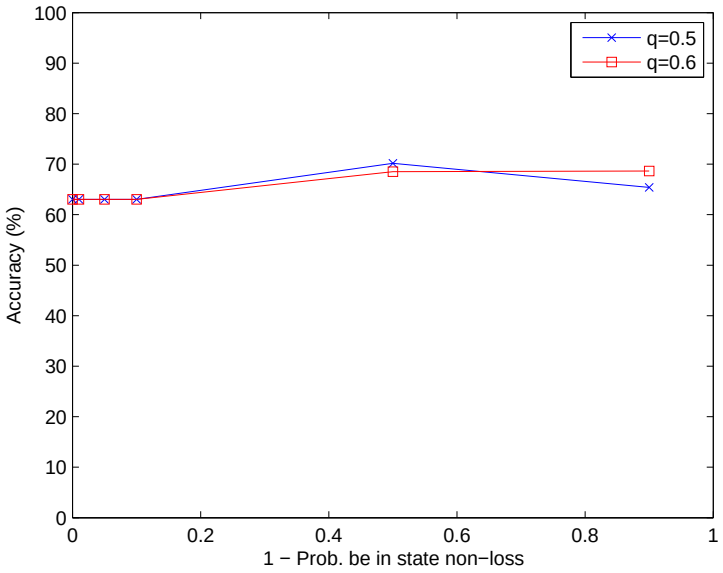


Número de bytes

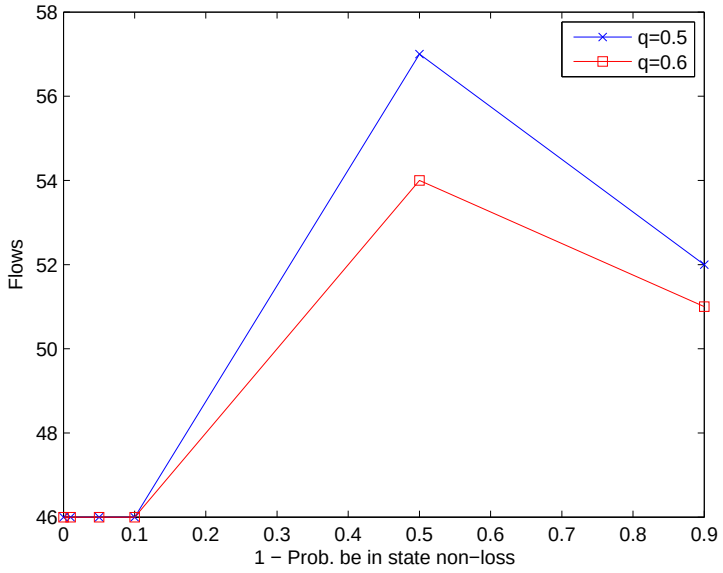
Figura 4.35: Precisión (bytes): Pérdidas Traza 2

En este caso ocurre de forma similar a lo que ocurría con el caso anterior, puesto que a medida que perdemos paquetes va decayendo la precisión como se observa en las figuras 4.33, 4.34 y 4.35, debido a la disminución del interarrival, con la única diferencia de que para la probabilidad de pasar del estado de pérdidas a no pérdidas de 0.6 vemos que se produce un aumento de precisión muy leve, esto puede ser debido a que cuando las probabilidades de perder paquetes son bajas las diferencias entre interarrival son muy pequeñas, casi iguales fig. 4.8, en cambio observando la distribución del tamaño del paquete, fig. 4.2, hay una mejora muy leve en estos casos produciendo dicho cambio de la precisión.

4.4.3. Traza 3: transferencia de archivo

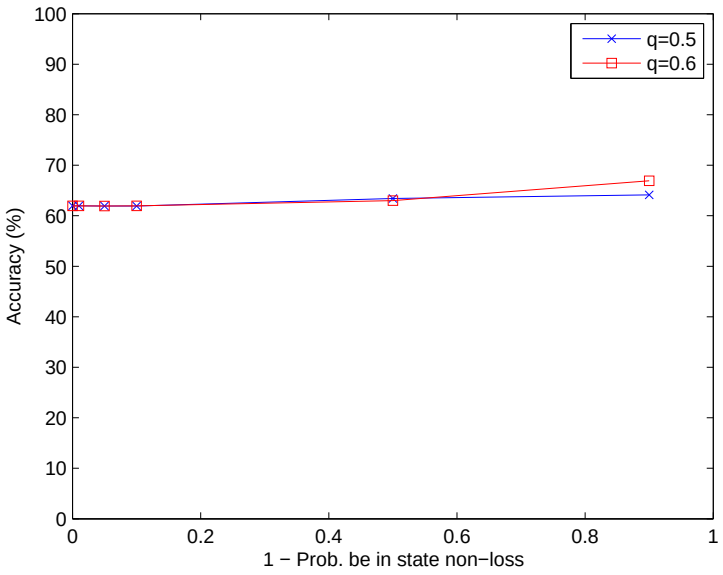


Precisión a nivel de flujo

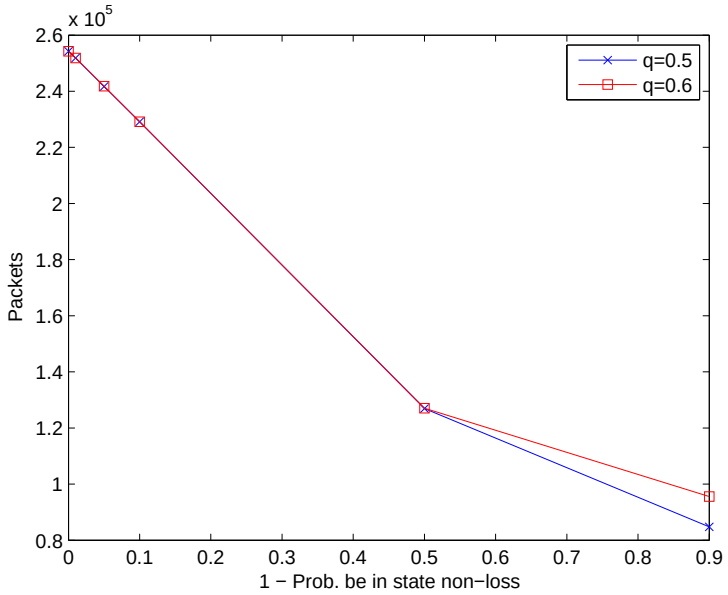


Número de flujos

Figura 4.36: Precisión (flujos): Pérdidas Traza 3

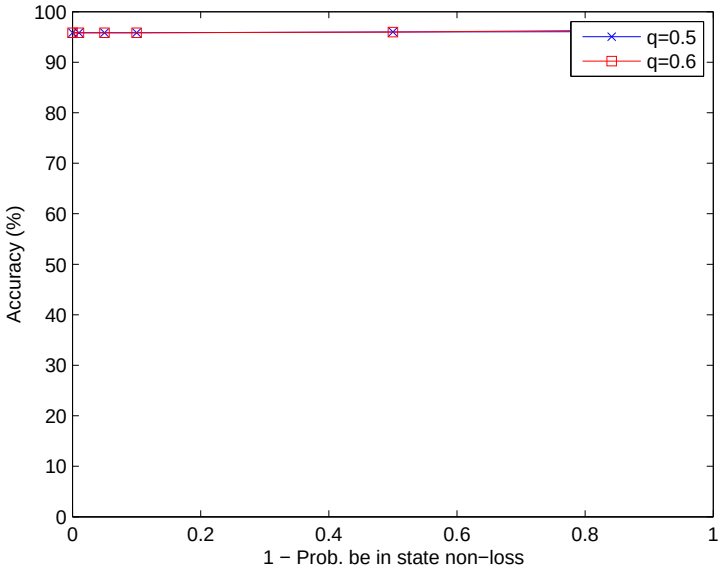


Precisión a nivel de paquete

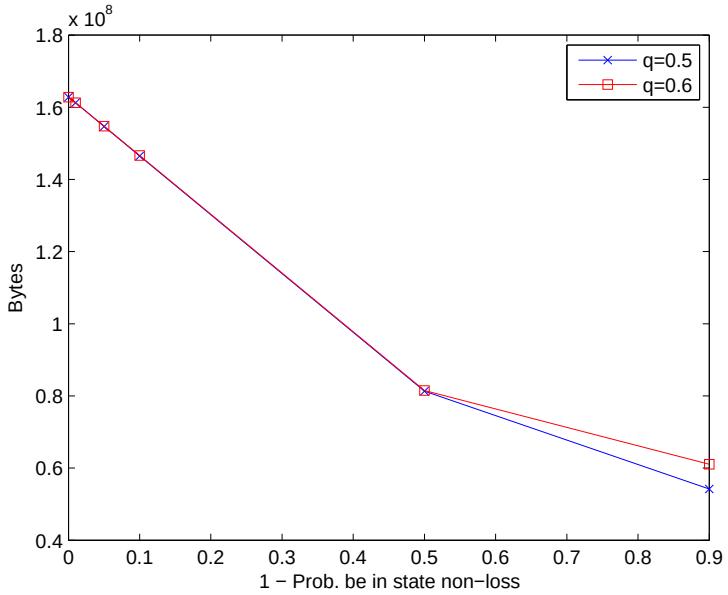


Número de paquetes

Figura 4.37: Precisión (paquetes): Pérdidas Traza 3



Precisión a nivel de bytes

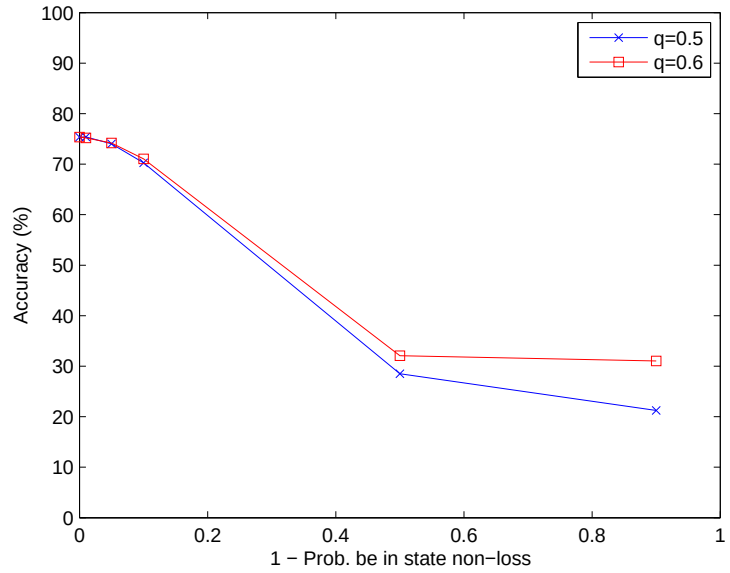


Número de bytes

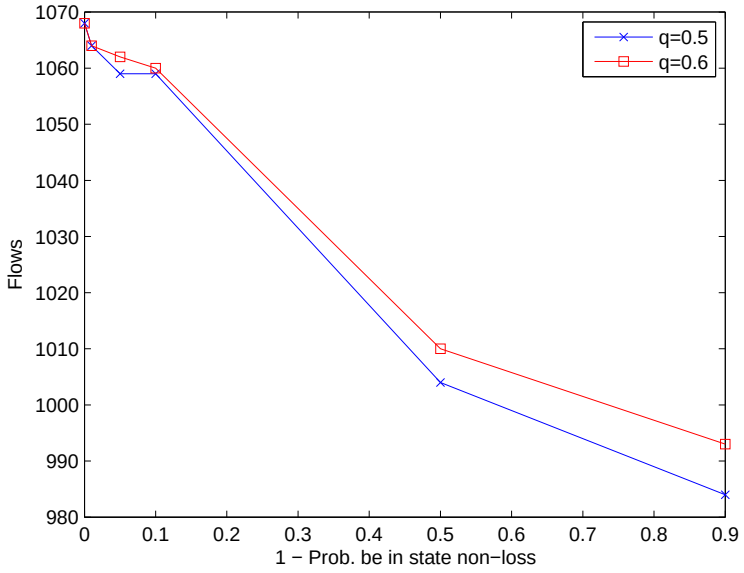
Figura 4.38: Precisión (bytes): Pérdidas Traza 3

En este caso, al tener una traza de transferencia de archivo donde no miramos el interarrival vemos que la precisión apenas varía, gráficas 4.36, 4.37 y 4.38, debido a que el tamaño del paquete tampoco lo hace, fig. 4.3; incluso pudiendo mejorar la detección a causa del aumento del número de flujos. En el caso de ver el comportamiento del detector a nivel de byte, nos acercamos a unos valores de precisión próximos al 100 %.

4.4.4. Traza 4: llamadas E2E

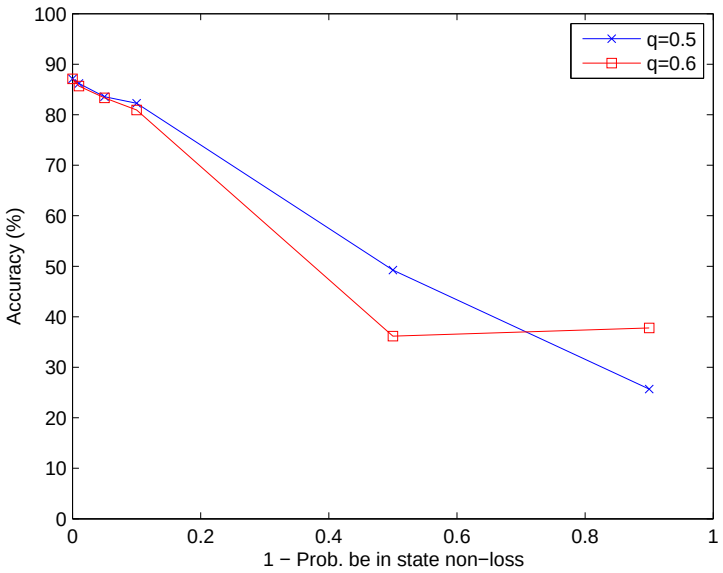


Precisión a nivel de flujo

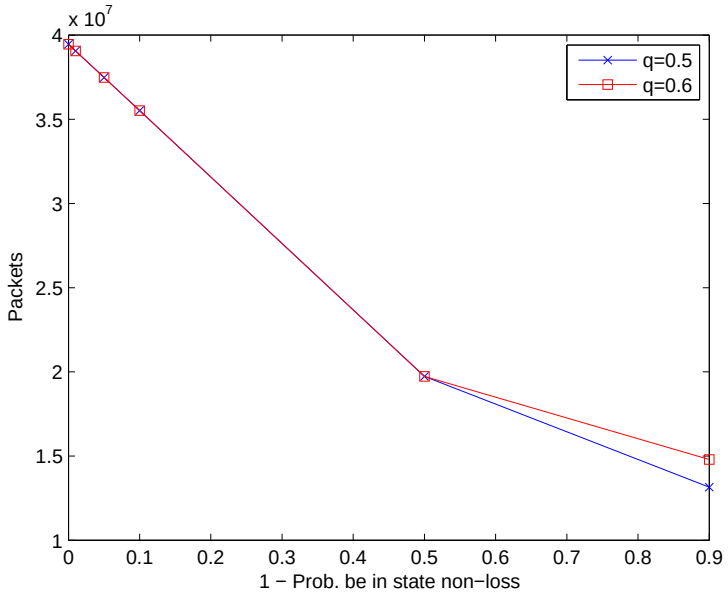


Número de flujos

Figura 4.39: Precisión (flujos): Pérdidas Traza 4

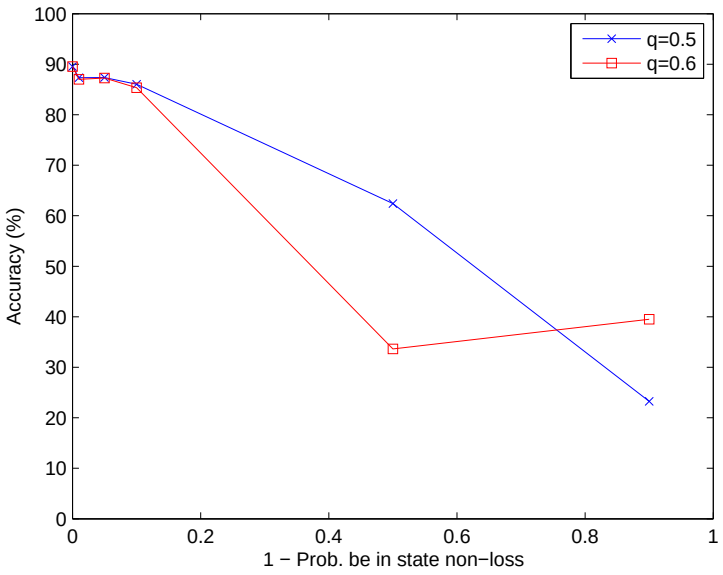


Precisión a nivel de paquete

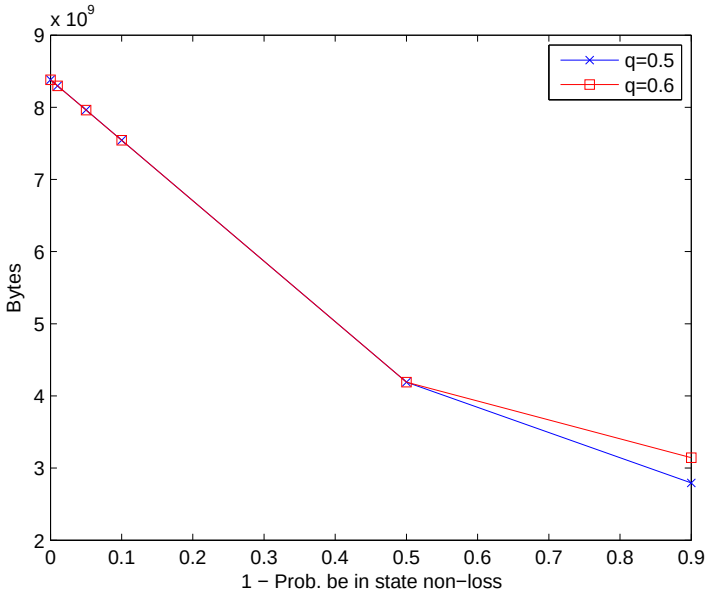


Número de paquetes

Figura 4.40: Precisión (paquetes): Pérdidas Traza 4



Precisión a nivel de bytes

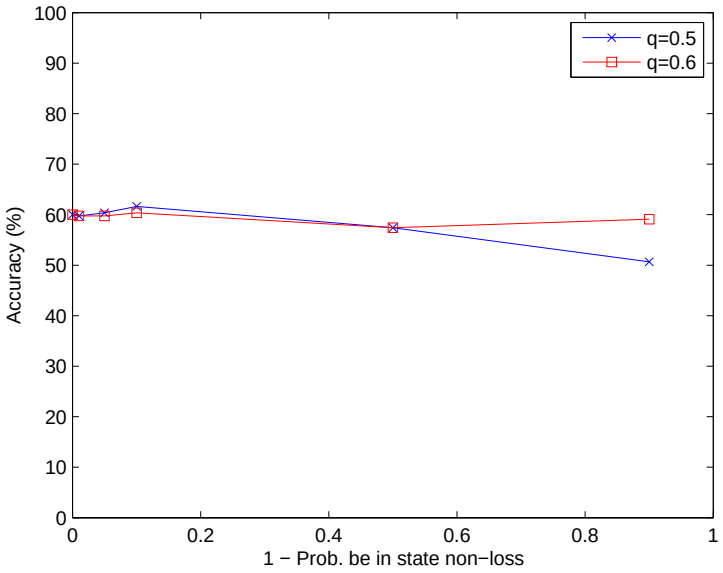


Número de bytes

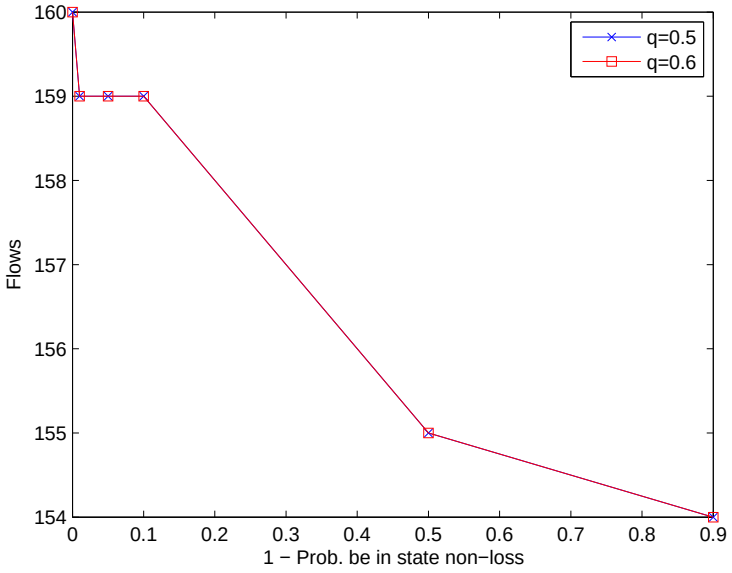
Figura 4.41: Precisión (bytes): Pérdidas Traza 4

Observamos en este experimento cambios significativos de la precisión si es a nivel de flujo, de paquete o de byte, ver figuras 4.39, 4.40 y 4.41, ya que cuando tenemos una probabilidad de estar en el estado de no pérdidas de 0.5, en el caso de la precisión a nivel de byte fig. 4.41, se obtienen mejores resultados con q=0.5 que con q=0.6, en contraste con lo que ocurre a nivel de flujo, que sería lo normal. Consideramos que este hecho es aleatorio puesto que las diferencias del tamaño de paquete y de interarrival son casi iguales para estos casos.

4.4.5. Traza 5: llamadas E2O

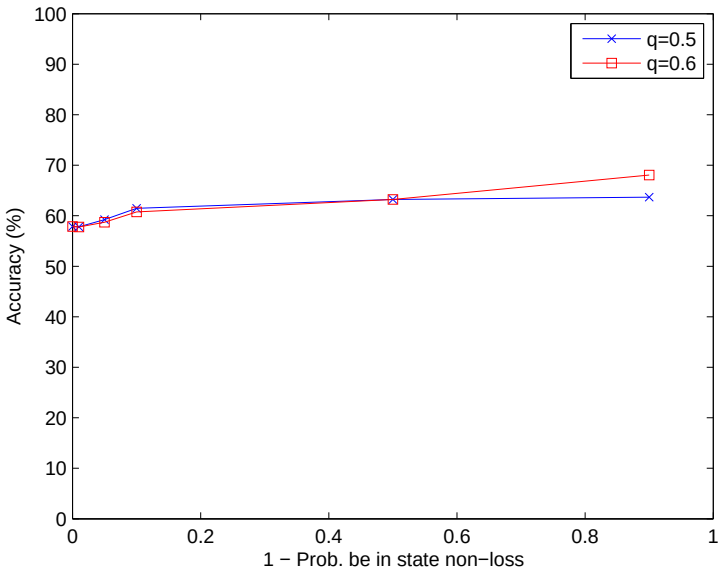


Precisión a nivel de flujo

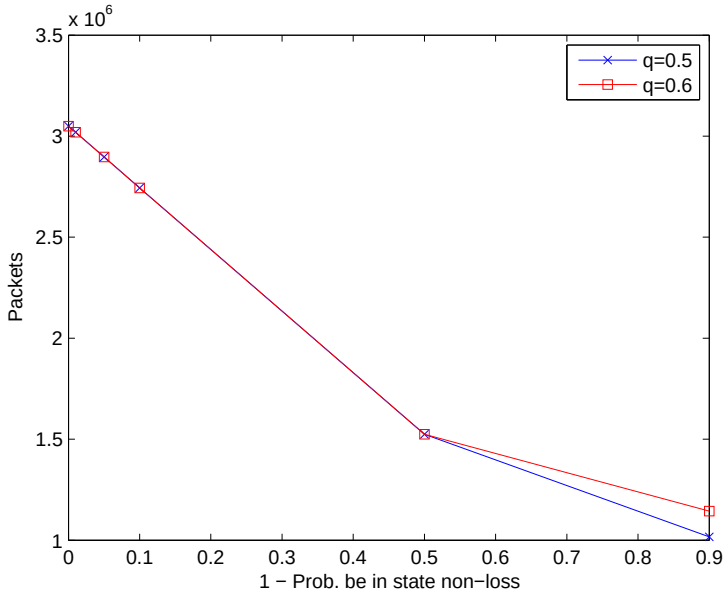


Número de flujos

Figura 4.42: Precisión (flujos): Pérdidas Traza 5

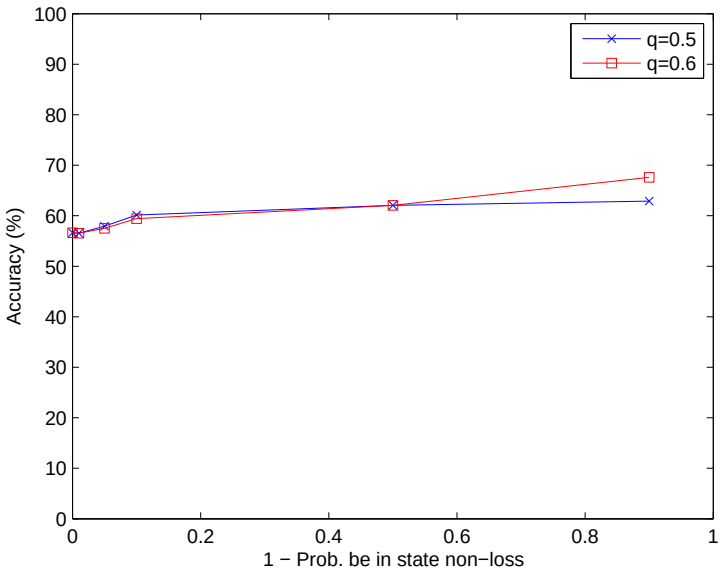


Precisión a nivel de paquete

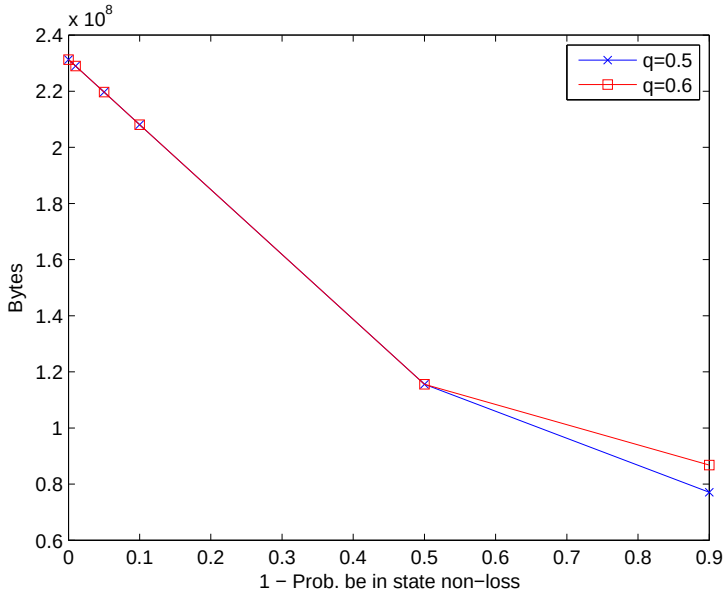


Número de paquetes

Figura 4.43: Precisión (paquetes): Pérdidas Traza 5



Precisión a nivel de bytes

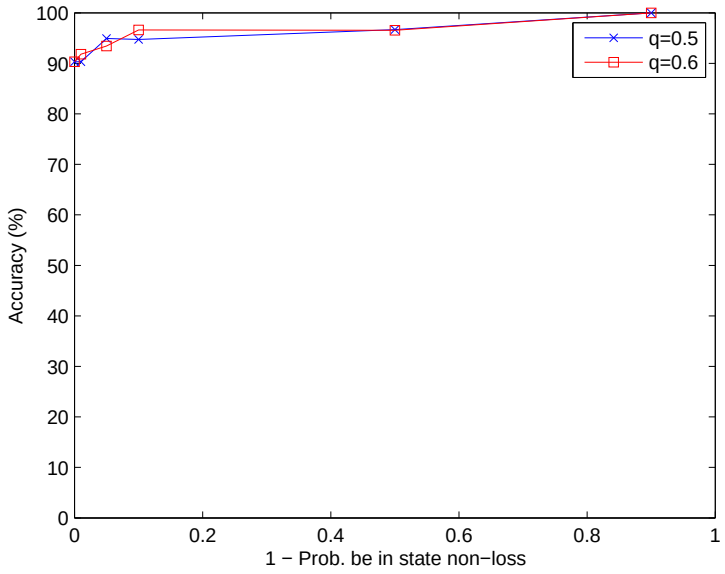


Número de bytes

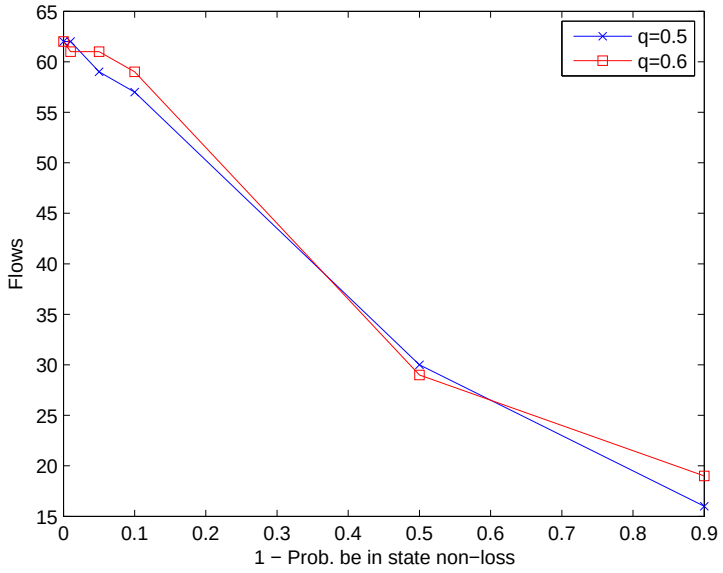
Figura 4.44: Precisión (bytes): Pérdidas Traza 5

Para esta traza la precisión del detector apenas varía, figuras 4.30 y 4.31, e incluso llega a mejorar muy ligeramente en algunos casos, precisión a nivel de byte fig. 4.32, debido a que las características estadísticas son muy similares para estos experimentos.

4.4.6. Traza 6: tráfico No-Skype

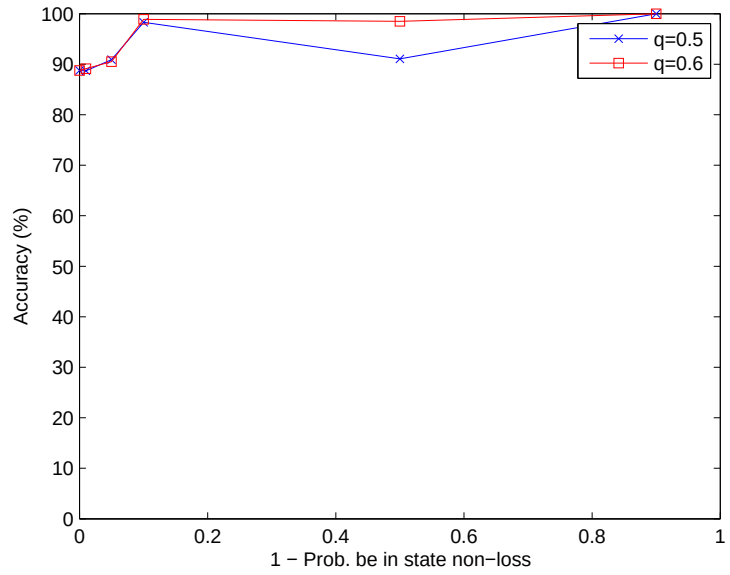


Precisión a nivel de flujo

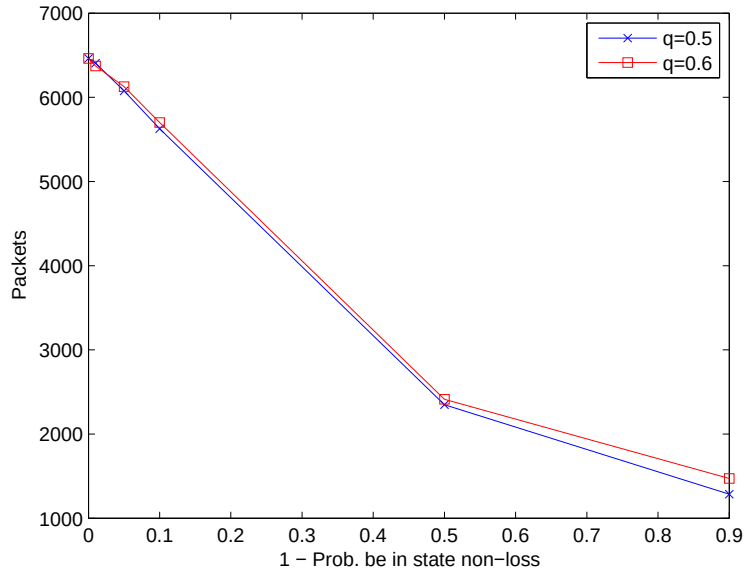


Número de flujos

Figura 4.45: Precisión (flujos): Pérdidas Traza 6

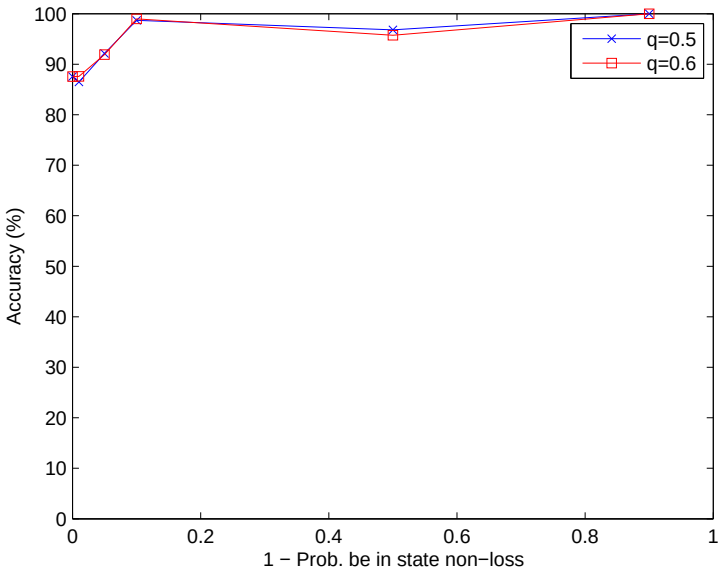


Precisión a nivel de paquete

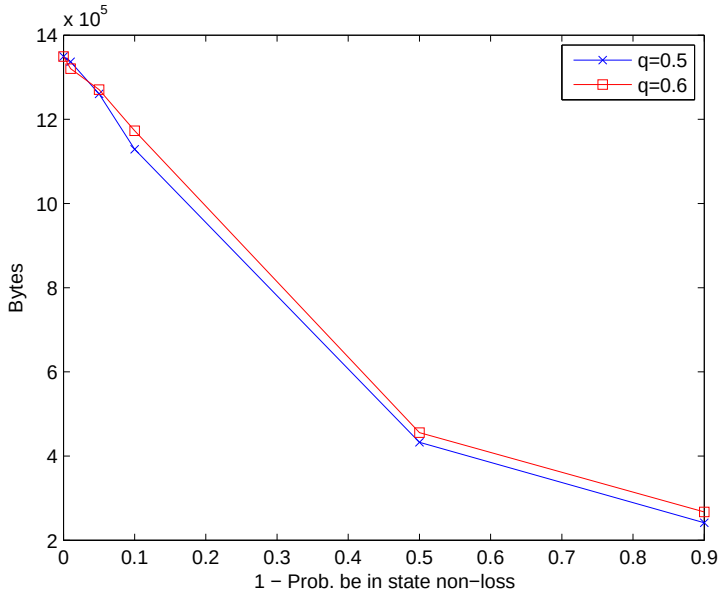


Número de paquetes

Figura 4.46: Precisión (paquetes): Pérdidas Traza 6



Precisión a nivel de bytes



Número de bytes

Figura 4.47: Precisión (bytes): Pérdidas Traza 6

La precisión aumenta según perdemos paquetes, como se aprecia en las figuras 4.45, 4.46 y 4.47, debido a que el interarrival se va alejando cada vez más del umbral necesario para clasificar el tráfico como Skype, como vemos en el análisis estadístico en la gráfica 4.11.

5

Mejora del detector

Después de haber analizado los experimentos mostrados en el capitulo anterior, cap. 4, se puede llegar fácilmente a la conclusión de que el factor que hace que el detector empeore es el interarrival, y que al tamaño de los paquetes no le afecta el muestreo y las pérdidas.

Debido a esto hemos modificado el código de Skypeness, multiplicando el interarrival por la tasa de muestreo aplicada. En el caso de simular las pérdidas de paquetes se ha calculado dicha tasa dividiendo el número de paquetes totales entre los no descartados.

En la tabla 5.1 mostramos el inverso de la tasa de muestreo para el caso en el que se produzcan pérdidas en función de la probabilidad de estar en el estado de no pérdidas y de la probabilidad de pasar del estado de pérdidas a no pérdidas, es decir, el factor por el que hay que multiplicar el interarrival para aplicar la mejora.

probabilidad de estar en no-pérdidas	**q=0.5**	**q=0.6**
0.99	1/1.0105	1/1.0106
0.95	1/1.0496	1/1.0486
0.9	1/1.1140	1/1.1089
0.5	1/2.0222	1/2.0015
0.1	1/2.9934	1/2.6502

Tabla 5.1: Tasa de muestreo equivalente para el modelo de pérdidas

5.1. Comparativa

Una vez hemos realizado dicho cambio pasamos a realizar los mismos experimentos para ver si la precisión mejora con respecto a los resultados del capítulo anterior. Se toman las mismas trazas salvo la traza 3 que no se toma debido a que no tiene en cuenta el interarrival y no tiene sentido aplicarle la mejora.

A continuación, mostramos dos tablas explicativas donde se ve numéricamente las diferencias de precisión, a nivel de byte, entre el detector sin mejora y con mejora, en caso de que se haya aplicado el muestreo o simulado pérdidas. Para la tabla sobre el muestreo presentamos los datos

para las tres diferentes políticas y para unas tasas de muestreo de 1/8, 1/64 y 1/128, menos para la traza No-Skype que hemos utilizado tasas de 1/2, 1/4 y 1/8 debido a que a tasas más altas no se detectan flujos. Y para la tabla de pérdidas mostramos la comparativa para q=0.5 y q=0.6, y probabilidades de estar en no-pérdidas de 0.99, 0.5 y 0.1. En cursiva son los valores de la precisión del detector con la mejora.

Traza	**Sin pérdidas**	**q=0.5**			**q=0.6**		
		0.99	**0.5**	**0.1**	**0.99**	**0.5**	**0.1**
Traza 1	100	97.74	26.55	8.74	100	24	15.65
		97.74	*68.86*	*66.42*	*100*	*64.25*	*68.30*
Traza 2	81.38	82.12	26.06	31.88	90.05	35.89	35.68
		82.12	*74.39*	*88.82*	*90.05*	*85.97*	*87.65*
Traza 3	95.83	95.83	95.95	96.37	95.83	95.95	96.37
Traza 4	89.55	86.15	62.42	23.24	86.99	33.62	39.49
		87.05	*89.90*	*93.14*	*87.05*	*89.90*	*91.31*
Traza 5	56.64	56.55	62.04	62.90	56.55	62.04	67.59
		56.55	*57.44*	*68.33*	*56.55*	*63.12*	*69.36*
Traza 6	87.56	86.47	96.82	100	87.59	95.73	100
		92.14	*96.82*	*55.48*	*92.14*	*92.75*	*95.92*

Tabla 5.2: Precisión(% de bytes) para pérdidas: Mejora vs Sin Mejora

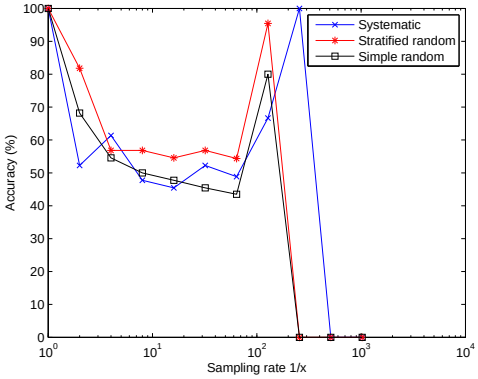
Rápidamente se observa en la tabla 5.3 como se produce un aumento considerable de la precisión, en casi todos los casos llegando a tasas de acierto del 90 %; salvo cuando tenemos tráfico No-Skype, donde el número de verdaderos negativos disminuye levemente. Dicho aumento de aciertos se hace más patente cuando muestreamos con tasas altas pasando de una precisión del 0 % a una del 90 % como en el caso de la traza 5. Para las pérdidas, tabla 5.2, ocurre algo similar ya que se consigue aumentar la precisión bastante para la mayoría de los casos, con la excepción de la traza que no contiene Skype, que empeora.

Vamos a analizar mejor estas variaciones observando gráficamente la comparativa de la precisión para las diferentes trazas (mejora vs sin mejora) y explicando lo que sucede en cada experimento.

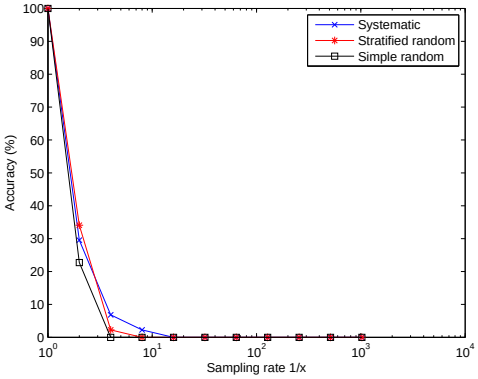
Traza	**Sin muestreo**	**Determinista**			**Aleatorio estratificado**			**Aleatorio simple**		
		1/8	**1/64**	**1/128**	**1/8**	**1/64**	**1/128**	**1/8**	**1/64**	**1/128**
Traza 1	100	2.41	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		54.51	*56.20*	*72.19*	*63.02*	*63.99*	*94.75*	*56.35*	*58.30*	*78.49*
Traza 2	81.38	5.96	0.00	2.68	6.05	1.40	10.40	0.00	0.85	2.76
		84.48	*81.97*	*82.75*	*86.16*	*91.51*	*82.70*	*88.73*	*86.55*	*70.82*
Traza 3	95.83	96.24	94.76	96.09	96.29	95.29	94.99	95.98	95.64	96.69
Traza 4	89.55	3.87	0.15	0.04	0.61	0.02	1.23	0.05	11.92	0.17
		90.72	*95.02*	*95.53*	*90.27*	*93.55*	*94.98*	*87.65*	*91.60*	*93.20*
Traza 5	56.64	35.24	0.54	0.00	24.66	0.00	0.00	9.07	0.00	0.00
		75.32	*85.36*	*90.86*	*73.85*	*92.36*	*96.52*	*65.14*	*71.63*	*88.92*
	Sin muestreo	**Determinista**			**Aleatorio estratificado**			**Aleatorio simple**		
		1/2	**1/4**	**1/8**	**1/2**	**1/4**	**1/8**	**1/2**	**1/4**	**1/8**
Traza 6	87.56	100	100	100	98.87	100	100	100	100	100
		83.00	*95.67*	*100*	*97.19*	*95.53*	*79.92*	*77.26*	*68.04*	*76.85*

Tabla 5.3: Precisión(% de bytes) para el muestreo: Mejora vs Sin Mejora

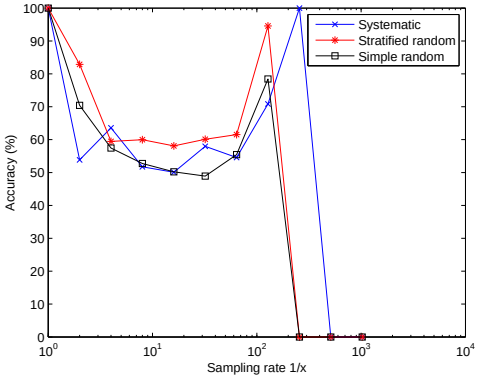
5.1.1. Traza 1: audio



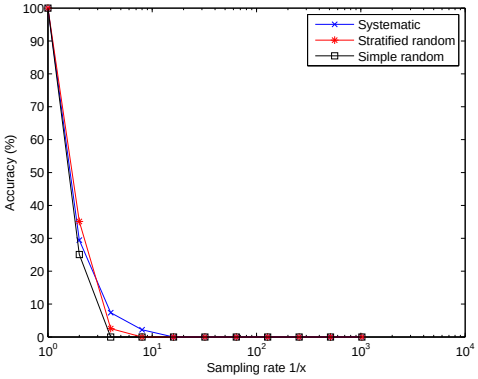
Mejora(flujo)



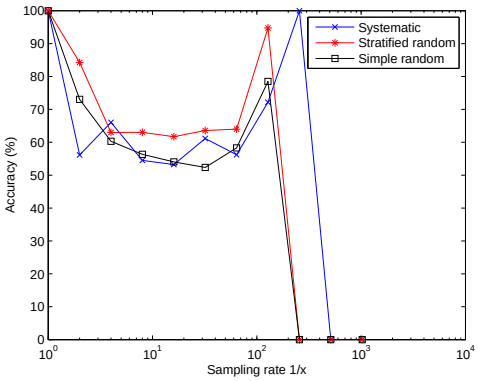
Sin mejora(flujo)



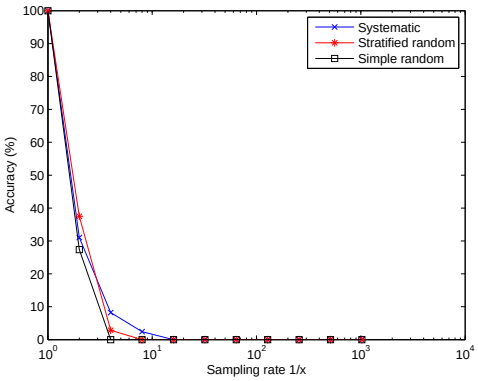
Mejora(paquete)



Sin mejora(paquete)

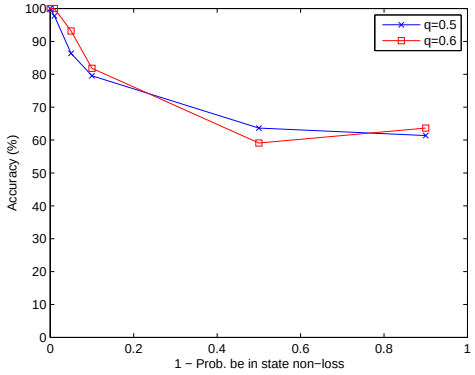


Mejora(byte)

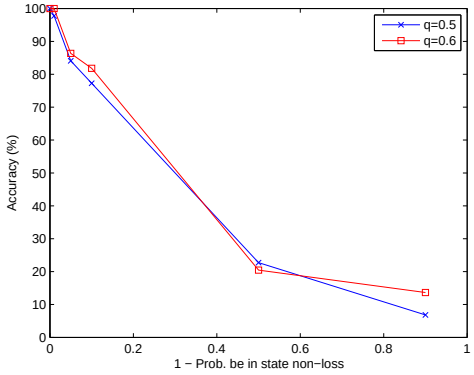


Sin mejora(byte)

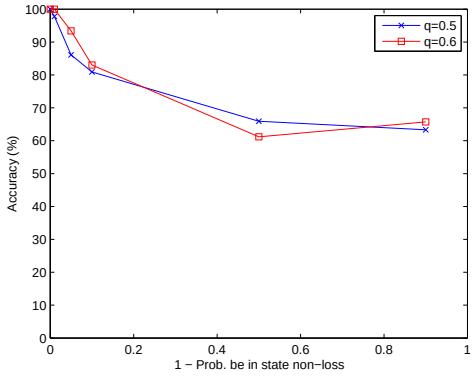
Figura 5.1: Precisión (mejora vs sin mejora): Muestreo Traza 1



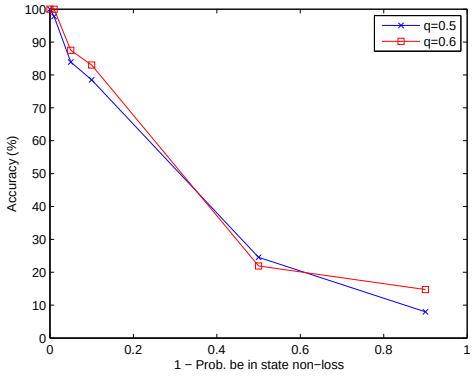
Mejora(flujo)



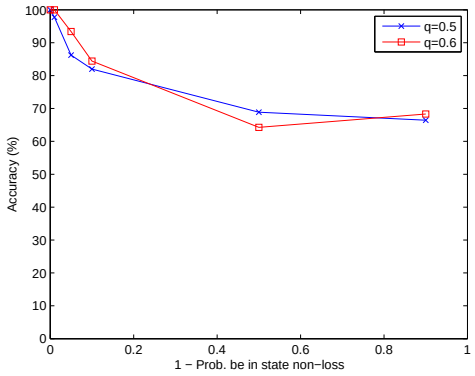
Sin mejora(flujo)



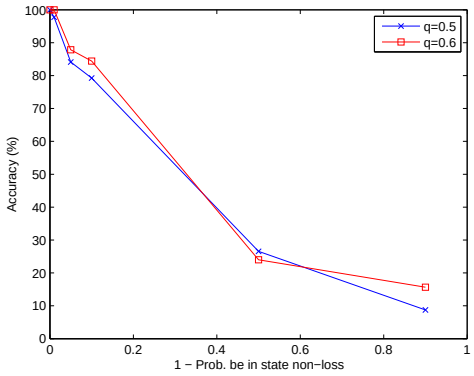
Mejora(paquete)



Sin mejora(paquete)



Mejora(byte)

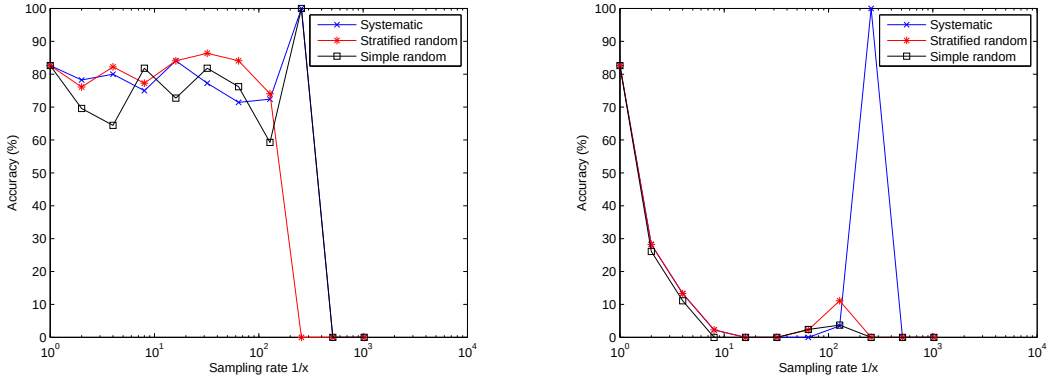


Sin mejora(byte)

Figura 5.2: Precisión (mejora vs sin mejora): Pérdidas Traza 1

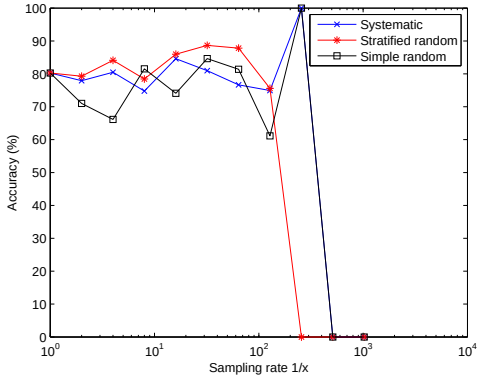
Observamos cómo, tanto para el muestreo, fig. 5.1, como cuando hay presencia de pérdidas, fig. 5.2, hay un aumento significativo de la precisión del detector en comparación con el detector sin cambios, para esta traza de audio. Incluso para tasas de muestreo altas la precisión mejora debido al menor número de flujos.

5.1.2. Traza 2: video

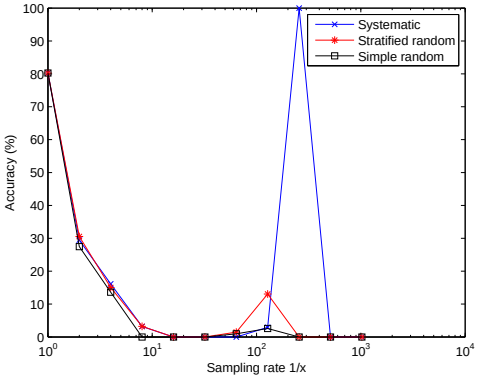


Mejora(flujo)

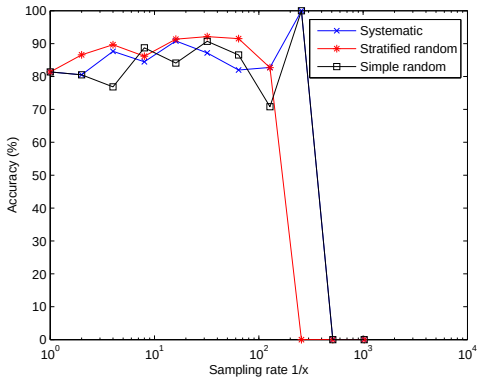
Sin mejora(flujo)



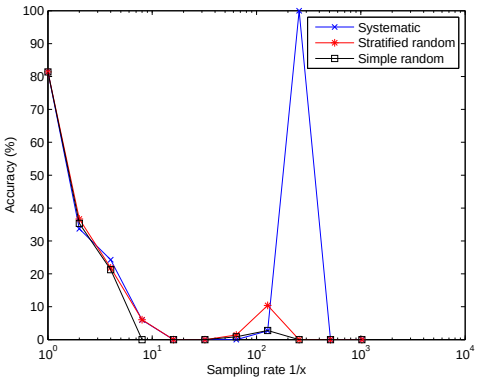
Mejora(paquete)



Sin mejora(paquete)

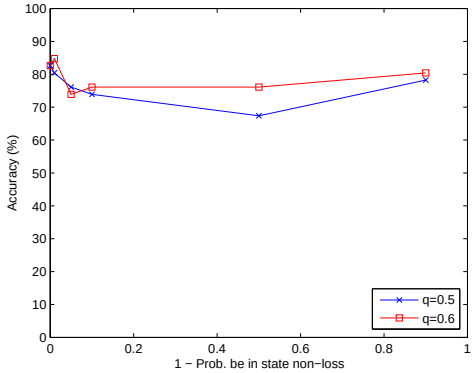


Mejora(byte)

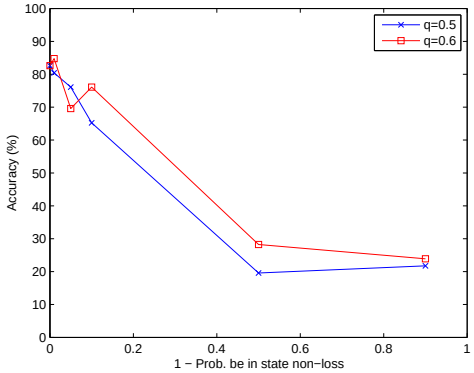


Sin mejora(byte)

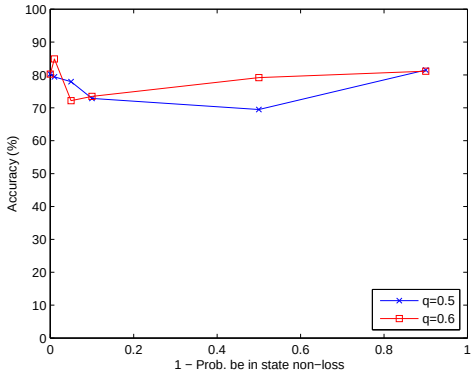
Figura 5.3: Precisión (mejora vs sin mejora): Muestreo Traza 2



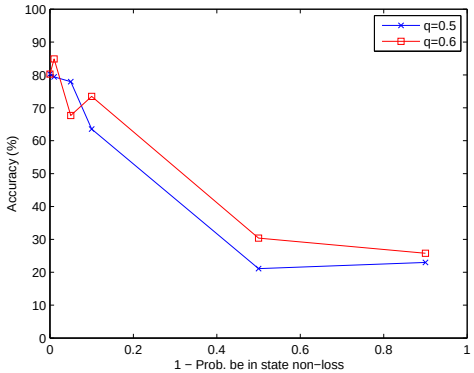
Mejora(flujo)



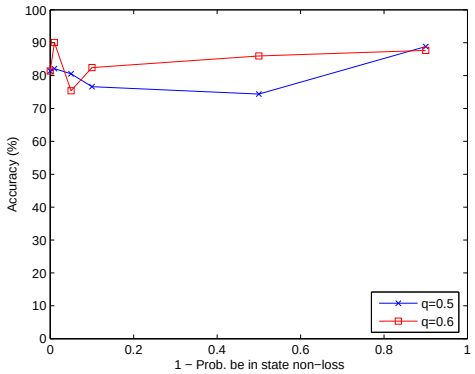
Sin mejora(flujo)



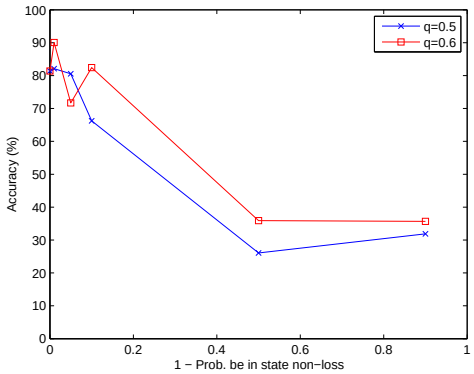
Mejora(paquete)



Sin mejora(paquete)



Mejora(byte)

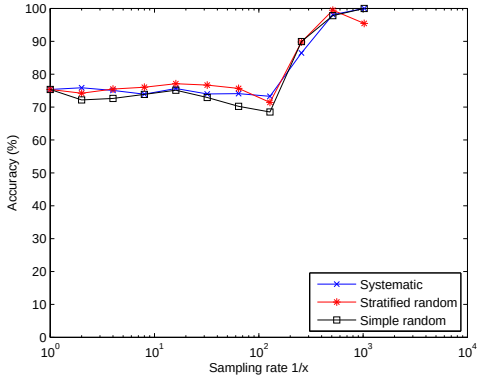


Sin mejora(byte)

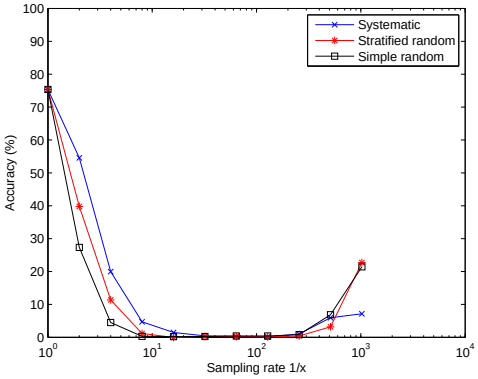
Figura 5.4: Precisión (mejora vs sin mejora): Pérdidas Traza 2

Se repite los mismos resultados que para la traza anterior, mejora significativa del acierto del detector, gráficas 5.3 y 5.4, alcanzando una tasa de aciertos del 100 % para una tasa de muestreo de 1/256.

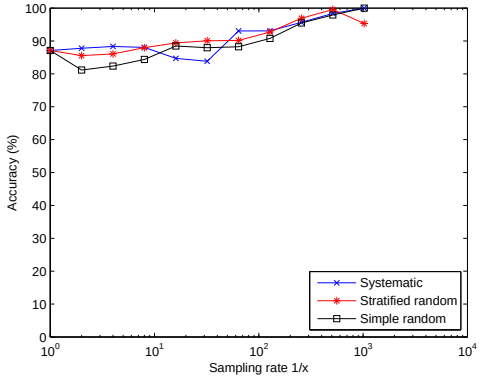
5.1.3. Traza 4: llamadas E2E



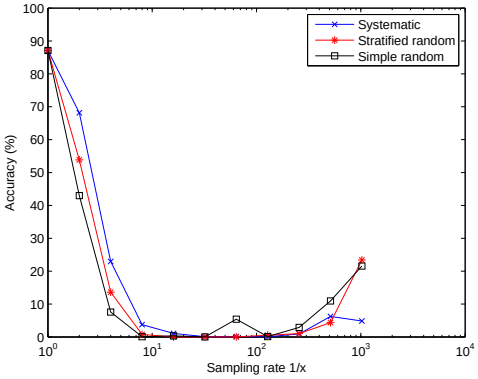
Mejora(flujo)



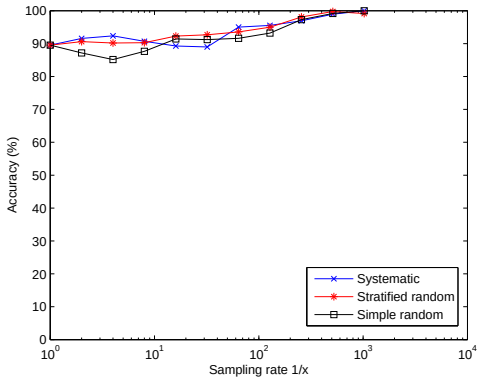
Sin mejora(flujo)



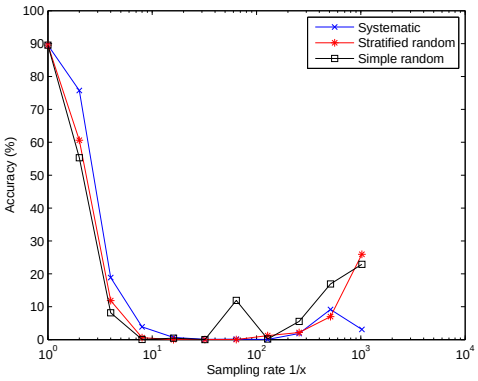
Mejora(paquete)



Sin mejora(paquete)

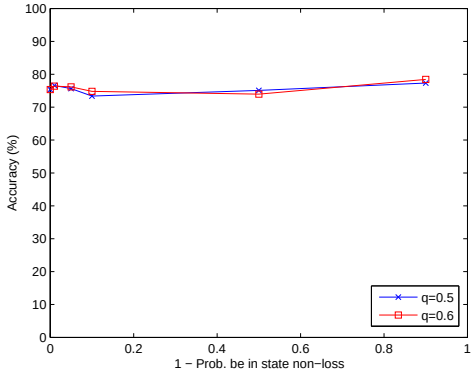


Mejora(byte)

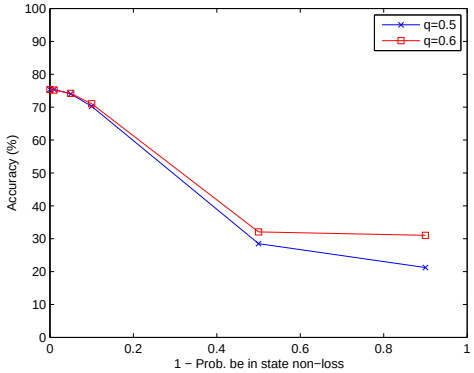


Sin mejora(byte)

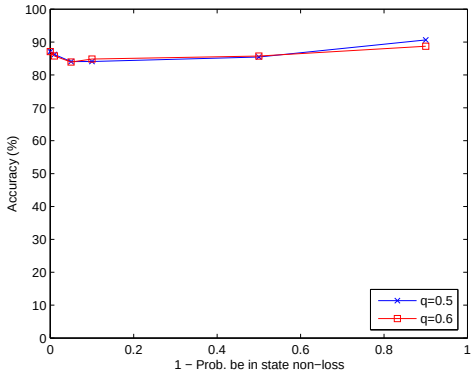
Figura 5.5: Precisión (mejora vs sin mejora): Muestreo Traza 4



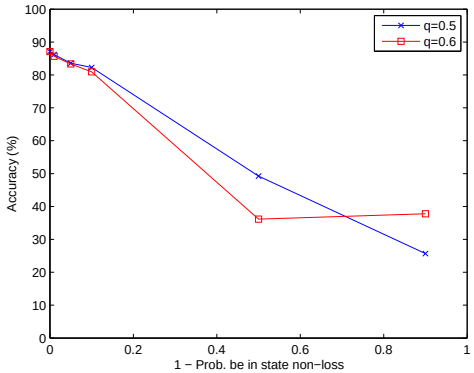
Mejora(flujo)



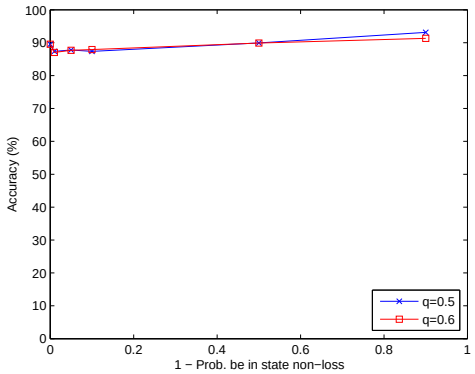
Sin mejora(flujo)



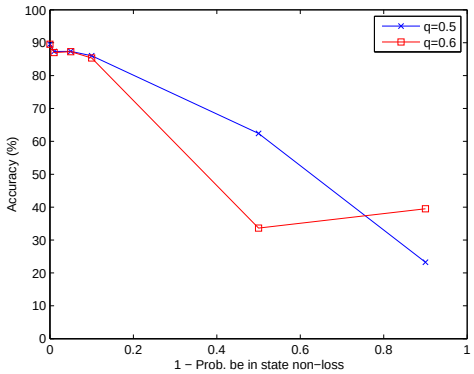
Mejora(paquete)



Sin mejora(paquete)



Mejora(byte)

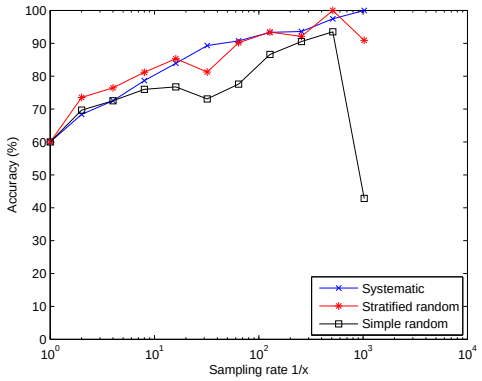


Sin mejora(byte)

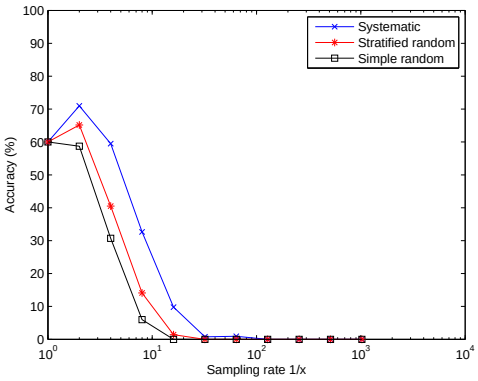
Figura 5.6: Precisión (mejora vs sin mejora): Pérdidas Traza 4

En este caso, aparte de conseguir mayores aciertos al clasificar el tráfico como Skype con respecto a la herramienta sin mejora, se consigue que la precisión aumente según perdemos paquetes, figuras 5.5 y 5.6, posiblemente debido al menor número de flujos.

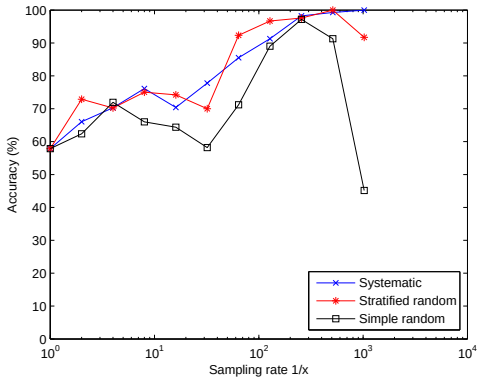
5.1.4. Traza 5: llamadas E2O



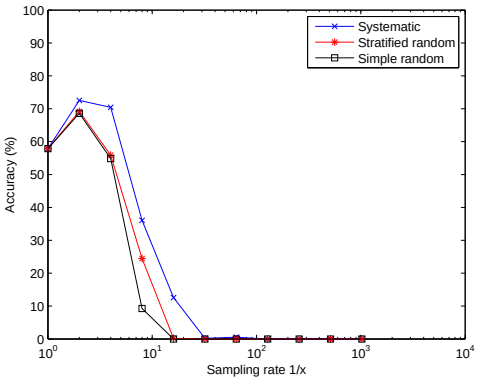
Mejora(flujo)



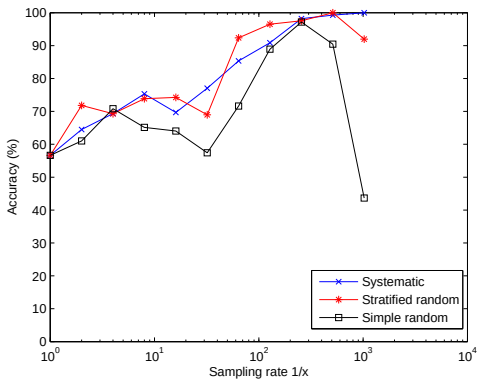
Sin mejora(flujo)



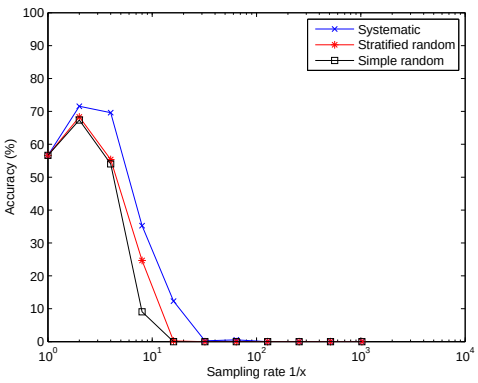
Mejora(paquete)



Sin mejora(paquete)

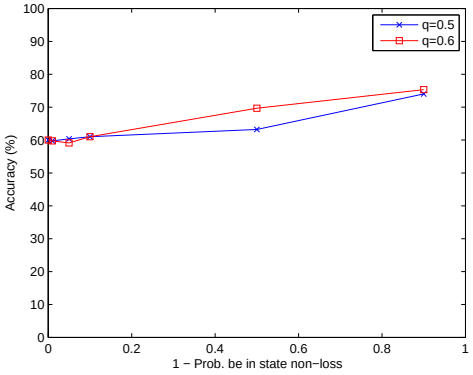


Mejora(byte)

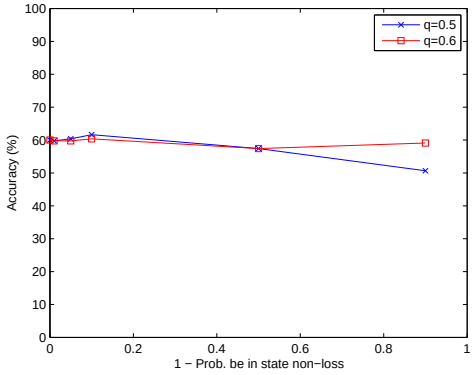


Sin mejora(byte)

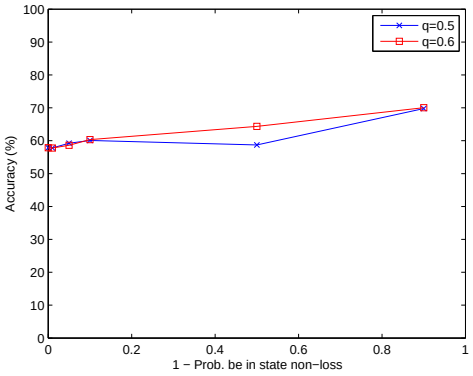
Figura 5.7: Precisión (mejora vs sin mejora): Muestreo Traza 5



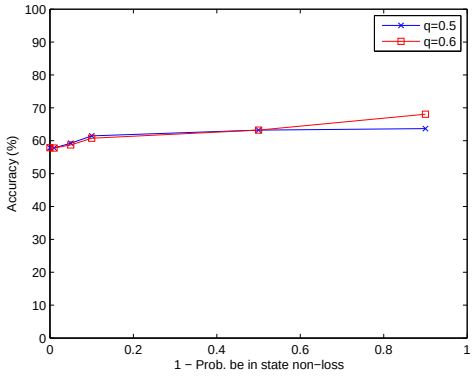
Mejora(flujo)



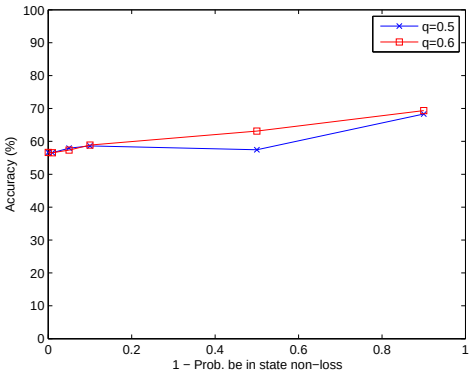
Sin mejora(flujo)



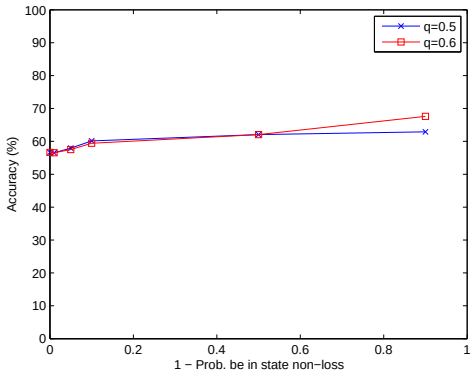
Mejora(paquete)



Sin mejora(paquete)



Mejora(byte)

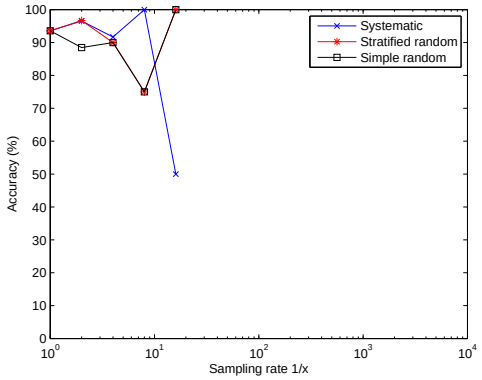


Sin mejora(byte)

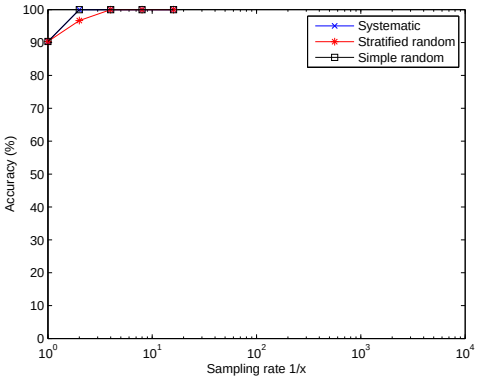
Figura 5.8: Precisión (mejora vs sin mejora): Pérdidas Traza 5

En el muestreo los resultados con claramente mejores, fig. 5.7, y en la presencia de pérdidas la precisión es similar o algo mejor para el detector con la mejora aplicada, fig. 5.8.

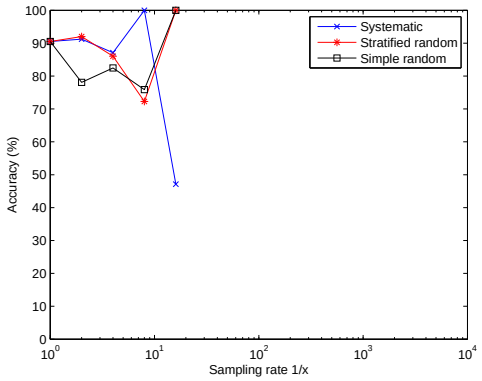
5.1.5. Traza 6: tráfico No-Skype



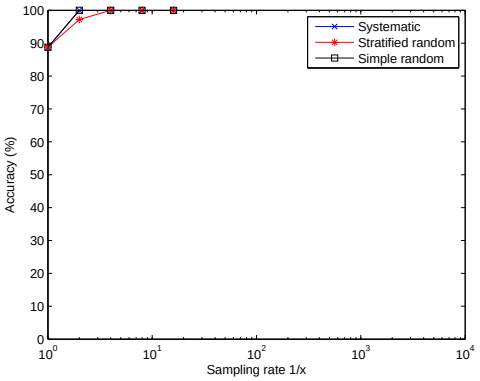
Mejora(flujo)



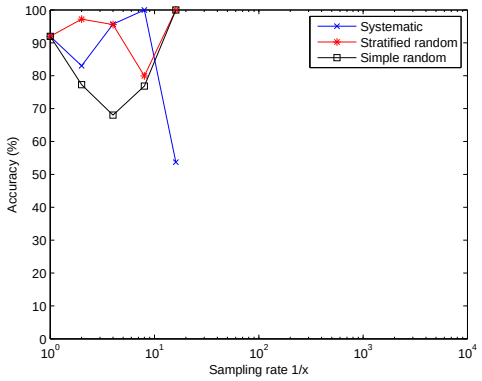
Sin mejora(flujo)



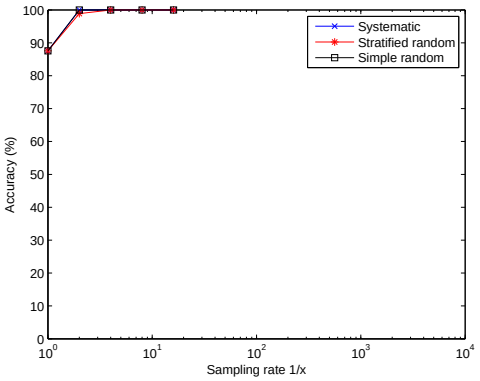
Mejora(paquete)



Sin mejora(paquete)

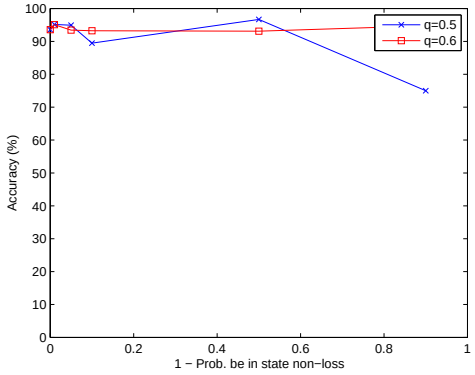


Mejora(byte)

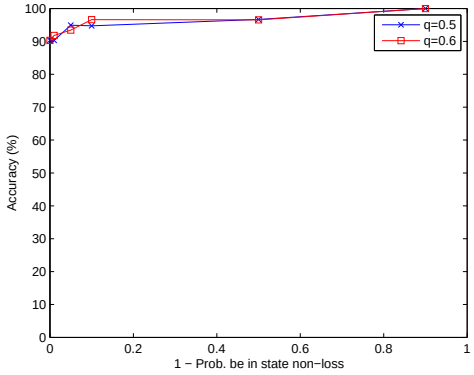


Sin mejora(byte)

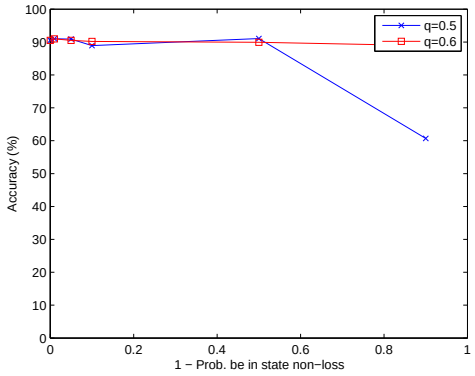
Figura 5.9: Precisión (mejora vs sin mejora): Muestreo Traza 6



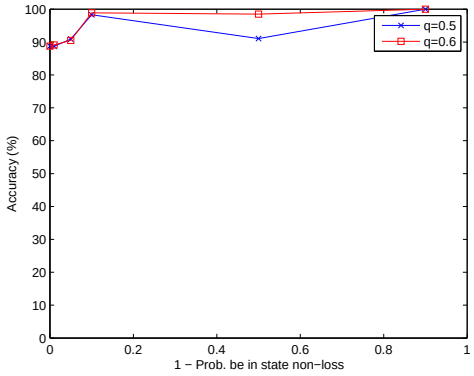
Mejora(flujo)



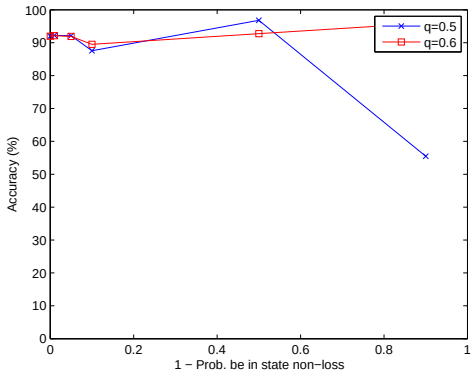
Sin mejora(flujo)



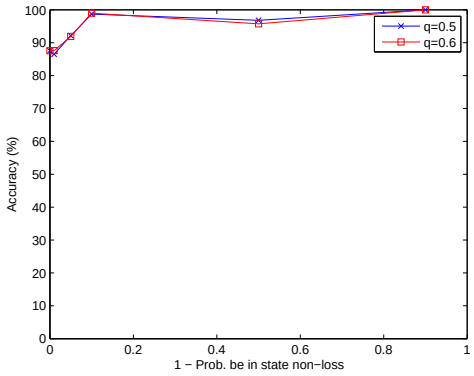
Mejora(paquete)



Sin mejora(paquete)



Mejora(byte)



Sin mejora(byte)

Figura 5.10: Precisión (mejora vs sin mejora): Pérdidas Traza 6

Este es el único caso, donde el detector empeora su precisión, aunque en muy pequeña medida, tanto en presencia de muestreo como de pérdidas, ver gráficas 5.9 y 5.10, debido a que ahora al aumentar el interarrival se acerca más al umbral por el cual caracterizamos el tráfico como Skype. Aún perdiendo precisión los resultados siguen siendo óptimos ya que la precisión está entre el 70 % y el 90 %.

Después de haber realizado la comparativa, se observa que al realizar la mejora la degradación en la precisión es mucho menor para las trazas que solo contienen tráfico Skype, es decir, la precisión del detector ha aumentado notablemente tanto cuando se le aplica muestreo como si existen pérdidas. Incluso a tasas de muestreo altas el detector tiene una buena precisión. En el único caso en que se ha empeorado el número de aciertos ha sido para la traza que no contiene tráfico Skype, donde ha habido un aumento de falsos positivos, debido a que al realizar la multiplicación ahora el interarrival se encuentra más cerca de estar en los intervalos definidos para el tráfico Skype.

Se observa que la mejora afecta indistintamente tanto si la traza es de audio como de vídeo, o si la llamada es E2E o E2O. En cuanto al tipo de muestreo utilizado no se extrae una conclusión clara sobre qué política es más favorable, aunque si podemos decir que en la mayor parte de los casos se obtienen mejores resultados si se aplica un muestreo determinista o aleatorio estratificado, ya que la media del interarrival en estos casos es menor que si aplica un muestreo aleatorio simple, como hemos visto en el análisis estadístico, debido a que los paquetes pueden estar más distanciados entre sí. En cambio para el muestreo determinista siempre se seleccionan con la misma separación.

También nos fijamos que con la mejora introducida en el detector según aumentamos la tasa de muestreo la precisión se suele mantener más o menos entorno a los mismos resultados y para tasas altas la precisión aumenta, este hecho es debido a que se reduce el número de flujos considerablemente haciendo que el número de aciertos tenga mayor peso. Sin la mejora lo que ocurre es que a medida que se aumenta la tasa de muestreo la precisión cada vez era peor por regla general.

Si estamos en una situación de pérdidas vemos que la precisión suele ser más o menos constante, incluso llegando a mejorar cuando hay mayor probabilidad de perder paquetes. También mejora en comparación con el capitulo anterior. Por regla general, se obtienen mejores resultados cuando la probabilidad de perder paquetes es menor.

6

Conclusiones y trabajo futuro

6.1. Conclusiones

Como principales conclusiones y hallazgos de este proyecto fin de carrera se pueden destacar:

- El análisis estadístico de las trazas sugiere que la distribución del tamaño del paquete no se ve afectada por el muestreo o las pérdidas, ya que aunque se cuente con un número de menor de paquetes la media de sus tamaños tenderá a un número similar. Sin embargo, se aprecia algún cambio en la media, aunque poco significativo, cuando tenemos un número muy reducido de paquetes, por ejemplo, con una tasa de muestreo de 1/1024, sin afectar a la precisión del detector.
- El análisis estadístico de las trazas sugiere que la distribución del interarrival sufre una severa modificación causada por el muestreo o las pérdidas, ya que la ECDF de esta característica muestra que según descartamos paquetes el interarrival va disminuyendo considerablemente.
- Tras la evaluación de la precisión del detector cuando se aplica muestreo o pérdida de paquetes, se puede concluir que ésta se ve degradada debido a la modificación del interarrival, puesto que según se aumenta la tasa de muestreo o la probabilidad de perder paquetes, esta característica estadística se va alejando del umbral necesario para considerar que el tráfico es Skype.
- Se propone e implementa una simple modificación del detector, que corrige el deterioro del interarrival. Dicha modificación consiste en multiplicar el interarrival por la tasa de muestreo aplicada, haciendo que ahora dicha característica se acerque al nivel deseado para aumentar considerablemente la tasa de aciertos. Esto se puede ver claramente observando la comparativa que se ha hecho en el capitulo anterior.
- Por tanto, una vez analizados todos los resultados obtenidos y de haber realizado los cambios anteriormente contados en el detector Skypeness (multiplicar el interarrival por la tasa de muestreo), podemos decir que hemos alcanzado el objetivo principal de este proyecto, que era el poder ser capaz de clasificar tráfico Skype en un entorno de alta velocidad (10-40-100 Gb/s) donde se busca eficiencia y rapidez; reduciendo el número

de paquetes analizados, ya sea por realizar un muestreo o por estar en una situación de pérdidas, para no tener una alta carga computacional que ralentizaría el proceso de clasificación, con el único inconveniente del leve aumento de los falsos positivos y una ligera disminución de la precisión.

- Cabe destacar que, como contribución de este proyecto fin de carrera, parte del trabajo aquí expuesto ha sido enviado a la conferencia internacional IFIP/IEEE *Integrated Network Management Symposium* (IM 2013) con proceso de selección y revisión de artículos *peer review* (incluido en el anexo D). Dicho artículo, ***"On the Impact of Packet Sampling on Skype Traffic Classification"***[28], ha sido aceptado para su presentación en la conferencia.

- Además de tal publicación, otra contribución adicional de este proyecto a la comunidad científica, ha sido el código de los emuladores de muestreo y pérdidas y de las trazas utilizadas para la evaluación del detector. Todo ello está disponible como código abierto en: `http://www.eps.uam.es/~psantiago/skypeness.html`.

6.2. Trabajo futuro

Hemos visto como se han alcanzado los objetivos que nos proponíamos al iniciar este proyecto, siendo capaces de clasificar el tráfico Skype a altas velocidades, mediante la herramienta Skypeness. En este proyecto nos hemos centrado en detectar el tráfico UDP pero no el TCP, por tanto, una posible mejora sería también realizar este mismo estudio para TCP.

Otro posible trabajo para el futuro podría ser el centrarse en otras herramientas de tráfico Skype, como las vistas en el estado del arte, las cuáles caracterizan el tráfico de forma diferente, por ejemplo, mediante los mensajes que se intercambian entre clientes Skype, y entre el cliente y los supernodos, y se consiga que también funcione el detector en un entorno de alta velocidad.

Con miras al futuro se podría investigar otros métodos para la mejora de la precisión para el detector Skypeness, que aunque se ha corroborado que nuestro cambio es óptimo, quizás exista otro método que pueda ser más favorable y por tanto que pueda mejorar la precisión del detector en el caso de que se realice un muestreo o que existan pérdidas.

Además también se podría estudiar el aplicar esta misma metodología para detectores de otro tipo de tráfico, por ejemplo, el generado por otras aplicaciones P2P como emule o bittorrent.

Glosario de acrónimos

- **ACK**: ACKNOWLEDGEMENT
- **DFA**: Determinist Finite Automata
- **DPI**: Deep Packet Inspection
- **E2E**: End to End
- **E2O**: End to Out
- **ECDF**: Empirical Cumulative Distribution Function
- **FEC**: Forward Error Correction
- **HTTP**: Hypertext Transfer Protocol
- **IP**: Internet Protocol
- **ISP**: Internet Service Provider
- **QoE**: Quality of Experience
- **QoS**: Quality of Service
- **LAN**: Local Area Network
- **MBFS**: Message Based per-Flow State
- **NAT**: Network Address Translation
- **P2P**: Peer to Peer
- **PBFS**: Packet Based per-Flow State
- **RTCP**: Real-time Control Protocol
- **RTP**: Real-time Transport Protocol
- **SVOPC**: Sinusoidal Voice Over Packet Coder
- **SoM**: Start of Message
- **TCP**: Transmission Control Protocol
- **UDP**: User Datagram Protocol
- **VoIP**: Voice over Internet Protocol

Bibliografía

[1] Thuy T.T. Nguyen and Grenville Armitege. A survey of techniques for internet traffic classification using machine learning. *IEE Communications Surveys and Tutorials*, 10(4):56–76, 2008.

[2] Dario Bonfiglio, Marco Melia, Michela Meo, and Dario Rossi. Detailed analysis of skype traffic. *IEE Trans Multimed.*, 11(1), 2009.

[3] Sándor Molnár and Marcell Perényi. On the identification and analysis of skype traffic. *Internacional Journal of Communications Systems*, 24(1), April, 2011.

[4] Niccolò Cascarano, Luigi Ciminiera, and Fulvio Risso. Optimizing deep packet inspection for high-speed traffic analysis. *J. Netw. Syst. Manage.*, 19(1):7–31, 2011.

[5] P.M. Santiago del Río, J. Ramos, J.L. García-Dorado, J. Aracil, A. Cuadra-Sánchez, and M. Cutanda-Rodríguez. On the processing time for detection of skype traffic. *2nd International Workshop on Traffic Analysis and Classification (IWCMC2001-TRAC)*, July, 2011.

[6] P. A. Branch, A. Heyde, and G. J. Armitage. Rapid identification of skype traffic flows. *ACM NOSSDAV*, 2011.

[7] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. *SIGCOMM Comput. Commun. Rev.*, 37(4):37–48, 2007.

[8] B. Trammell, E. Boschi, G. Procissi, C. Callegari, P. Dorfinger, and D. Schatzmann. Identifying skype traffic in a large-scale flow data repository. *TMA LNCS*, 2011.

[9] D. Adami, C. Callegari, S. Giordano, M. Pagano, and T. Pepe. A real-time algorithm for skype traffic detection and classification. *NEW2AN/ruSMART LNCS*, 2009.

[10] Davide Adami, Christian Callegari, Stefano Giordano, Michele Pagano, and Teresa Pepe. Skype-hunter: A real-time system for the detection and classification of skype traffic. *Internacional Journal of Communications Systems*, 2011.

[11] Kimberly C. Claffy and George C. Polyzos. Application of sampling methodologies to network traffic characterization. *SIGCOMM*, 1993.

[12] A. Callado, Carlos Kamienski, G. Szabó, B. P. Geró, J. Kelner, S. Fernandez, and D. Sadok. A duervey on internet traffic identification. *IEEE Communications Surveys and Tutorials*, 11(3), 2009.

[13] Wenhong Ma and Changcheng Huang. Adaptive sampling for network performance measurement under voice traffic. *IEEE ICC*, 2004.

[14] G. Androulikadis, V. Chatzigiannakis, and S. Papavassiliou. Network anomaly detection and classification via opportunistic sampling. *IEEE Network*, 2009.

[15] V. Carela-Español, P.Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta. Analysis of the impact of sampling on netflow traffic classification. *Computer Networks*, 55, 2011.

[16] Davide Tammaro, Silvio Valenti, Dario Rossi, and Antonio Pescapé. Exploiting packet-sampling measurements for traffic characterization and classification. *Internacional Journal of Network Management*, 2012.

[17] Silvio Valenti and Dario Rossi. Fine-grained behavioral classification in the core: the issue of flow sampling. *2nd International Workshop on Traffic Analysis and Classification (IWCMC2011-TRAC)*, 2011.

[18] Oliver Hohlfeld, Rüdiger Geib, and Gerhard Hablinger. Packet loss in real-time services: markovian models generating qoe impairments. *In Proc. IWQoS*, 2008.

[19] Wenyu Jiang and Henning Schulzrinne. Modeling of packet loss and delay and their effect on real-time multimedia service quality. *NOSSDAV 2000*.

[20] Descargar tstat. `http://tstat.tlc.polito.it/index.shtml`. 2012.

[21] Analizador tstat. Polito. `http://tstat.tlc.polito.it/software.php`. 2012.

[22] L7-filter. `http://l7-filter.sourceforge.net/`. 2012.

[23] Netfilter. `http://netfilter.org/`. 2012.

[24] Descargar L7-filter. `http://download.clearfoundation.com/l7-filter/l7-filter-userspace-0.12-beta1.tar.gz`. 2012.

[25] Programa para la simulación del muestreo y las pérdidas. UAM. `http://arantxa.ii.uam.es/~psantiago/sampling.html`. 2012.

[26] Trazas de tráfico real de internet capturado en Polito: E2E(audio y video) y E2O. `http://tstat.tlc.polito.it/traces-skype.shtml`. 2012.

[27] Trazas capturadas en la UAM. `http://arantxa.ii.uam.es/~psantiago/skypeTraces.html`. 2012.

[28] Pedro Santiago del Rio, Diego Corral, José Luis García Dorado, and Javier Aracil. On the impact of packet sampling on skype traffic classification. In *IEEIM 2013 - TechSessions ()*, 2013.

A

Presupuesto

1)	**Ejecución Material**	
	▪ Compra de ordenador personal (Software incluido)	2.000 €
	▪ Alquiler de impresora láser durante 6 meses	260 €
	▪ Material de oficina	150 €
	▪ Total de ejecución material	2.400 €
2)	**Gastos generales**	
	▪ sobre Ejecución Material	352 €
3)	**Beneficio Industrial**	
	▪ sobre Ejecución Material	132 €
4)	**Honorarios Proyecto**	
	▪ 1800 horas a 15 €/ hora	27000 €
5)	**Material fungible**	
	▪ Gastos de impresión	280 €
	▪ Encuadernación	200 €
6)	**Subtotal del presupuesto**	
	▪ Subtotal Presupuesto	32.774 €
7)	**I.V.A. aplicable**	
	▪ 16 % Subtotal Presupuesto	5.243,8 €
8)	**Total presupuesto**	
	▪ Total Presupuesto	38.017,8 €

Madrid, NOVIEMBRE 2012

El Ingeniero Jefe de Proyecto

Fdo.: Diego Corral González

Ingeniero Superior de Telecomunicación

B

Pliego de condiciones

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un *Evaluación de la influencia del muestreo y de la pérdida de paquetes sobre la detección de tráfico Skype*. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales.

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4 % del presupuesto y la provisional del 2 %.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrata" y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares.

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.

C

Manual del programador

C.1. Código Matlab: Calculo de la precisión del detector

```
%introducir nombre archivo .dat a leer, con las estadisticas
A=load ('file_v3_salida.dat');

%%%%%%%% MUESTREO %%%%%%%%%

%Precisión a nivel de flujo
x=2.^[0:10];
recall_1=A([1:11],13)';
recall_2=A([12:22],13)';
recall_3=A([23:33],13)';

figure(1)
semilogx(x,recall_1,'x-')
hold on
semilogx(x,recall_2,'r*-')
semilogx(x,recall_3,'ks-')
axis([0 10^4 0 100])
xlabel ('Sampling rate 1/x')
ylabel ('Accuracy (%)')
legend ('Systematic', 'Stratified random', 'Simple random')
hold off

%numero de flujos
flows_1=A([1:11],1)';
flows_2=A([12:22],1)';
flows_3=A([23:33],1)';

figure(2)
semilogx(x,flows_1,'x-')
hold on
semilogx(x,flows_2,'r*-')
semilogx(x,flows_3,'ks-')
xlabel ('Sampling rate 1/x')
ylabel ('Flows')
```

```
legend ('Systematic', 'Stratified random', 'Simple random')
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Precisión a nivel de paquete
recallp_1=A([1:11],14)';
recallp_2=A([12:22],14)';
recallp_3=A([23:33],14)';

figure(3)
semilogx(x,recallp_1,'x-')
hold on
semilogx(x,recallp_2,'r*-')
semilogx(x,recallp_3,'ks-')
axis([0 10^4 0 100])
xlabel ('Sampling rate 1/x')
ylabel ('Accuracy (%)')
legend ('Systematic', 'Stratified random', 'Simple random')
hold off

%numero de paquetes
pack_1=A([1:11],2)';
pack_2=A([12:22],2)';
pack_3=A([23:33],2)';

figure(4)
semilogx(x,pack_1,'x-')
hold on
semilogx(x,pack_2,'r*-')
semilogx(x,pack_3,'ks-')
xlabel ('Sampling rate 1/x')
ylabel ('Packets')
legend ('Systematic', 'Stratified random', 'Simple random')
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Precisión a nivel de byte
recallb_1=A([1:11],15)';
recallb_2=A([12:22],15)';
recallb_3=A([23:33],15)';

figure(5)
semilogx(x,recallb_1,'x-')
hold on
semilogx(x,recallb_2,'r*-')
semilogx(x,recallb_3,'ks-')
axis([0 10^4 0 100])
xlabel ('Sampling rate 1/x')
ylabel ('Accuracy (%)')
legend ('Systematic', 'Stratified random', 'Simple random')
hold off

%numero de bytes
by_1=A([1:11],3)';
by_2=A([12:22],3)';
by_3=A([23:33],3)';
```

```
figure(6)
semilogx(x,by_1,'x-')
hold on
semilogx(x,by_2,'r*-')
semilogx(x,by_3,'ks-')
xlabel ('Sampling rate 1/x')
ylabel ('Bytes')
legend ('Systematic', 'Stratified random', 'Simple random')
hold off

%%%%%%%%%%% PERDIDAS %%%%%%%%%%%%

%Precisión a nivel de flujo
xl=1-[1 0.99 0.95 0.9 0.5 0.1];
recall_11=A([34:39],13)';
recall_21=A([40:45],13)';

figure(7)
plot(xl,recall_11,'x-')
hold on
plot(xl,recall_21,'rs-')
xlabel ('1 - Prob. be in state non-loss')
ylabel ('Accuracy (%)')
legend ('q=0.5', 'q=0.6')
axis([0 1 0 100])
hold off

%numero de flujos
flows_11=A([34:39],1)';
flows_21=A([40:45],1)';

figure(8)
plot(xl,flows_11,'x-')
hold on
plot(xl,flows_21,'rs-')
xlabel ('1 - Prob. be in state non-loss')
ylabel ('Flows')
legend ('q=0.5', 'q=0.6')
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Precisión a nivel de paquete
xl=1-[1 0.99 0.95 0.9 0.5 0.1];
recall_12=A([34:39],14)';
recall_22=A([40:45],14)';

figure(9)
plot(xl,recall_12,'x-')
hold on
plot(xl,recall_22,'rs-')
xlabel ('1 - Prob. be in state non-loss')
ylabel ('Accuracy (%)')
legend ('q=0.5', 'q=0.6')
axis([0 1 0 100])
hold off
```

```
%numero de paquetes
pk_11=A([34:39],2)';
pk_21=A([40:45],2)';

figure(10)
plot(xl,pk_11,'x-')
hold on
plot(xl,pk_21,'rs-')
xlabel ('1 - Prob. be in state non-loss')
ylabel ('Packets')
legend ('q=0.5', 'q=0.6')
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Precisión a nivel de bytes
xl=1-[1 0.99 0.95 0.9 0.5 0.1];
recall_13=A([34:39],15)';
recall_23=A([40:45],15)';

figure(11)
plot(xl,recall_13,'x-')
hold on
plot(xl,recall_23,'rs-')
xlabel ('1 - Prob. be in state non-loss')
ylabel ('Accuracy (%)')
legend ('q=0.5', 'q=0.6')
axis([0 1 0 100])
hold off

%numero de bytes
by_11=A([34:39],3)';
by_21=A([40:45],3)';

figure(12)
plot(xl,by_11,'x-')
hold on
plot(xl,by_21,'rs-')
xlabel ('1 - Prob. be in state non-loss')
ylabel ('Bytes')
legend ('q=0.5', 'q=0.6')
hold off
```

C.2. Código Matlab: Análisis estadístico

```
A=load('Skype_UAM_audio_caract.dat');
dA=diff(A);
B=dA(find(abs(dA(:,2))<1260000000000),:);
C=B(find(B(:,3)==0),2);          %%ELIMINA PAQUETE PRIMERO DE CADA FLUJO
C=abs(C/10^6);
s=length(C);
i=[1:s];
w=0;
l=0;
sum_int=0;
flow=A(:,6)';

for i=2:s
        if (flow(i)==flow(i-1))
            w=w+1;
            C(i-1);
            sum_int=C(i-1)+sum_int;
            if (w==10)
                l=l+1;
                mean_int(l)=sum_int/w;
                sum_int=0;
                w=0;
            end

        else
            sum_int=0;
            w=0;
        end

end

A1=load('Skype_UAM_audio_caract_d2.dat');
dA1=diff(A1);
B1=dA1(find(abs(dA1(:,2))<1260000000000),:);
C1=B1(find(B1(:,3)==0),2);          %%ELIMINA PAQUETE PRIMERO DE CADA FLUJO
C1=abs(C1/10^6);
s=length(C1);
flow1=A1(:,6)';
i=[1:s];
w=0;
l=0;
sum_int=0;
for i=2:s
        if (flow1(i)==flow1(i-1))
            w=w+1;
            C1(i-1);
            sum_int=C1(i-1)+sum_int;
            if (w==10)
                l=l+1;
                mean_int1(l)=sum_int/w;
                sum_int=0;
                w=0;
            end

        else
```

```
            l=l+1;
            mean_int1(l)=sum_int/w;
            sum_int=0;
            w=0;
        end

end

A2=load('Skype_UAM_audio_caract_d4.dat');
dA2=diff(A2);
B2=dA2(find(abs(dA2(:,2))<1260000000000),:);
C2=B2(find(B2(:,3)==0),2);          %%ELIMINA PAQUETE PRIMERO DE CADA FLUJO
C2=abs(C2/10^6);
flow2=A2(:,6)';
s=length(C2);
i=[1:s];
w=0;
l=0;
sum_int=0;
for i=2:s
        if (flow2(i)==flow2(i-1))
            w=w+1;
            C2(i-1);
            sum_int=C2(i-1)+sum_int;
            if (w==10)
                l=l+1;
                mean_int2(l)=sum_int/w;
                sum_int=0;
                w=0;
            end

        else
            l=l+1;
            mean_int2(l)=sum_int/w;
            sum_int=0;
            w=0;
        end

end

A3=load('Skype_UAM_audio_caract_d8.dat');
dA3=diff(A3);
B3=dA3(find(abs(dA3(:,2))<1260000000000),:);
C3=B3(find(B3(:,3)==0),2);          %%ELIMINA PAQUETE PRIMERO DE CADA FLUJO
C3=abs(C3/10^6);
flow3=A3(:,6)';

s=length(C3);
i=[1:s];
w=0;
l=0;
sum_int=0;
for i=2:s
        if (flow3(i)==flow3(i-1))
            w=w+1;
            C3(i-1);
            sum_int=C3(i-1)+sum_int;
            if (w==10)
```

```
                    l=l+1;
                    mean_int3(l)=sum_int/w;
                    sum_int=0;
                    w=0;
                end


            else
                sum_int=0;
                w=0;
            end

end

A4=load('Skype_UAM_audio_caract_d16.dat');
dA4=diff(A4);
B4=dA4(find(abs(dA4(:,2))<1260000000000),:);
C4=B4(find(B4(:,3)==0),2);          %%ELIMINA PAQUETE PRIMERO DE CADA FLUJO
C4=abs(C4/10^6);
s=length(C4);
i=[1:s];
w=0;
l=0;
sum_int=0;
flow4=A4(:,6)';
for i=2:s
            if (flow4(i)==flow4(i-1))
                w=w+1;
                C4(i-1);
                sum_int=C4(i-1)+sum_int;
                if (w==10)
                    l=l+1;
                    mean_int4(l)=sum_int/w;
                    sum_int=0;
                    w=0;
                end

            else
                l=l+1;
                mean_int4(l)=sum_int/w;
                sum_int=0;
                w=0;
            end

end

A5=load('Skype_UAM_audio_caract_d64.dat');
dA5=diff(A5);
B5=dA5(find(abs(dA5(:,2))<1260000000000),:);
C5=B5(find(B5(:,3)==0),2);          %%ELIMINA PAQUETE PRIMERO DE CADA FLUJO
C5=abs(C5/10^6);

s=length(C5);
i=[1:s];
w=0;
l=0;
sum_int=0;
flow5=A5(:,6)';
```

```
for i=2:s
        if (flow5(i)==flow5(i-1))
            w=w+1;
            C5(i-1);
            sum_int=C5(i-1)+sum_int;
            if (w==10)
                l=l+1;
                mean_int5(l)=sum_int/w;
                sum_int=0;
                w=0;
            end

        else
            l=l+1;
            mean_int5(l)=sum_int/w;
            sum_int=0;
            w=0;
        end

end


figure(2)
[y,x]=ecdf(mean_int);
stairs(x,y,'r')
hold on
[y,x]=ecdf(mean_int1);
stairs(x,y,'b')
[y,x]=ecdf(mean_int2);
stairs(x,y,'m')
[y,x]=ecdf(mean_int3);
stairs(x,y,'g')
[y,x]=ecdf(mean_int4);
stairs(x,y,'k')
[y,x]=ecdf(mean_int5);
stairs(x,y,'m--')
legend('1','2','4','8','16','64')
stairs([0.015 0.015],[0 1],'r--','linewidth',2)
axis([0 0.05 0 1])
ylabel('ECDF')
xlabel('Interarrival(s)')
hold off
```

C.3. Código C: Realización del muestreo y pérdidas

```
#include <unistd.h>
#include <stdio.h>
#include <pcap.h>
#include <netinet/in.h>
#include <sys/socket.h>

#include <string.h>
#include <stdlib.h>
#include <libgen.h>
#include <arpa/inet.h>

#define PCAP_SAVEFILE "_deterministic.pcap"
#define PCAP_SAVEFILE_2 "_stratified.pcap"
#define PCAP_SAVEFILE_3 "_random.pcap"
#define PCAP_SAVEFILE_4 "_loss.pcap"

int packets = 0;          //counter of packets in original trace
int cont=0;               //counter of packets in new trace
int state=0;              //initial state in the simulation of loss(no-loss)

/*************************************
*
* Function: deterministic_sampling_and_dumping
*
* Description: Doing the systematic sampling taking the
*              appropriate packages to the new trace
*
*************************************/
void deterministic_sampling_and_dumping(const struct pcap_pkthdr *hdr,
const u_char *data,pcap_dumper_t *pdump,char sample_rate,int count_pkts)
{

        if(count_pkts%sample_rate==0)
        {

                cont++;
                pcap_dump((u_char*)pdump,hdr,data);

        }
}


/*************************************
*
* Function: stratified_sampling_and_dumping
*
* Description:  Doing the stratified sampling taking
*               the appropriate packages to the new trace
*
*************************************/
void stratified_sampling_and_dumping(const struct pcap_pkthdr *hdr,
const u_char *data,pcap_dumper_t *pdump,int sample_rate,int count_pkts,
int random)
{
```

```
        if(count_pkts==random)
        {
                        cont++;
                        pcap_dump((u_char*)pdump,hdr,data);

        }
}

/*************************************
*
* Function: random_sampling_and_dumping
*
* Description:  Doing the stratified sampling taking
*               the appropriate packages to the new trace
*
*************************************/
void random_sampling_and_dumping(const struct pcap_pkthdr *hdr,
const u_char *data,pcap_dumper_t *pdump,int sample_rate,int count_pkts,
int random)
{

        if(random==0)
        {
                cont++;
                pcap_dump((u_char*)pdump,hdr,data);

        }
}


/*************************************
*
* Function: loss_sampling_and_dumping
*
* Description:  Doing the simulate of loss taking
*               the appropriate packages to the new trace
*
*************************************/
void loss_sampling_and_dumping(const struct pcap_pkthdr *hdr,
const u_char *data,pcap_dumper_t *pdump,int sample_rate,
int count_pkts,double prob_p,double prob_np)
{

        int random;

        random=rand();
        switch(state)
        {
                case 0:                 /*state no loss*/
                {
                        if((double)random/RAND_MAX>=prob_p)
                        {
                                cont++;
                                pcap_dump((u_char*)pdump,hdr,data);
                                state=0;
                                break;

                        }
```

```
                        else
                        {
                                state=1;
                                break;
                        }
                }

                case 1:             /*state no loss*/
                        if((double)random/RAND_MAX<prob_np)
                        {
                                cont++;
                                pcap_dump((u_char*)pdump,hdr,data);
                                state=0;
                                break;

                        }
                        else
                        {
                                state=1;
                                break;
                        }
        }

}



int main(int argc, char* argv[])
{

        pcap_t *p;                                      /*original trace*/
        pcap_dumper_t *pd;                              /*new trace*/
        struct pcap_pkthdr hdr;
        u_char* bp;
        char filename[80];
        char errbuf[PCAP_ERRBUF_SIZE];
        int dice;                                       /*random number*/
        int n=0;
        double p_p;                                     /*probability of loss*/
        double rate_p;
        char trace[80];

        char type[30];
        char rute[80];
        double r1;
        double r2;

        if (argc >= 2)
        {
                strcpy(rute,argv[1]);
                strcpy(type,argv[2]);
                strcpy(trace,argv[3]);

        }


        strcat(rute,trace);     /*opening the selected trace in read mode*/
```

```
        if (!(p = pcap_open_offline(rute, errbuf)))
        {
                fprintf(stderr,"Error␣openinng␣the␣file:␣%s\n", errbuf);
                exit(2);
        }



        if(strcmp(type,"systematic")==0)
        {

                r1=atoi(argv[4]);
                strcat(rute, PCAP_SAVEFILE);
                strcpy(filename, rute);

                /*open an output file in which to save the data
                that we capture*/
                if ((pd = pcap_dump_open(p,filename)) == NULL)
                {
                        fprintf(stderr,"Error␣opening␣the␣file␣\"␣%s\"
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣writing:␣%s\n",filename, pcap_geterr(p));
                        exit(7);
                }

                while((bp=pcap_next(p,&hdr))!=NULL)
                {
                        deterministic_sampling_and_dumping(&hdr,bp,pd,r1,
                        packets);
                        packets++;
                }

                pcap_dump_close(pd);
                pcap_close(p);

                printf("\nPackets␣read␣from␣the␣initial␣file:␣%d\n",
                packets);
                printf("Packets␣processed␣after␣sampling:␣%d\n\n", cont);
                return 0;
        }

        if(strcmp(type,"stratified")==0)
        {

                r1=atoi(argv[4]);
                strcat(rute, PCAP_SAVEFILE_2);
                strcpy(filename, rute);

                /*open an output file in which to save the data
                that we capture*/
                if ((pd = pcap_dump_open(p,filename)) == NULL)

                {
                        fprintf(stderr,"Error␣opening␣the␣file␣\"␣%s\"
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣writing:␣%s\n",filename, pcap_geterr(p));
                        exit(7);
                }

                while((bp=pcap_next(p,&hdr))!=NULL)
```

```
                {
                        if(packets%(int)r1==0)
                        {
                                /*random number to pick up a package
                                in the range*/
                                dice = rand()%(int)r1;
                                dice=dice+r1*n;
                                n++;

                        }
                        stratified_sampling_and_dumping(&hdr,bp,pd,r1,
                        packets,dice);
                        packets++;
                }

                pcap_dump_close(pd);
                pcap_close(p);

                printf("\nPackets read from the initial file: %d\n",
                packets);
                printf("Packets processed after sampling: %d\n\n", cont);
                return 0;
        }

        if(strcmp(type,"random")==0)
        {
                r1=atoi(argv[4]);
                strcat(rute, PCAP_SAVEFILE_3);
                strcpy(filename, rute);

                /*open an output file in which to save the data
                that we capture */
                if ((pd = pcap_dump_open(p,filename)) == NULL)
                {
                        fprintf(stderr,"Error opening the file \" %s \"
                        for writing: %s\n",filename, pcap_geterr(p));
                        exit(7);
                }

                while((bp=pcap_next(p,&hdr))!=NULL)
                {
                        /*random number to pick up packets*/
                        dice = rand()%(int)r1;
                        random_sampling_and_dumping(&hdr,bp,pd,r1,
                        packets,dice);
                        packets++;
                }

                pcap_dump_close(pd);
                pcap_close(p);

                printf("\nPackets read from the initial file: %d\n",
                packets);
                printf("Packets processed after sampling: %d\n\n", cont);
                return 0;
        }

        if(strcmp(type,"loss")==0)
```

```
        {
                r1=atof(argv[4]);
                r2=atof(argv[5]);
                if(r1>1 || r2>1)
                {
                        printf("\nERROR: The probability must be less
                        or equal to one\n\n");
                        return 0;
                }

                strcat(rute, PCAP_SAVEFILE_4);
                strcpy(filename, rute);

                /*open an output file in which to save
                the data that we capture*/
                if ((pd = pcap_dump_open(p,filename)) == NULL)
                {
                        fprintf(stderr,"Error opening the file \" %s \"
                        for writing: %s\n",filename, pcap_geterr(p));
                        exit(7);
                }

                /*calculation of the probability of loss*/
                p_p=r2*(1-r2)/r1;
                printf("\nProbability of loss: %f\n", p_p);

                while((bp=pcap_next(p,&hdr))!=NULL)
                {
                        loss_sampling_and_dumping(&hdr,bp,pd,r1,packets,
                        p_p,r2);
                        packets++;
                }

                pcap_dump_close(pd);
                pcap_close(p);

                printf("\nPackets read from the initial file: %d\n",
                packets);
                printf("Packets processed after sampling: %d\n", cont);

                /*calculation of the sampling rate*/
                rate_p=(double)packets/cont;
                printf("Value of x (1/x):%lf\n\n",rate_p);
                return 0;
        }

        printf("\nERROR\n\n");
        return 0;

}
```

D

Artículo IM 2013

A continuación mostramos el artículo enviado a la conferencia internacional IFIP/IEEE *Integrated Network Management Symposium* (IM 2013) con proceso de selección y revisión de artículos *peer review*. Dicho artículo, con nombre ***"On the Impact of Packet Sampling on Skype Traffic Classification"*** ha sido aceptado, el día 7 Noviembre de 2012, para su presentación en la conferencia [28].

Este artículo se centra en una parte de este proyecto, en el impacto del muestreo en el detector de tráfico Skype, Skypeness; donde se analiza la posible caracterización y clasificación de este tipo de tráfico en un entorno de alta velocidad.

D.1. Mail de aceptación del artículo

Your paper #107198 ("On the Impact of Packet Sampling on Skype Traffic Classification") has been accepted to the Short Papers track of the IFIP/IEEE Integrated Network Management Symposium (IM 2013). Congratulations!

All submitted papers underwent a review process with a minimum of 3 reviews for each paper. After author rebuttals were submitted, TPC discussions took place on-line through the JEMS conference management system and in a face-to-face TPC meeting held October 25 in Las Vegas, USA. Every paper has been discussed taking into account its content, the reviews and the submitted rebuttal.

The reviews are at the end of this message and can also be found at:

https://jems.sbc.org.br/PaperShow.cgi?m=107198.

You are invited to submit a camera-ready version of your paper to appear in the IM 2013 proceedings and in IEEE Xplore. The paper length for short papers is 4 pages maximum. We kindly ask you to address the suggestions raised by the reviewers. The IM 2013 Posters Co-Chairs will check that this has been performed properly. Short papers are presented in the Poster Sessions during the Symposium, the Poster Sessions will have a prominent place in the symposium program.

The deadline for you to finalize and submit your camera-ready manuscript is *** 15 January 2013 ***. It must be formatted according to the instructions available at the "Authors Informa-

tion" section of the IM 2013 website (http://www.ieee-im.org/authors.html). Further information regarding the preparation and upload of your camera-ready manuscript, as well as its presentation during the Symposium, will be available shortly at http://www.ieee-im.org/authors.html.

Please note that IEEE Communications Society policies require that at least one co-author of the paper be registered for the conference, in order for the paper to appear in the proceedings and in IEEE Xplore. The paper must be presented at the conference, otherwise it will be excluded from IEEE Xplore and IFIP database. Additional details about the registration policy as well as the IM 2013 program will soon be available at http://www.ieee-im.org.

Please also note that the call for workshop papers is open

(http://www.ieee-im.org/program/workshops), an interesting tutorial program is being composed, each registered participant will be offered one free tutorial. We also would like to encourage participants to show a demo during the symposium (http://www.ieee-im.org/submissions/call-for-demonstrations-and-exhibitions): in addition to a paper presentation, it can be interesting to refer to a real-life demo during a demo session.

Again, congratulations on your fine work. We look forward to seeing you in Ghent, Belgium, 27-31 May 2013!

Best regards,

Luciano Paschoal Gaspary and Aldri dos Santos IFIP/IEEE IM 2013 Posters Co-Chairs

Filip De Turck, Yixin Diao, Choong Seon Hong IFIP/IEEE IM 2013 Technical Program Committee Co-Chairs

========== Reviews ==========

===== Review =====

*** 2: Relevance (): 1:Out of Scope 2:Somewhat Relevant 3:Highly Relevant

Evaluation=Highly Relevant (3)

*** 3: Technical Content and Originality (): 1:Poor 2:Fair 3:Good 4:Excellent

Evaluation=Good (3)

*** 4: Organization and Presentation (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Good (3)

*** 5: Reference to Related Work (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Poor (2)

*** 6: Overall Recommendation (): 1:Strong Reject - I have strong arguments against acceptance 2:Weak Reject - I will not fight strongly against it 3:Weak Accept - I will not fight strongly in favour of acceptance 4:Strong Accept - I have strong arguments in favour of acceptance

Evaluation=Weak Accept - I will not fight strongly in favour of acceptance (3)

*** 7: Poster Acceptance (If this paper happens to be rejected as a full paper, please express your opinion on accepting it for a poster presentation / short paper (4 pages) publication.): 1: Strong Reject - I have strong arguments against accepting this work as a poster 2: Weak Reject - I will not fight strongly against accepting this work as a poster 3: Weak Accept - I will not fight strongly in favor of accepting this work as a poster 4: Strong Accept - I have strong arguments in favor of accepting this work as a poster

Evaluation=Strong Accept - I have strong arguments in favor of accepting this work as a poster (4)

*** 8 (What are the major strengths of this paper?): The authors present results of a modified version of Skypeness, a tool that they have developed and published earlier. Skypeness can detect and classify Skype traffic using statistical features such as delimited packet sizes, inter-arrival times and bounded bitrates. The authors claim that they have released Skypeness as open-source Skype traffic detector.

By combining sampling and Skypeness, authors aimed to reduce resource consumption (computing and sniffing resources) of traffic detection. The modification is based on sampling mechanisms and authors empirically show the accuracy reduction in the sampling based approaches. Then authors present a modified version of sampling mechanism that improves the accuracy.

*** 9 (What are the major shortcomings of this paper?): The authors only consider UDP based Skype traffic. However, in many scenarios where there are firewalls, Skype traffic is transmitted using TCP; particularly when one end-point is located in a restricted environment (e.g., behind a firewall). The proposed scheme will not work in such contexts. The authors can look into these scenarios and can further improve their solution.

*** 10 (Comments for the authors. Please make sure to provide a solid and constructive review for the authors to improve their paper. Include detailed comments and inform any missing related work (especially in regard to previous IM/NOMS/CNSM editions and related journals IJNM/JNSM/TNSM/etc).): The sampling policies discussed in the paper are fairly limited (Sec. III B). There are other ways to sample. For example, a window based sample mechanisms can be used: that is sampling can be done in a given window capturing many packets in that window and then wait for a particular time period without performing any sampling/sniffing. This approach will significantly reduce the problem of not being able to capture inter-arrival time correctly.

===== Review =====

*** 2: Relevance (): 1:Out of Scope 2:Somewhat Relevant 3:Highly Relevant

Evaluation=Highly Relevant (3)

*** 3: Technical Content and Originality (): 1:Poor 2:Fair 3:Good 4:Excellent

Evaluation=Fair (2)

*** 4: Organization and Presentation (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Good (3)

*** 5: Reference to Related Work (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Good (3)

*** 6: Overall Recommendation (): 1:Strong Reject - I have strong arguments against acceptance 2:Weak Reject - I will not fight strongly against it 3:Weak Accept - I will not fight strongly in favour of acceptance 4:Strong Accept - I have strong arguments in favour of acceptance

Evaluation=Weak Reject - I will not fight strongly against it (2)

*** 7: Poster Acceptance (If this paper happens to be rejected as a full paper, please express your opinion on accepting it for a poster presentation / short paper (4 pages) publication.): 1: Strong Reject - I have strong arguments against accepting this work as a poster 2: Weak Reject - I will not fight strongly against accepting this work as a poster 3: Weak Accept - I will not fight strongly in favor of accepting this work as a poster 4: Strong Accept - I have strong arguments in favor of accepting this work as a poster

Evaluation=Weak Reject - I will not fight strongly against accepting this work as a poster (2)

*** 8 (What are the major strengths of this paper?): The main strengths of this paper are that the evaluation is extensive for a short paper and that the ideas in it are well presented.

*** 9 (What are the major shortcomings of this paper?): The main weakness is that the contribution is slim. The solution proposed to compensate for the sampling does not represent a significant contribution to those familiar with sampling.

*** 10 (Comments for the authors. Please make sure to provide a solid and constructive review for the authors to improve their paper. Include detailed comments and inform any missing related work (especially in regard to previous IM/NOMS/CNSM editions and related journals IJNM/JNSM/TNSM/etc).): As stated above, the technique you use to compensate for the sampling is not a significant contribution per se.

When describing your dataset, you refer to synthetic trace captured. It is unclear to me what you mean by this. Are the traces synthetic (i.e., generated outside a network) or have they been captured from real traffic?

Figure 3 shows a spike around 10r. That is counterintuitive. You should explain the cause for that spike.

===== Review =====

*** 2: Relevance (): 1:Out of Scope 2:Somewhat Relevant 3:Highly Relevant

Evaluation=Somewhat Relevant (2)

*** 3: Technical Content and Originality (): 1:Poor 2:Fair 3:Good 4:Excellent

Evaluation=Fair (2)

*** 4: Organization and Presentation (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Good (3)

*** 5: Reference to Related Work (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Good (3)

*** 6: Overall Recommendation (): 1:Strong Reject - I have strong arguments against acceptance 2:Weak Reject - I will not fight strongly against it 3:Weak Accept - I will not fight strongly in favour of acceptance 4:Strong Accept - I have strong arguments in favour of acceptance

Evaluation=Weak Accept - I will not fight strongly in favour of acceptance (3)

*** 7: Poster Acceptance (If this paper happens to be rejected as a full paper, please express your opinion on accepting it for a poster presentation / short paper (4 pages) publication.): 1: Strong Reject - I have strong arguments against accepting this work as a poster 2: Weak Reject - I will not fight strongly against accepting this work as a poster 3: Weak Accept - I will not fight strongly in favor of accepting this work as a poster 4: Strong Accept - I have strong arguments in favor of accepting this work as a poster

Evaluation=Strong Accept - I have strong arguments in favor of accepting this work as a poster (4)

*** 8 (What are the major strengths of this paper?): The authors extend their approach to classify skype traffic such that it can be applied even when sampling is performed. They show that with a little adaptation, the accuracy is still acceptable. The tool and the datasets are available online which is highly appreciated.

*** 9 (What are the major shortcomings of this paper?): As the main motivation is to enhance the scalability by handling voluminous traffic flow, it is surprising that there is no evaluation about that.

*** 10 (Comments for the authors. Please make sure to provide a solid and constructive review for the authors to improve their paper. Include detailed comments and inform any missing

related work (especially in regard to previous IM/NOMS/CNSM editions and related journals IJNM/JNSM/TNSM/etc).): - Why defining all the different metrics as well as accuracy since finally each dataset only contains a certain kind of traffic. For instance, accuracy uses TP and TN but such terms can be never calculated simultaneously. Maybe just defining accuracy as the proportion of traffic rightly classified is enough as there is no two types of traffic meantime. - Another option should be to mix the different dataset. - The datasets does not look realist as the only trace containing non-skype traffic is "synthetic". Could you tell a bit more about how you generate it.

- Please avoid colors in figure for Black and white printing or at least combine them with different line styles (fig. 1).

[The rebuttal has been read and modifications have been performed in the review.]

===== Review =====

*** 2: Relevance (): 1:Out of Scope 2:Somewhat Relevant 3:Highly Relevant

Evaluation=Highly Relevant (3)

*** 3: Technical Content and Originality (): 1:Poor 2:Fair 3:Good 4:Excellent

Evaluation=Good (3)

*** 4: Organization and Presentation (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Poor (2)

*** 5: Reference to Related Work (): 1:Unacceptable 2:Poor 3:Good 4:Excellent

Evaluation=Good (3)

*** 6: Overall Recommendation (): 1:Strong Reject - I have strong arguments against acceptance 2:Weak Reject - I will not fight strongly against it 3:Weak Accept - I will not fight strongly in favour of acceptance 4:Strong Accept - I have strong arguments in favour of acceptance

Evaluation=Strong Accept - I have strong arguments in favour of acceptance (4)

*** 7: Poster Acceptance (If this paper happens to be rejected as a full paper, please express your opinion on accepting it for a poster presentation / short paper (4 pages) publication.): 1: Strong Reject - I have strong arguments against accepting this work as a poster 2: Weak Reject - I will not fight strongly against accepting this work as a poster 3: Weak Accept - I will not fight strongly in favor of accepting this work as a poster 4: Strong Accept - I have strong arguments in favor of accepting this work as a poster

Evaluation=Strong Accept - I have strong arguments in favor of accepting this work as a poster (4)

*** 8 (What are the major strengths of this paper?): - The simple idea of sampling traffic according to the well known Skype traffic profile is a simple and good idea. - It works.

*** 9 (What are the major shortcomings of this paper?): - The quality of English is poor. The paper is full of spelling mistakes. A proof reading would have limited their number.

*** 10 (Comments for the authors. Please make sure to provide a solid and constructive review for the authors to improve their paper. Include detailed comments and inform any missing related work (especially in regard to previous IM/NOMS/CNSM editions and related journals IJNM/JNSM/TNSM/etc).): The paper addresses a well presented problem that raised a significant amount of work already. This paper does not propose a completely new method, but based on previous work, the originality of the paper contribution is clear. Despite obvious, this approach was never used before up to my knowledge. It proved to work well on the few traces on which Skypeness was used.

On the Impact of Packet Sampling on Skype Traffic Classification

P.M. Santiago del Río, D. Corral, J.L. García-Dorado, J. Aracil
High Performance Computing and Networking
Universidad Autónoma de Madrid, Spain
Email: pedro.santiago@uam.es,diego.corral@estudiante.uam.es,{jl.garcia,javier.aracil}@uam.es

***Abstract*—Nowadays, traffic classification technology addresses the exciting challenge of dealing with ever-increasing network speeds, which implies more computational load especially when on-line classification is required, but avoiding to reduce classification accuracy. Among other solutions, the research community has proposed to sample packets to reduce such a load but the impact of this measurement on traffic classification has only been marginally studied. This paper addresses such a study focusing on Skype application given its tremendous popularity and continuous expansion. Skype, unfortunately, is based on a proprietary design, and typically uses encryption mechanisms, making the study of statistical traffic characteristics and the use of Machine Learning techniques the only possible solution. Consequently, we have studied Skypeness, an open-source system that allows to detect Skype at multi-10 Gb/s rates applying statistical principles. We have assesses its performance applying different packet sampling rates and policies concluding that classification accuracy is significantly degraded when packet sampling is applied. Nevertheless, we propose a simple modification in Skypeness that lessens such a degradation. This consists in scaling the measured packet interarrivals used to classify according to the sample rate which have resulted in a significant gain.**

***Index Terms*—Skype; Traffic Classification; Packet sampling; High-speed networks.**

I. Introduction

Both the research community and network operators have dedicated extensive effort to the development of the traffic classification technology given its relevance in management tasks as important as the network design and engineering, security, advertising, or DiffServ mechanisms [1]. Similarly, traffic classification allows to analyze changes in the Internet, understand the behavior of different applications and the traffic generated by them. Specifically, on-line traffic classification has proven useful for a set of tasks that require to take measurements on-the-fly. Examples of such tasks are intrusion detection, accounting, quality of service (QoS) or quality of experience (QoE) management and lawful-interception.

Nonetheless, the ever-increasing data transmission rates has become traffic classification in an exciting challenge. In multi-10Gb/s networks, very common nowadays, traffic classifiers have to be able to capture and analyze up to several tens of millions of packets per second. In spite of improvements on capture capabilities and efforts to optimize and relieve classification mechanisms of burden [2], to date many network monitoring systems only deal with packet sampling data in an attempt to reduce such burden. That is, traffic classification systems are not provided with all the traffic but only a fraction of the packets are taken into account.

The relationship between traffic classification and packet sampling was first pointed out in [3]. In such work, the monitoring system first sampled at packet level, then generated Netflow records, and finally the records were classified using machine learning (ML) techniques [2] (specifically, decision trees). Note that Netflow data records only comprises information about the source and destination IP addresses, port numbers, protocol and counters of bytes and packets. Similarly, the authors in [4] proposed to use packet-sampled flow records that included a more extensive set of features, e.g., RTT or number of ACKs. Both studies concluded that sampling entails a significant impact on the classification performance above all in terms of volume in bytes and packets. Differently, this paper does not analyze packet-sampled flows but assumes a monitoring system fed with a sample of the total packets traversing the monitored link. The advantages are twofold, the first one is that the accuracy increases, and the second one is that it allows to classify on-the-fly. Note that flow-based classifying requires that flows end before being analyzed. This is unacceptable in VoIP applications where operators have to apply measurements, such as accounting, improve quality or, conversely, blocking if some VoIP applications are not allowed by contract, while the call is in course, and not after its finalization.

Specifically, we turn our interest to Skype classification given its tremendous popularity and continuous expansion between the clients of VoIP [5]. In fact, Skype has also attracted the attention of the research community which has characterized its behavior [5] and proposed several detection algorithms [6], [7]. In this paper, we have evaluated the impact of sampling on the classification of Skype using *Skypeness* [7] over both synthetic and real traces from public repositories. Skypeness is a commodity off-the-shelf system to Skype traffic detection at multi-10G/s rates based on the functionality of Tstat Skype module [8] but with a simpler software implementation to allow its on-line execution.

The results show that Skype detectors are affected by sampling because the statistical characteristics which they are based on, such as interarrival times, are distorted by sampling. However, we propose a simple modification in the detection algorithm to mitigate such effects. Particularly, the observed interarrivals that Skypeness uses to make a decision are scaled

according to the sampling rate. With this modification, the results are similar to those with unsampled traffic, although at the expense of a small increment in the false positive ratio. Consequently, this study proves that sampling is not a definitive pitfall to track Skype at multi-10Gb/s.

As an additional contribution of this work, we have made public for the research community as open-source the code of Skypeness, the programs that we have used to sample packets in traces, as well as the Skype traces used as testbed[1].

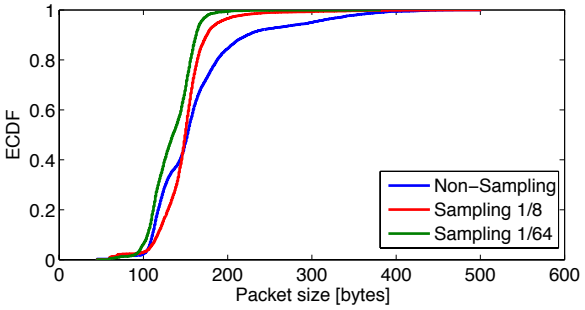
II. Skype traffic classification

Skype traffic, unlike traditional services and protocols, cannot be detected using well-known ports or applying deep packet inspection (DPI) techniques, because Skype uses a proprietary, obfuscated and encrypted protocol that employs per-session random ports. The answer of the research community has been the use of statistical traffic characteristics and ML techniques [2].

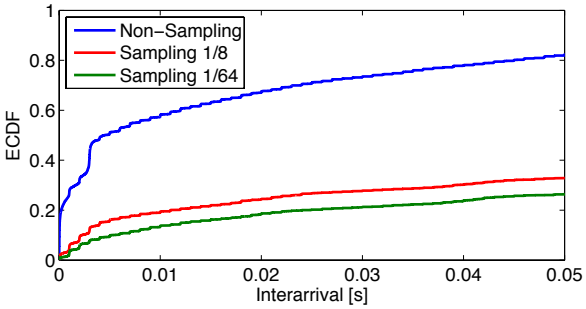
In [6] the authors presented a Skype traffic detection algorithm based on two statistical techniques: First, they infer a probability distribution of both packet length and inter-arrival time from audio and video codecs used by Skype. Then, it is checked if the empirical distributions of a given flow fit with the hypothesized ones, using a Bayesian classifier. Second, as Skype traffic is encrypted, it is checked if the payload of a given flow follows a uniform distribution, using Pearson's Chi-Square estimator. The algorithm is implemented as a module of Tstat [8]. However, Tstat documentation explains that the Bayesian classifier configuration requires a fine parameter configuration and significant computation load limiting its applicability to multi-10Gb/s networks.

In this light, we borrowed Tstat's proposals and developed Skypeness [7], a high-performance Skype traffic classifier based on three intrinsic characteristics of Skype traffic, namely: delimited packet size, nearly constant packet interarrival times and bounded bitrate. Specifically, Skypeness computes the mean values of these three features (packet size, interarrival time and bitrate), averaging in windows of 10 packets, for each flow. If the ratio of packet windows whose mean values are inside of a given interval or is greater than a given threshold, such flows is marked as Skype. For instance, Fig. 1 shows the appropriate interval and threshold values for audio Skype calls, specifically it shows the empirical cumulative distribution functions for packet size and interarrival time increments from 44 Skype audio calls when no sampling is applied (darker line). Thus, packet size is well delimited (between 60 and 200 bytes more than 75% of the packets) and more than 60% of the interrarival increments are less than 15 ms. Table I shows all intervals and thresholds corresponding to the different classes of Skype traffic, namely, only audio calls, video (and audio) calls and file transfers. Note that the detector only considers UDP flows which have more than 30 packets (three packet windows). Skype typically uses only UDP as transport-layer because it is more suitable in real-time applications. However, it is uncommon but possible that Skype shifts to TCP in an attempt to evade firewalls or other similar restrictions. As we leverage on packet interarrivals assuming they are fairly constants, and TCP can modify this depending on its configuration, we have focused on UDP traffic.

[1]http://www.eps.uam.es/~psantiago/skypeness.html



(a) Packet size.



(b) Interarrival time increments.

Fig. 1: Empirical CDF for packet size and interarrival times in audio Skype calls.

TABLE I: Intervals and threshold values used by Skypeness detector.

Media	Characteristic	Interval	Threshold
Audio	Packet size [Bytes]	$[60, 200]$	0.75
	Interarrival [ms]	$[i_{n-1} \pm 15]$	0.6
	Bitrate [Kbps]	$[0, 150]$	0.75
Video	Packet size [Bytes]	$[150, 1200]$	0.19
	Interarrival [ms]	$[i_{n-1} \pm 15]$	0.6
File Transfer	Packet size [Bytes]	$[480, 540] \cup [950, 1050] \cup [1310, 1380]$	0.44

Although packet size is not affected by packet sampling (Fig. 1a), interarrival time is distorted when sampling is applied (Fig. 1b) and, therefore, the expected interval values are not longer valid. Thus, Skypeness detection accuracy is reduced to nearly zero in presence of packet sampling. This fact will be analyzed in more detail in Section V.

III. Methodology

A. Classification accuracy metrics

In order to measure the detector accuracy, let us define the following metrics:

- False negative (FN): amount of Skype traffic classified as Non-Skype traffic.
- False positive (FP): amount of Non-Skype traffic classified as Skype traffic.
- True negative (TN): amount of Non-Skype traffic well classified.
- True positive (TP): amount of Skype traffic well classified.
- Accuracy: $\frac{TP+TN}{TP+FP+TN+FN}$

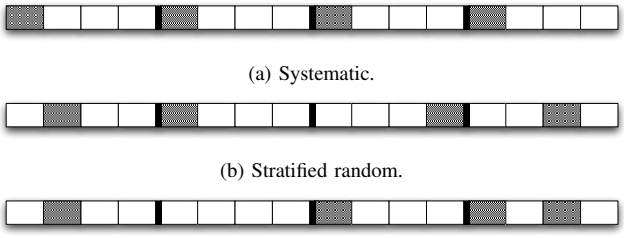
That is, accuracy is the ratio of traffic correctly classified. Metrics described above can be counted using bytes, packets or flows. The choice of the unit (packets, bytes or flows) depends on the purpose of the classification.

B. Packet sampling policies

Packet sampling techniques allows to choose a fraction of the total amount of packets, following a given criterion to reduce the computational burden of any subsequent analysis. Figure 2 shows the three main packet sampling policies [9], namely:

- Systematic: data are split in cycles of n packets and the first element of each cycle is deterministically chosen.
- Stratified random: data are also split in cycles of n packets but one element of each cycle is randomly chosen.
- Simple random: each packet is randomly chosen with a given probability $1/n$.

Sampling techniques can be implemented using mechanisms based on either events or timer [9]. That is, each cycle can be either an amount of packets or a time interval. In our case, the cycle is an amount of packets (equal to the inverse of the sampling rate) due to its better performance.



(a) Systematic.

(b) Stratified random.

(c) Simple Random.

Fig. 2: Packet Sampling Policies.

IV. Datasets

We have made use of four different traces of UDP traffic, Table II shows an overview of the datasets. The first and second traces, named as Trace 1 and Trace 2 in the following, contain Skype traffic captured on the access link of Politecnico di Torino [10]. The set of users are students, faculty and administration staff. The capture duration is 96 hours in May/June, 2006. Trace 1 only contains end-to-end Skype audio and video calls whereas Trace 2 only contains Skype end-to-out calls. Trace 1 and Trace 2 contain 40M and 3M packets respectively. The third trace, named as Trace 3, contains Skype traffic generated in our laboratory at Universidad Autónoma de Madrid in May, 2010. The trace contains 700K packets from end-to-end Skype voice (3A) and video (3B) calls, as well as file transfers (3C). The last trace used, named as Trace 4, is a synthetic trace captured in our laboratory that contains 5K packets of P2P traffic from several applications, such as eMule and BitTorrent. With this in mind, traces 1, 2 and 3 are useful to estimate false negatives rate because such traces only contain Skype traffic. True positives rate are estimated with Trace 4 as this trace does not contain Skype traffic.

TABLE II: Datasets.

Trace		Skype	Non-Skype	Skype Media
Trace 1	Bytes	8,381,658,970	0	Audio and Video
	Packets	39,458,562	0	
	Flows	1059	0	
Trace 2	Bytes	231,257,652	0	Audio
	Packets	3,049,148	0	
	Flows	159	0	
Trace 3A	Bytes	30,950,000	0	Audio
	Packets	230,100	0	
	Flows	44	0	
Trace 3B	Bytes	108,700,000	0	Video
	Packets	217,300	0	
	Flows	46	0	
Trace 3C	Bytes	162,800,000	0	File transfer
	Packets	254,300	0	
	Flows	46	0	
Trace 4	Bytes	0	1,098,935	-
	Packets	0	5312	
	Flows	0	52	

V. Performance evaluation

To assess the effect of packet sampling on the accuracy of Skypenes detector, we have applied the three sampling policies (described in Section III-B) varying the sampling rate between $1/2^0$ (no sampling) and $1/2^{10}$ over the four packet traces.

As an example, Fig. 3 shows the accuracy of Skypeness (continuous line) for trace 3A, while Table III reports the results for all traces (roman fonts). For space constraints,

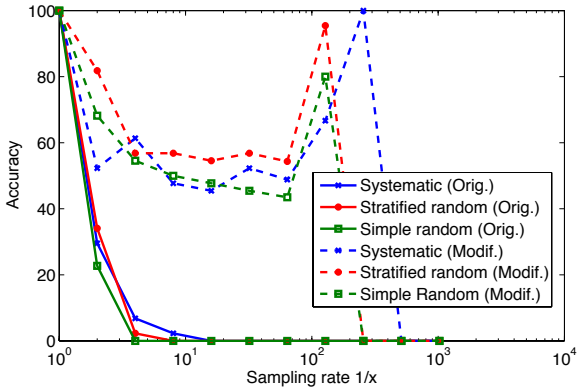


Fig. 3: Skypeness (original and modified versions) accuracy (in bytes) applying different sampling policies and varying sampling rate over Trace 3A (audio calls).

TABLE III: Accuracy (% of bytes) of Skypeness detector original version (roman fonts) and modified version (italic fonts).

Trace	Non-Sampling	Systematic			Stratified Random			Simple random		
		1/8	**1/64**	**1/128**	**1/8**	**1/64**	**1/128**	**1/8**	**1/64**	**1/128**
Trace 1	99.59	3.87	0.15	0.04	0.61	0.02	1.23	0.05	11.92	0.17
		90.72	*95.02*	*95.53*	*90.27*	*93.55*	*94.98*	*87.65*	*91.60*	*93.20*
Trace 2	94.22	35.24	0.54	0.00	24.66	0.00	0.00	9.07	0.00	0.00
		75.32	*85.36*	*90.86*	*73.85*	*92.36*	*96.52*	*65.14*	*71.63*	*88.92*
Trace 3A	100	2.41	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		54.51	*56.20*	*72.19*	*63.02*	*63.99*	*94.75*	*56.35*	*58.30*	*78.49*
Trace 3B	81.38	5.96	0.00	2.68	6.05	1.40	10.40	0.00	0.85	2.76
		84.48	*81.97*	*82.75*	*86.16*	*91.51*	*82.70*	*88.73*	*86.55*	*70.82*
Trace 3C	95.83	96.24	94.76	96.09	96.29	95.29	94.99	95.98	95.64	96.69
	Non-Sampling	**Systematic**			**Stratified Random**			**Simple random**		
		1/2	**1/4**	**1/8**	**1/2**	**1/4**	**1/8**	**1/2**	**1/4**	**1/8**
Trace 4	100	100	100	100	98.87	100	100	100	100	100
		83.00	*95.67*	*100*	*97.19*	*95.53*	*79.92*	*77.26*	*68.04*	*76.85*

we only show the results for the cases of sampling rates, $s \in \{1/8, 1/64, 1/128\}$. Note that in the case of Trace 4 $s \in \{1/2, 1/4, 1/8\}$, because there is no enough packets when greater sampling rates are applied (recall that we only consider UDP flows with more than 30 packets).

The accuracy suffers a significant cut even when a sampling rate of only $1/8$ is applied for both audio and video traces. This is because packet mean interarrival times do not longer fall inside of the expected intervals assuming unsampled traffic. That is, flows are not identified as Skype calls as packet interarrival time is proportionally incremented with sampling rate – as shown in Fig. 1b. Conversely, in the case of trace 3C (file transfer), packet sampling does not have impact on the accuracy because, in this case, the classifier is only based on packet sizes – and packet size distribution is not affected by packet sampling , as shown in Fig. 1a).

In order to adapt the detector to packet sampling, we multiply the observed interarrival times by the sampling rate, thus reducing their values up to the expected intervals when no sampling is applied. Table III shows the accuracy obtained by such modified version of Skypeness detector (italic fonts). The detector is able to correctly classify, applying systematic or stratified sampling over the Trace 1 (the best case), more than 90% of the traffic regardless the sampling rate. Note that this implies that the detector is able to classify with only 1 out of 128 packets, indeed the results show that the detector after the modification is practically insensitive to the sampling rate. The rest of the traces show also significant accuracy (but the Trace 3A), such accuracy ranges between 73% and more than 95%. In the case of Trace 3A, its accuracy ranges between 54% and 95%, we are investigating on the reasons of this behavior. Similarly, the false positive ratio, shown in Trace 4, presents also good results, that is, only a moderate increase.

Regarding the sampling policies, we have found very little differences. Nonetheless, the results suggest that systematic or stratified random sampling are better choices than simple random sampling.

VI. Conclusion

We have empirically studied the impact of packet sampling on the open-source Skype traffic detector *Skypeness*, which is based on three statistical features of Skype traffic: delimited packet sizes, nearly constant interarrival times and bounded bitrates. We analyze the effect on the detector accuracy of two packet sampling factors, namely: the sampling rate and the sampling policy.

Accuracy decreases dramatically when packet sampling is applied, even with the smallest sampling rates ($1/8$) due to distortion on the observed interarrival times. We have proposed a simple modification in the detector (to multiply the observed interarrivals by the sampling rate), which lessens the accuracy reduction, at the expense of a moderated increment on the false positive ratio. Thus, this work shows that sampling is not a definitive drawback to identify Skype at multi-10Gb/s rates. Particularly, Skypeness would be able to detect Skype traffic at more than 300 Gb/s with notable accuracy, given a sampling rate of $1/8$ [7].

References

[1] A. Dainotti, A. Pescapè, and K. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, no. 1, pp. 35–40, 2012.

[2] T.T.T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surv. Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[3] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Sol-Pareta, "Analysis of the impact of sampling on Netflow traffic classification," *Computer Networks*, vol. 55, no. 5, pp. 1083–1099, 2011.

[4] D. Tammaro, S. Valenti, D. Rossi, and A. Pescapè, "Exploiting packet-sampling measurements for traffic characterization and classification.," *Int. J. Netw. Manag.*, doi 10.1002/nem.1802, 2012.

[5] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi, "Detailed analysis of Skype traffic," *IEEE Trans. Multimed.*, vol. 11, no. 1, pp. 117 –127, 2009.

[6] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing Skype traffic: when randomness plays with you," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 37–48, 2007.

[7] P.M. Santiago del Río, J. Ramos, J.L. García-Dorado, J. Aracil, A. Cuadra-Sánchez, and M. Cutanda-Rodríguez, "On the processing time for detection of Skype traffic," in *Wireless Communications and Mobile Computing Conference*, Istanbul, Turkey, July 2011.

[8] A. Finamore, M. Mellia, M. Meo, M.M. Munafò, and D. Rossi, "Experiences of Internet traffic monitoring with Tstat," *IEEE Network*, vol. 25, no. 3, pp. 8 –14, 2011.

[9] K.C. Claffy, G.C. Polyzos, and H.-W. Braun, "Application of sampling methodologies to network traffic characterization," *SIGCOMM Comput. Commun. Rev.*, vol. 23, no. 4, pp. 194–203, 1993.

[10] Telecommunication Networks Group Politecnico di Torino, "Skype traces: Traces from real Internet traffic: http://tstat.tlc.polito.it/tracce/Polito/2006/11_01_29_May_SKYPE_UDP_E2E.dump.anonim.gz," .