Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXI, págs. 3741-3749 (artículo completo)
Alberto Cardona, Paul H. Kohan, Ricardo D. Quinteros, Mario A. Storti (Eds.)
Salta, Argentina, 13-16 Noviembre 2012

IMPLEMENTACIÓN DE UN ESQUEMA DE TELEOPERACIÓN UTILIZANDO EL SISTEMA OPERATIVO ROS EN EL CONTEXTO DE UN LABORATORIO REMOTO

Sebastián J. Tosco[a], **Fernando Corteggiano**[a], **Mariana Broll**[a]

[a]*Departamento de Telecomunicaciones, Universidad Nacional de Río Cuarto, Ruta Nac. 36 Km 601, 5800 Río Cuarto, Argentina, sjtosco@ing.unrc.edu.ar, fcorteggiano@ing.unrc.edu.ar, mbroll@ing.unrc.edu.ar*

Palabras clave: ROS, middleware, laboratorio remoto, teleoperación.

Resumen. En el presente trabajo se propone un modelo de arquitectura que permite la construcción de un laboratorio teleoperado flexible e independiente de las especificaciones relativas a las redes de comunicaciones. Cuando se plantea este tipo de sistema, se hace necesaria una capa intermedia de software que facilite la abstracción de los detalles de la red y sus parámetros para centrarse en lo relevante desde el punto de vista de la ley de control. El objetivo que se persigue es proponer una arquitectura de teleoperación basada en el uso de un sistema operativo orientado a robótica (ROS) como capa intermedia del sistema, utilizando técnicas de procesamiento de imágenes para la adquisición de variables físicas.

1 INTRODUCCIÓN

Los primeros sistemas de laboratorios remotos planteaban lazos de control de forma local monitoreados a distancia. A lo sumo, se podían establecer ciertos parámetros del lazo, el cual seguía corriendo "in-situ".

Sin embargo, una segunda generación de laboratorios remotos pretende poder implementar el lazo de control *a través de la red*. Esto implica que el control no lo estaría realizando un sistema en la planta, sino uno que remotamente controlaría los dispositivos a través de la red (Wang, 2008). En (Guinaldo, 2010) se puede observar los resultados de implementar técnicas de control anticipativo en un esquema semejante.

Ahora bien, las redes de comunicaciones no son perfectas ni transparentes y, lo que es más, plantean nuevos desafíos en lo que se refiere a llevar a cabo la teleoperación. Entre éstos se cuentan: el retardo de transmisión y el consumo de ancho de banda. Además, hay que tener presente que es todo un desafío establecer un lazo de control *a través de* una red de área amplia (WAN) sin ningún tipo de garantía (por ejemplo Internet), lo que significa que no necesariamente es segura y su calidad no se puede manejar ni establecer, más allá de algunas condiciones de contratación de servicio. Todas estas consideraciones cobran importancia a la hora de implementar aplicaciones reales que trabajen con una red de comunicaciones de por medio (Lum, 2009).

Es en este marco donde se encuentra la necesidad de contar con una capa intermedia (middleware) que permita abstraerse de las cuestiones relacionadas con los parámetros y características intrínsecas de la red para centrarse en el lazo de control en sí (Guinaldo, 2010). De esta manera, un estudiante que opera un sistema remotamente no vería la diferencia con respecto a la operación local del mismo, lo cual sugiere la importancia de escoger bien la capa intermedia a utilizar. Existen varias capas propuestas y plataformas que pueden implementarse (Hardison, 2008). En el presente trabajo se propone utilizar el Robotic Operation System (ROS) dentro de un sistema de teleoperación en el contexto de los laboratorios remotos de nueva generación como modelo de arquitectura.

A continuación se describirán algunas características del sistema ROS y su funcionamiento. Después de eso se presentará un modelo de aplicación del mismo en un laboratorio remoto ubicado en una planta piloto del departamento de tecnología química de la Universidad Nacional de Río Cuarto (UNRC). Finalmente, se comentarán algunos resultados experimentales y posibles avances a futuro.

2 SISTEMA OPERATIVO ORIENTADO A ROBÓTICA (ROS)

Durante la investigación bibliográfica realizada, se buscó un sistema de capa intermedia de arquitectura fácilmente utilizable y de código abierto, pues eso permitiría el estudio, implementación y puesta a punto de forma accesible, contando con la posibilidad de evacuar dudas con la comunidad de investigadores en el tema. Se cree firmemente que el uso de plataforma y software libre en el ámbito académico facilita el trabajo colaborativo, incluso entre profesionales ubicados en partes remotas del mundo.

2.1 Aspectos generales

ROS es un meta-sistema operativo de código abierto pensado para robótica que se ejecuta sobre plataformas basadas en UNIX. Ofrece los servicios que se esperarían de un sistema operativo, por ejemplo: la abstracción de hardware de bajo nivel de control del dispositivo, la

aplicación de la funcionalidad de uso común, el paso de mensajes entre los procesos y la gestión de paquetes. También proporciona herramientas y bibliotecas para la obtención, construcción, escritura y ejecución de código en varias arquitecturas. En nuestro caso la implementamos sobre el sistema operativo Ubuntu (kernel 3+) en plataformas x86 y también ARM Cortex A8.

2.2 Arquitectura del sistema ROS

La filosofía de ROS se puede resumir en los siguientes items:

- Punto a Punto
- Basado en herramientas
- Multilenguaje
- Modular
- De código abierto

Por ejemplo, en grandes robots para los cuales ROS fue diseñado, normalmente hay varios ordenadores de a bordo conectados a través de Ethernet (ver figura 1). Este segmento de la red es interconectado a través de una LAN inalámbrica de alta potencia a las máquinas (de escritorio o servidores) que están ejecutando las tareas de computación intensiva (informática, visión, reconocimiento de voz, etc.)

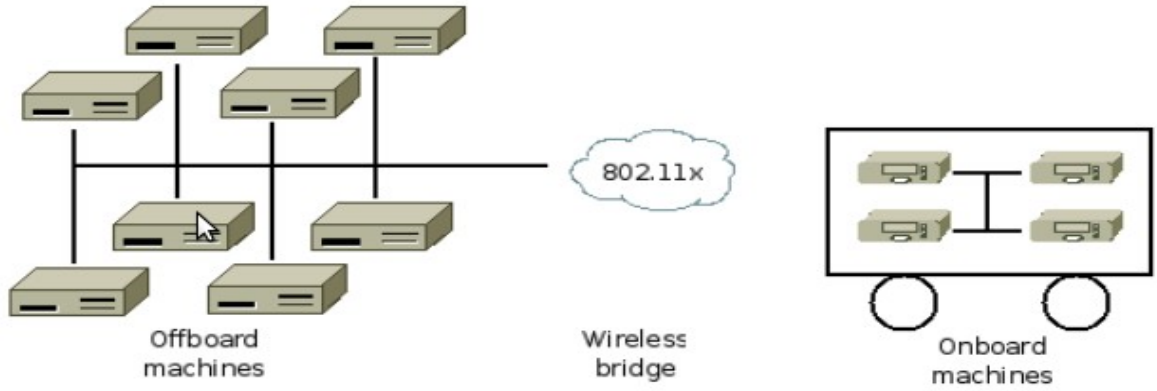


Figura 1: Arquitectura de ROS

Siendo así, un sistema construido utilizando ROS consta de un número de procesos, los que podrían estar corriendo en un número de hosts diferentes, conectados en tiempo de ejecución mediante una topología punto a punto.

Además, el hecho de que este sistema sea neutral en cuanto a los lenguajes permite que los usuarios se sientan más cómodos al programar. Actualmente son soportados cuatro lenguajes: C++, Python, Octave y LISP (Quigley, 2009). Esta flexibilidad en cuanto a lenguaje se basa en el hecho de que ROS trabaja mediante mensajes sobre conexiones punto a punto; esto es mediante un sistema de publicación y registro basado en el protocolo XML-RCP. Para este protocolo hay soporte para la mayoría de lenguajes.

En cuanto a estructura de software, es de interés notar que sus autores (Quigley, 2009)

apostaron al diseño de un microkernel con funciones específicas y a la modularización de las demás librerías y drivers.

Finalmente, es interesante destacar que este desarrollo está liberado bajo licencia BSD, la cual permite su uso para desarrollos con fines tanto comerciales como no comerciales. Este hecho permite que haya muchos desarrolladores independientes que puedan trabajar con esta herramienta sin necesidad de grandes inversiones.

3 IMPLEMENTACIÓN EN LABORATORIO DE PLANTA PILOTO

Desde el departamento de química de la UNRC se planteó la posibilidad de tener un laboratorio remoto que permita a los alumnos realizar prácticas de laboratorio a distancia e incluso poder compartir recursos, experiencias y conocimientos con docentes y alumnos de otras universidades.

3.1 Arquitectura del sistema propuesto para el laboratorio remoto

Inicialmente se propone el control de un sistema simple constituido por tanques, electroválvulas ON-OFF y cañería. Éste es un prototipo de diseño flexible que, por tener varios actuadores y sensores en forma distribuida, permitirá llevar a cabo remotamente distintas configuraciones de prácticas de laboratorio. A continuación se analiza un caso particular de estudio en donde las variables físicas de interés se estimarán a través de técnicas de procesamiento de imágenes. Además, tanto los sensores como los actuadores se conectarán a nodos implementados sobre tecnología ARM (placa BeagleBone). La cantidad de nodos a implementar podrá crecer en función del tamaño del laboratorio y de la carga de procesamiento en el nodo.

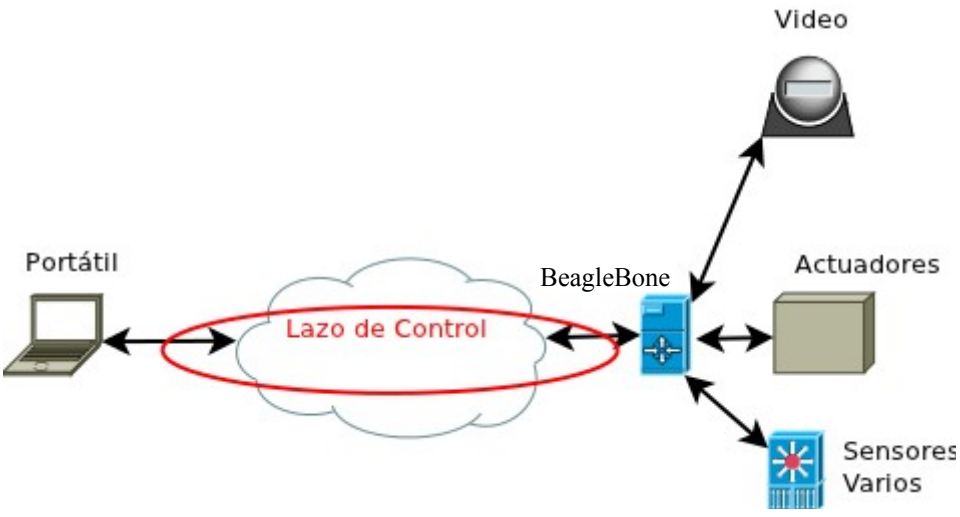


Figura 2: Sistema propuesto para laboratorio remoto en planta piloto

En la figura 2, se muestra un diagrama esquemático de la arquitectura de teleoperación propuesta. Es evidente entonces que el lazo de control se establece a través de la red, debiendo resultar éste *transparente* al usuario.

Por otra parte, para lograr la flexibilidad del laboratorio, se considerará que cada

sensor/actuador publica a tasa constante información de estado (temperatura, presión, nivel, etc) sobre un canal de datos ("tópico") al cual el controlador (lado del cliente) podrá suscribirse para efectuar y mantener un lazo de control a través de la red.

El sistema antes presentado requiere que tanto el portátil del alumno como el servidor (en planta del laboratorio) que recolecta y maneja las señales de los sensores y actuadores tengan instalado el sistema ROS. Por lo tanto, se propone utilizar tecnología compatible con Linux Ubuntu, por ejemplo amd64, x86, Arm. Para la pruebas realizadas se trabajó sobre las dos últimas arquitecturas mencionadas; en el caso de Arm se utilizó la placa Beaglebone.

3.2 Implementación y mediciones experimentales

Para probar la capacidad de respuesta y la factibilidad de implementación del sistema propuesto basado en la transmisión de datos utilizando ROS, se dispuso el siguiente escenario de teleoperación (ver figura 3):

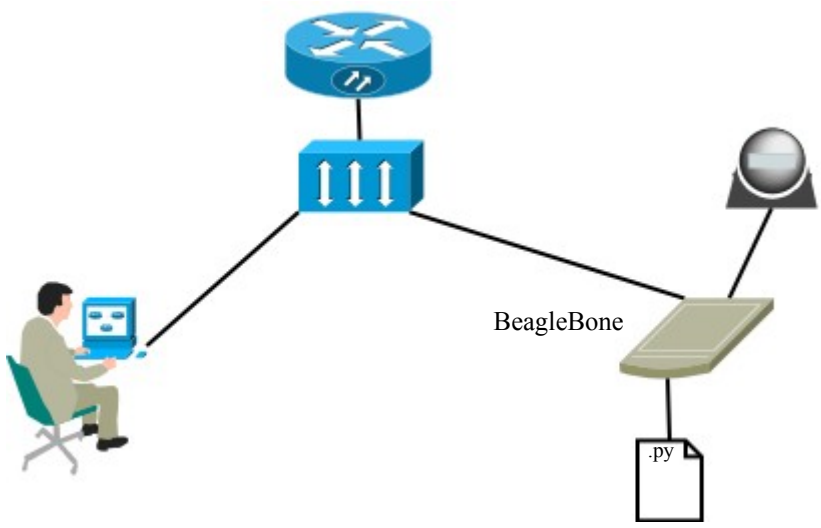


Figura 3: Sistema propuesto para pruebas iniciales de factibilidad

Puede apreciarse en la figura 3 que se utiliza una webcam de buena calidad como generadora de flujo de video y una *microcomputadora* (Beaglebone) como servidor y nodo ROS. Dentro de ésta, también se realiza el procesamiento de imagen en tiempo real y devuelve datos numéricos (emulando niveles, temperaturas, etc.). El código implementado en el nodo se realizó en Python v2.7 utilizando la librería OpenCV v2.3 y los datos publicados por el nodo consisten en la cantidad de pixeles blancos de la imagen binarizada.

También puede apreciarse en la figura que desde el lado del cliente se utiliza una PC de escritorio con un software pensado para que, vía ROS, pueda conectarse tanto al flujo de video emitido como a un canal con los datos procesados en tiempo real.

Se plantea una prueba inicial del sistema en entorno de red local con asignación fija de IP para después dar paso a pruebas sobre redes de área amplia.

3.3 Resultados

A continuación, en la figura 4 y 5, se muestran los datos publicados luego del

procesamiento de la imagen (que representa un nivel) ubicada sobre el lado derecho de la figura. También se visualizan resultados.

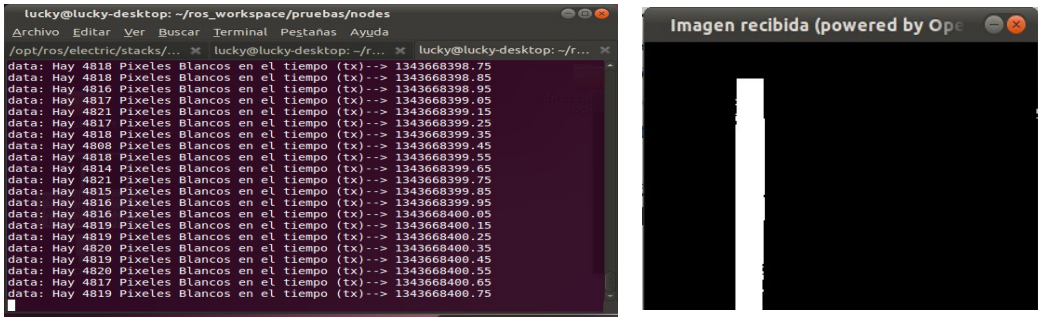


Figura 4: Resultados sobre x86

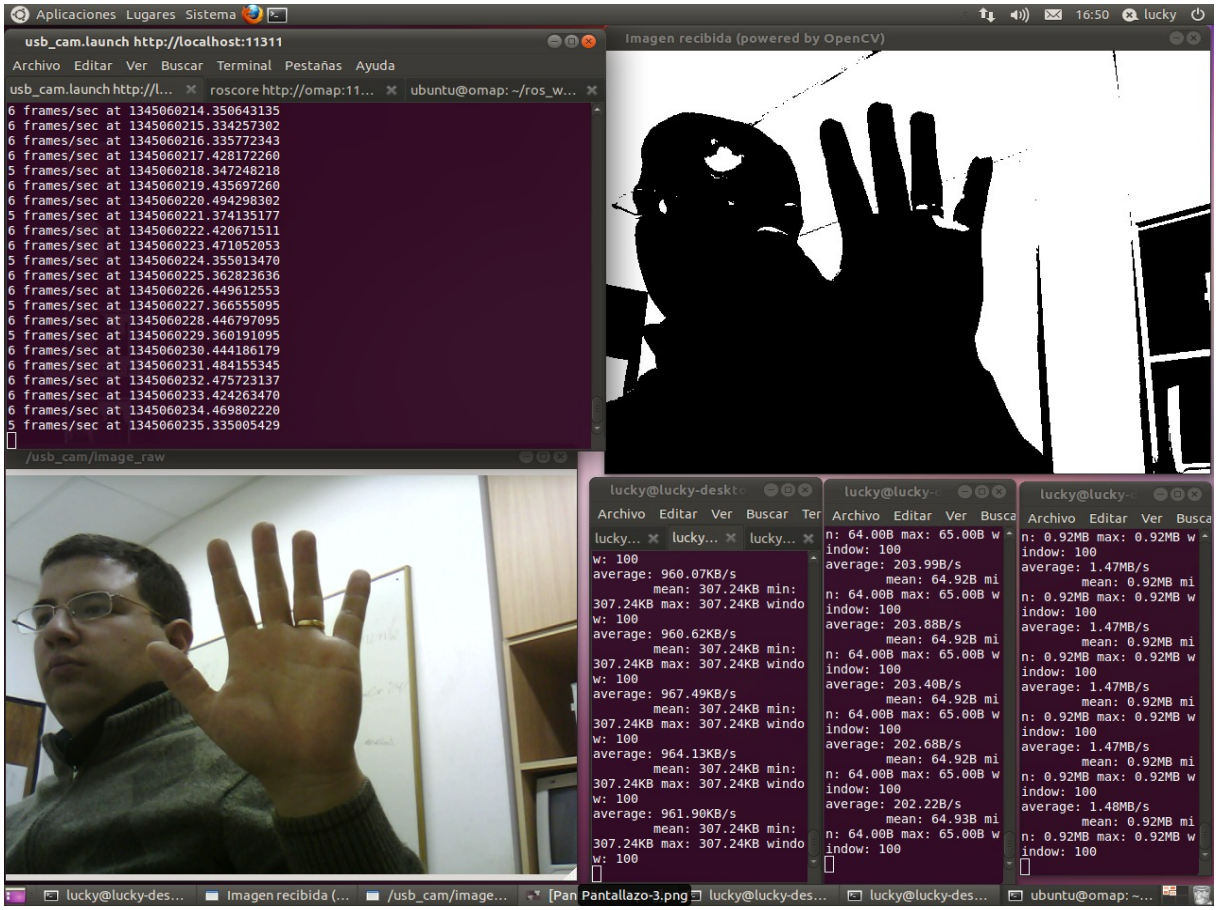


Figura 5: Resultados sobre arm cortex A8(BeagleBone)

Como puede apreciarse, se realizaron incluso algunas mediciones de consumo de ancho de banda de los tópicos a los que se hizo conexión. Resulta evidente que la transmisión de la imagen sin procesar ('image_raw') es la que requiere más recursos, mientras que los datos en sí no consumen siquiera 2 kb/s. Una gráfica que habla por sí sola sobre los consumos de ancho de banda de cada tópico se muestra en la figura 6.

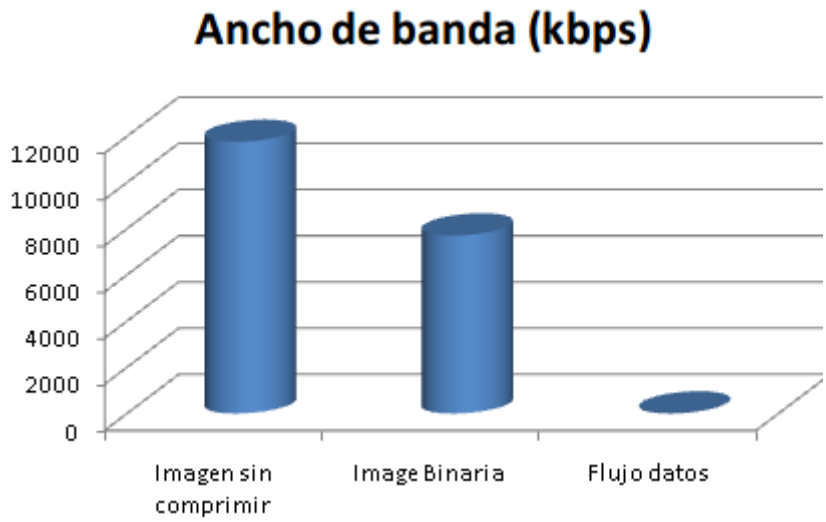
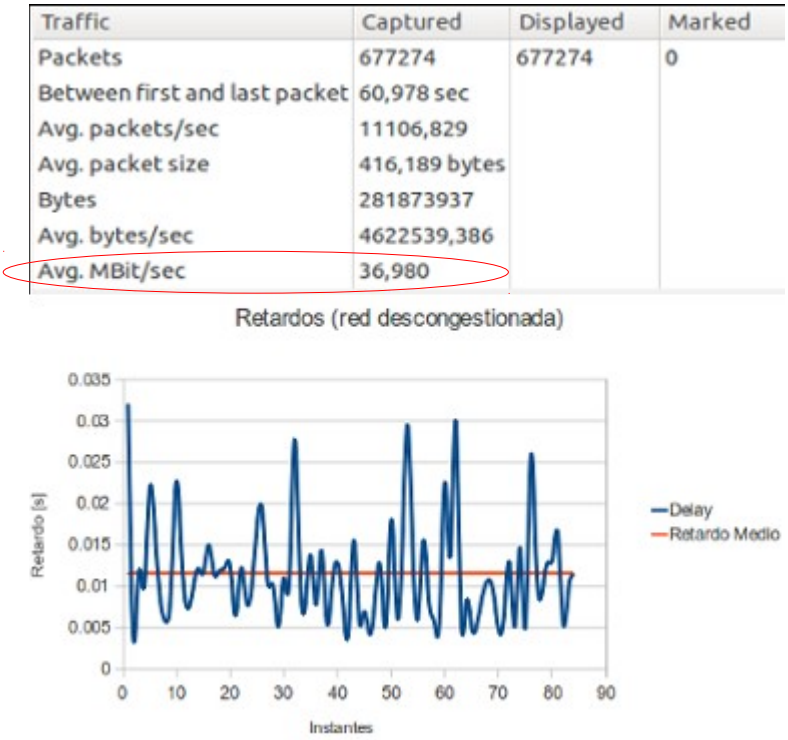


Figura 6: Comparativa anchos de banda por tópico

A continuación se trata de establecer el retardo medio y el comportamiento del mismo ante carga en la red. Cabe aclarar que para las mediciones que se comentan a continuación se sincronizó los relojes de los dispositivos (arm y x86) mediante el protocolo ntp.

Como primer paso se realiza una captura del tráfico en la red mediante el analizador Wireshark y se estima el ancho de banda consumido por el flujo de los tres tópicos de interés: La imágen en color, la imágen procesada y los datos procesados; éste se estima en unos 37 Mbps (ver Figura 7). En este caso el retardo medio se encuentra en unos 11.5 ms.



Traffic	Captured	Displayed	Marked
Packets	677274	677274	0
Between first and last packet	60,978 sec		
Avg. packets/sec	11106,829		
Avg. packet size	416,189 bytes		
Bytes	281873937		
Avg. bytes/sec	4622539,386		
Avg. MBit/sec	36,980		

Figura 7: Resultados de ancho de banda en red sin congestión

A continuacion se genera tráfico tcp con el software iperf y se realiza una captura del

comportamiento del tráfico en la red. También se estima el retardo de los paquetes del tópico "imagen procesada" como muestra del comportamiento de ROS ante un escenario de red con carga.

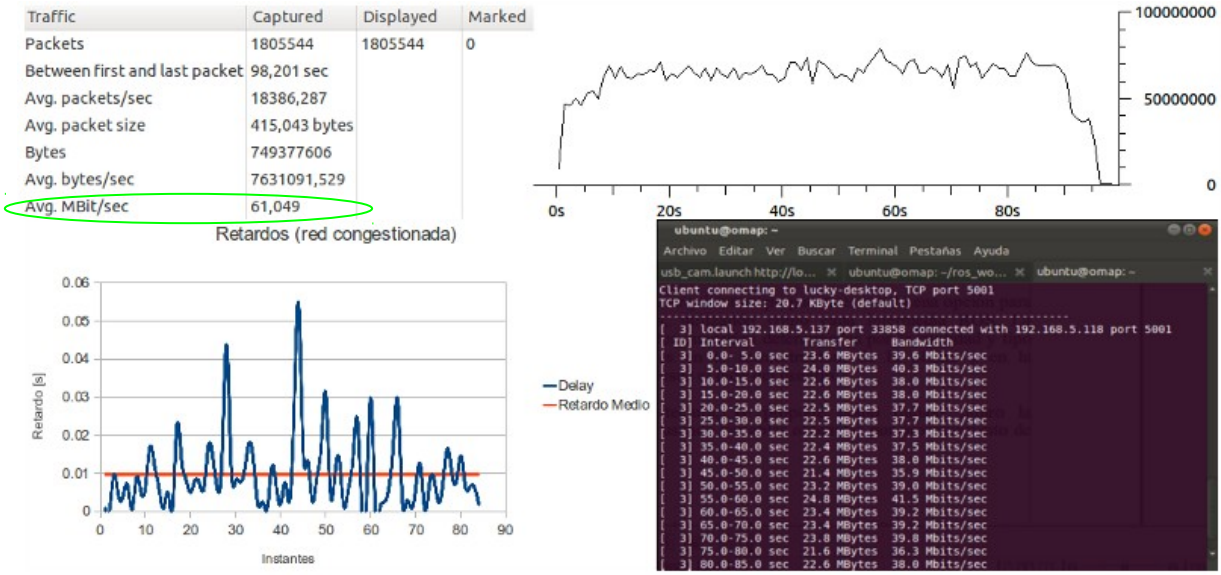


Figura 8: Resultados de ancho de banda en red sin congestión

En la figura 8 se muestra también el informe del generador de tráfico con el ancho de banda consumido por el tráfico tcp inyectado; aproximadamente 38Mbps. Se aprecia además que el retardo medio en estas condiciones no varía sobremanera con el anterior; en este caso es cercano a 10 ms.

En ambos casos, el retardo obtenido es menor que el propuesto como una cota superior para la aplicabilidad de técnicas de control anticipativas, las cuales son importantes en la implementación de laboratorios remotos; dicho umbral límite se ha estimado en 25[ms] (Guinaldo, 2010)

4 CONCLUSIONES

Los resultados obtenidos muestran que la arquitectura propuesta es una buena opción para implementar en los laboratorios remotos, utilizando como capa intermedia ROS. La flexibilidad del laboratorio y sus capacidades quedarían determinadas por la cantidad y tipo de nodos (sensores/actuadores) disponibles en la infraestructura de la planta y en la programación eficiente del lazo de control desde el lado del usuario.

El retardo medio ha rondado los 12 [ms] como promedio y hay que tener en cuenta que dicha demora no solo se debe a la red, sino al tiempo de procesamiento de la máquina, el cual se puede optimizar si se consigue buenos procesadores y el uso exclusivo y dedicado del equipo.

Siendo validado el modelo de arquitectura, se propone como trabajo futuro la implementación de lazos de control usando variables obtenidas por procesamiento de imágenes.

REFERENCIAS

Guinaldo, M., Sanchez, J., Dormido, S., A Paket-based Network Control System Architecture for Teleoperation and Remote Laboratories, 978-1-42447746-3, 2010

Hardison, J.L., DeLong, K., Bayley, P.H., Harvard, V.J., *Deploying Interactive Remote Labs Using the iLab Shared Architecture*, 978-1-4244-1970-8, 2008

Lum, M.J., Rosen, J., King, H, Friedman, D.C.W., Lendvay, T.S., Wright, A.S., Sinanan, M.N., Hannaford, B., *Teleoperation in Surgical Robotics – Network Latency Effects on Surgical Performance*, 978-1-4244-3296-7, 2009

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler,. R., Ng, A., *ROS: an open-source Robot Operating System*, 2009

Wang, F., Liu, D., *Networked Control Systems Theory and Aplications*, Springer, 2008