Implementación de Servidor para Medir el Throughput de Canales Inalámbricos Punto a Punto en la Empresa de Energía de Boyacá con el fin de Conocer la Viabilidad de Dichos Canales para la Transmisión de Datos de Video IP

Trabajo de Grado Para Título de Ingeniero Electrónico

Universidad Santo Tomás Seccional Tunja

Sergio A. Robles

Agosto de 2015.

Dedicatoria

Una dedicatoria de este arduo trabajo para Dios y mi familia, este reconocimiento es lo menos que puedo hacer por toda la ayuda que me han brindado.

Agradecimientos

Quiero agradecer a mi familia, a los ingenieros de la facultad que compartieron sus conocimientos conmigo y entre ellos particularmente al ingeniero José Ricardo Casallas por dar asesoría a este trabajo.

Exención de responsabilidad

El contenido del presente documento fue elaborado con fines netamente informativos y por lo tanto, el autor del mismo, se exime de toda responsabilidad referente a la usurpación de los derechos de autor de aquellos trabajos de otros sujetos que sirvieron de fundamento para la elaboración de este proyecto; se aclara que la información obtenida de dichos trabajos está debidamente citada dentro de este documento. También se exime de toda responsabilidad derivada del mal uso o interpretación que se le pueda dar a la información aquí contenida.

Nota de aceptación

_____

_____

_____

_____

_____

_____

_____

Firma del Presidente del jurado

_____

Firma del Jurado

_____

Firma del Jurado

Tunja, Agosto de 2015

Resumen

Gracias a las enormes ventajas que tienen las cámaras de video IP muchas empresas en la actualidad tienden a implementar este tipo de dispositivos, el problema es que son muy demandantes de ancho de banda, especialmente si se implementan sobre redes inalámbricas, por lo tanto, se tuvo el propósito de medir la velocidad de transmisión de datos (throughput) de conexiones sobre canales inalámbricos punto a punto por medio de un software desarrollado en java (lenguaje de programación) con el fin de conocer la capacidad de estos para la transmisión de video IP.

Dicho software simula el consumo de datos de un dispositivo de video que pretenda ser implementado en algún canal inalámbrico con el fin de saber si este canal es viable para soportar el dispositivo, esto sin lugar a dudas permite que se conozcan los parámetros de capacidad de los canales y por lo tanto se logre diseñar mejor la distribución de los servicios de red. Lógicamente que para el desarrollo del software se estudiaron temas como los fundamentos de programación en java, los protocolos TCP y UDP, las redes, entre otros.

El proyecto se dividió en tres etapas, la primera definió el objetivo, la segunda hizo los estudios previos y la tercera fue la construcción del programa. El programa desarrollado se ejecuta en consola y dispone de un menú de opciones que presenta dos funciones, ser cliente o servidor. Cuando se establece una conexión entre cliente y servidor el programa puede hacer dos cosas, la primera consiste en descargar archivos del servidor y la segunda en subir archivos hacia el servidor, se visualiza la velocidad para ambos casos en B/s (Bytes por segundo), lógicamente que el programa va a forzar dicha transmisión a la máxima velocidad posible. El funcionamiento del programa necesita tener instalada la máquina virtual de java con la versión 1.8 o superior, además pesa muy poco y es flexible. Los detalles del funcionamiento del programa se presentan en el capítulo de “resultados” que se encuentra más adelante.

Glosario

Definiciones

Ancho de Banda

Es el rango neto de bits y la máxima tasa de transferencia de datos exitosa en un sistema de comunicación digital.

Canal de Comunicaciones

Se considera el medio físico en el cual se transmiten o propagan las señales eléctricas, electromagnéticas, infrarrojas u ópticas de una red, dichas señales transportan el mensaje.

Cliente

Es una aplicación informática o un ordenador que consume un servicio remoto en otro ordenador conocido como servidor, normalmente a través de una red de telecomunicaciones.

Eficiencia

La capacidad del canal para poder transmitir la mayor cantidad de datos con la menor cantidad de pérdidas, en ello se tiene en cuenta diversos factores de análisis como el ancho de banda analógico, el consumo de energía y la tasa nominal y real de bps.

Fibra Óptica

Es un medio de transmisión empleado en redes de datos, es un hilo muy fino de material transparente, vidrio o materiales plásticos, por el que se envían pulsos de luz que representan los datos a transmitir.

IP

El protocolo de internet (Internet Protocol) es un protocolo utilizado para la comunicación de datos a través de una red de paquetes combinados.

Interferencia

La interferencia es un fenómeno en el que dos o más ondas se superponen para formar una onda resultante de mayor o menor amplitud.

Intermitencia

Interrupción y continuación sucesivas a intervalos regulares.

Java

Es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados.

Lenguaje de programación

Es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como las computadoras.

Máquina virtual

En informática una máquina virtual es un software que simula a una computadora y puede ejecutar programas como si fuese una computadora real. Este software en un principio fue definido como "un duplicado eficiente y aislado de una máquina física".

Modelo OSI

Es un marco de referencia o estándar en comunicaciones para la definición de arquitecturas en la interconexión de los sistemas de comunicaciones.

Paquete

Consiste en cada uno de los bloques en que se divide, en el nivel de Red, la información que se va a enviar. Por debajo del nivel de red se habla de trama de red, aunque el concepto es análogo.

Ping

Es una utilidad diagnóstica en redes de computadoras que comprueba el estado de la comunicación del host local con uno o varios equipos remotos de una red a IP por medio del envío de paquetes ICMP de solicitud y de respuesta.

Potencia

Es la cantidad de energía efectiva de una señal en el medio (espacio libre) irradiada a través de una antena.

Radio repetidor

Un repetidor es un dispositivo electrónico que recibe una señal débil o de bajo nivel y la retransmite a una potencia o nivel más alto, de tal modo que se puedan cubrir distancias más largas sin degradación o con una degradación tolerable.

Red

Conjunto de medios (transmisión y conmutación), tecnologías (procesado, multiplexación, modulaciones), protocolos y facilidades en general interconectados, necesarios para el intercambio de información entre los usuarios de la red.

Remoto

Apartado, distante, lejano.

Router

Es un dispositivo que proporciona conectividad a nivel de red. Su función principal consiste en enviar o encaminar paquetes de datos de una red a otra.

Ruido

Es toda señal no deseada que se mezcla con la señal útil que se quiere transmitir.

Tasa de transferencia

Define el número de bits que se transmiten por unidad de tiempo a través de un sistema de transmisión digital o entre dos dispositivos digitales. Así pues, es la velocidad de transferencia de datos.

Throughput

Es la cantidad de trabajo que un ordenador puede hacer en un período de tiempo determinado. Históricamente, ha sido una medida de la eficacia comparativa de equipos comerciales de gran tamaño que ejecutan varios programas simultáneamente.

Trama de datos

Es una unidad de envío de datos. Es una serie sucesiva de bits, organizados en forma cíclica, que transportan información y que permiten en la recepción extraer esta información. Viene a ser el equivalente de paquete de datos o Paquete de red, en el Nivel de enlace de datos del modelo OSI.

Servidor

Es un tipo de software que realiza ciertas tareas en nombre de los usuarios. El término servidor ahora también se utiliza para referirse al ordenador físico en el cual funciona ese software, una máquina cuyo propósito es proveer datos de modo que otras máquinas puedan utilizar esos datos. Un servidor sirve información a los ordenadores que se conecten a él. Cuando los usuarios se conectan a un servidor pueden acceder a programas, archivos y otra información del servidor.

Lista de Figuras

Lista de Tablas

Contenido

Capítulo 1

Introducción

Los servicios multimedia demandan muchos datos sobre la red por la cantidad de detalles que manejan, lo que hace que día tras día las empresas que trabajan para interconectar redes entre continentes, países o ciudades se preocupen por el manejo de una variable fundamental; el ancho de banda (Margarit, 2003). El ancho de banda es el rango neto de bits por segundo en la transferencia de datos de un enlace físico de comunicación, por lo tanto, medir esta variable para las empresas de comunicaciones es una necesidad apenas natural.

Una de las técnicas para conocer el ancho de banda por un usuario sin conocimientos técnicos, puede ser con un software informático que realice transferencia de archivos. La implementación de un servicio que transmita información tangible para el usuario como los archivos, y con ello pueda observar la capacidad de transmisión de datos de su conexión, le representa un mejor entendimiento acerca del ancho de banda (Murillo Morera, 2010).

La intención del presente documento es demostrar la medición eficaz del ancho de banda de una conexión inalámbrica mediante un servicio común como la transferencia de archivos. Para la creación de dicho software existen diversos lenguajes de programación en los que se puede desarrollar el programa para implementar la prueba, entre los lenguajes de propósito general más recomendados están C y C++, Basic, .Net, C# o java. De todos los anteriores, el lenguaje de preferencia para el desarrollo fue java, lo que no significa que los otros lenguajes sean menos buenos que este, sino que java se adapta mejor a los gustos de trabajo por ser “un lenguaje multiplataforma, flexible y portable” (Schildt, 2007, pág. 29). A lo largo de este documento se conocen los detalles del proceso de creación del software.

Capítulo 2

Justificación

¿Qué ventajas económicas ofrece el proyecto?

El proyecto no necesita comprar equipos especiales ni materiales para su implementación, solo requiere de una PC que esté conectada en la red y que funcione como servidor de referencia y otro PC que sea el cliente para hacer la prueba desde el punto donde se quiera diagnosticar la conexión. La disposición de dos PC es apenas normal dentro de cualquier empresa, además de que el servidor no necesita estar permanentemente activo sino solo el tiempo que dure la prueba.

La medida comprende los dos puntos de conexión sin importar la cantidad de segmentos que tenga la ruta, por lo tanto no sería necesario acceder segmento tras segmento a las interfaces de los radios en caso de que la red sea muy extendida, sino que solo se necesita de una prueba, lo que ahorraría los costos de tantos desplazamientos, el desgaste físico y tiempo. La ejecución y configuración del programa es muy sencilla y no necesita del manejo de expertos, esto facilita aún más su implementación.

¿Qué beneficios académicos y operativos deja el proyecto tras su terminación?

El proyecto abre la puerta para que futuros alumnos opten por implementar soluciones que mejoren la calidad del servicio de la señal inalámbrica de internet que suministran los access point de sus universidades o academias, también permite que los trabajadores de la empresa de energía que operan en la red de datos dispongan de una herramienta fácil de usar e instalar sin depender de la asesoría de personal experto.

¿Cuál es la diferencia del proyecto con los medidores de velocidad o test de velocidad que se usan desde internet?

En esencia el proyecto trata de emular a los medidores de internet ya que ambos consisten en la comunicación entre dos aplicaciones en TCP/IP. El proyecto funciona cuando se comunican dos equipos, uno con la aplicación funcionando en modo cliente y el otro en modo servidor, los medidores de internet funcionan igual, el cliente es el usuario que carga un complemento web en su navegador y este se comunica con la aplicación que tiene programada internamente el servidor de la página web y con ello se realiza la prueba de velocidad.

La gran diferencia es que el proyecto se implementa en una red LAN, aunque cubra largas distancias por usar radio enlaces no deja de ser LAN porque es privada, solo conecta las dependencias de una empresa y está bajo la administración de una sola persona, es pequeña en términos administrativos y físicos también, en cambio, los test de velocidad de un servidor web en internet (la red WAN más grande del mundo) están disponibles para todo el mundo.

Capítulo 3

Planteamiento del Problema

Es normal que una compañía grande tenga muchas dependencias en un departamento, país, o el resto del mundo. Por lo tanto también es normal que tenga canales de comunicación propios para operar entre sus dependencias. La larga distancia y el costo por kilómetro de un canal cableado hace muy elevado su precio, por lo tanto, las compañías pueden reducir gastos usando canales inalámbricos (punto a punto y punto a multipunto), el problema es que estos no aseguran altas prestaciones y son muy vulnerables a cambios en el medio ambiente.

La situación empeora aún más cuando a través de ellos se utilizan servicios especiales como la video-vigilancia remota (video IP en largas distancias con fines de seguridad) debido a que consume bastante ancho de banda de acuerdo con la resolución de las cámaras (puede ser superior a 2000 kbps por dispositivo), lo que aumenta la carga de tráfico y puede generar bloqueos en los radio-enlaces que no las soporten, por lo tanto es necesario conocer el ancho de banda de estos canales para evitar problemas de saturación de tráfico por falta de conocimiento de las capacidades de los mismos.

Al conocer el ancho de banda, los radio-enlaces se podrían clasificar de acuerdo al tipo de servicio que soporten y esto les permite a los operarios rediseñar una topología más efectiva o también les sirve para reconfigurar el canal inalámbrico y mejorar el throughput, todo lo anterior tiene la finalidad de aumentar la calidad de la red inalámbrica para la transmisión de video IP.

Capítulo 4

Objetivos

General

Diseñar e implementar un programa que sirva para medir el throughput de un canal de comunicaciones inalámbrico mediante el protocolo TCP/IP.

Específicos

- Evitar instalaciones infructuosas de cámaras IP y en las subestaciones que se interconectan inalámbricamente con radio-enlace de bajo rendimiento.
- Crear una herramienta alternativa, fácil y eficaz para la medición de ancho de banda sobre canales inalámbricos, en este caso para la Empresa de Energía de Boyacá con el fin de estimular en ella, el apoyo hacia los profesionales de la región de Boyacá.
- Definir los requisitos mínimos y los alcances del proyecto.
- Medir el throughput en un canal cableado patronado con el software del proyecto.
- Medir el throughput en un canal inalámbrico de la Empresa de Energía de Boyacá con el software del proyecto.
- Explicar en un documento toda la información acerca del desarrollo del proyecto.

Capítulo 5

Marco teórico

Este capítulo corresponde al estado del arte que expone los criterios analíticos como punto de partida de la solución del problema llevado a cabo en este trabajo. Por un lado, describiremos los métodos de medida de ancho de banda y, en particular, nos centraremos en las dos familias de técnicas, las basadas en trenes de paquetes y las basadas en transferencia de ficheros. Se expondrán los inconvenientes que tienen las mediciones de ancho de banda con tráfico interferente tanto en TCP como en UDP y luego se describirán las características y el funcionamiento de TCP que nos lleve a comprender la razón por la cual este protocolo presenta una degradación alta con el exceso de tráfico en comparación con UDP. Por otro lado, se describirán las técnicas estadísticas usadas en el análisis de la medición del ancho de banda y se justificara su implementación. Por último se explican algunos programas que sirven para medir ancho de banda y fueron usados como ejemplo para orientar el desarrollo del proyecto.

Métodos de medida del ancho de banda

5.01. Métodos basados en trenes de paquetes. El primer gran grupo de técnicas de medida del ancho de banda que vamos a describir es el de las basadas en pares (o trenes) de paquetes. La Fig. 5.1 ilustra el principio de funcionamiento básico de estos métodos. Al enviar dos paquetes entre dos puntos de la red, el tiempo que transcurre entre la llegada del último bit del primer paquete y la llegada del último bit del segundo paquete, viene determinado por la velocidad de transmisión del enlace cuello de botella (es decir, del enlace con menor velocidad de transmisión de todo el camino) (Jacobson, 1988).

En el primer tramo los paquetes se separan tanto como el tiempo que tarda en transmitirse el segundo paquete (la longitud del paquete entre el ancho de banda), en el segundo

tramo, cuyo ancho de banda es menor, los paquetes se separan más, mientras en el tercer tramo, aunque el ancho de banda sea de nuevo mayor, los paquetes siguen estando dispersos tanto tiempo como lo estaban tras el segundo tramo, por tanto, la dispersión entre dos paquetes viene dada por el tiempo en que tarda en transmitirse el segundo paquete por el enlace cuello de botella (Keshav, 1991).

Figura 5.1 Representación del cuello de botella de una conexión entre dos puntos

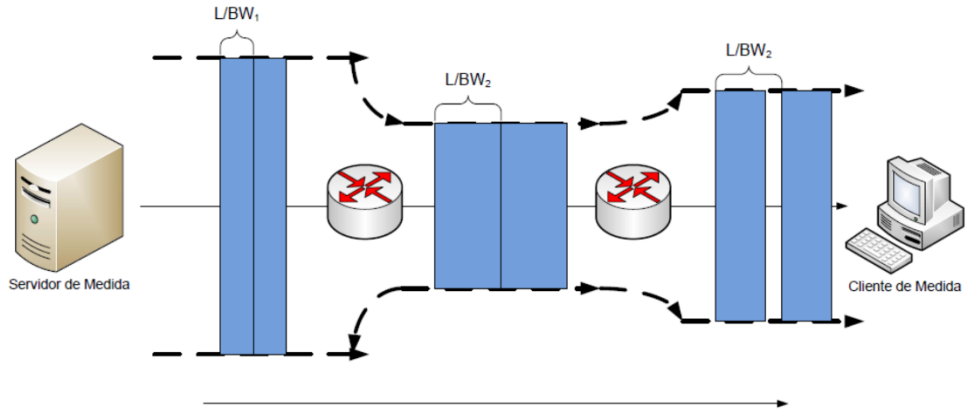


Figura 5.1 El canal de banda estrecha del medio es el tramo de la conexión con la más baja capacidad de transmisión de datos y los dos segmentos en azul indican el tren de paquetes de la medida.

- $BW =$ ancho de banda del enlace cuello de botella entre dos puntos de la red.
- $t_i =$ tiempo de llegada del último bit del i-ésimo paquete.
- $L =$ longitud del paquete.
- $\Delta_i = t_{i+1} - t_i$ tiempo entre llegadas entre el paquete i y el paquete i + 1.
- N es el número de paquetes del tren. En el caso del par de paquetes $N = 2$.

Aprovechando el fenómeno de dispersión entre dos paquetes, podemos estimar el ancho de banda de un enlace del siguiente modo:

- Desde un extremo, enviamos dos paquetes, de tamaño L, a la máxima velocidad.

- Si los paquetes son enviados de manera consecutiva, el tiempo de llegada nos proporciona el ancho de banda del enlace cuello de botella, como el cociente del tamaño del paquete y el tiempo de transmisión.

  Así, el ancho de banda del enlace cuello de botella, viene dado por la siguiente fórmula:

$$BW = \frac{L}{t_2 - t_1} = \frac{L}{\Delta_1}$$

  Las principales ventajas de este método son varias, a saber:

- Rapidez. La duración de la medida es igual al tiempo de transmisión de dos paquetes. Por ejemplo, si medimos un único enlace cuyo ancho de banda es de 10 Mbps con paquetes de longitud 1500 Bytes, la medida tarda $2x\frac{1500Bx8b}{10x^{10^6 b/_s}} = 2.4\ ms$.

- No intrusividad. Es decir, los paquetes de medida, al ser sólo dos, no interfieren en el rendimiento de las conexiones existentes en los enlaces que se pretenden medir. Esto es muy útil, ya que no es necesario interrumpir ni degradar el funcionamiento del enlace para poder medir su ancho de banda.

Sin embargo, este método tiene una gran desventaja relacionada con el tráfico cruzado (tráfico perteneciente a otras aplicaciones que se cuela entre los dos paquetes de prueba), en el caso de que al menos un paquete perteneciente a otra aplicación se colara entre los dos paquetes de la medida, el tiempo de llegada Δ aumentaría y, por tanto, el ancho de banda estimado sería inferior al real (Paxson, 1996). Tal y como podemos observar en la Fig. 5.2.

Figura 5.2 Dispersión del par de paquetes cuando hay tráfico cruzado

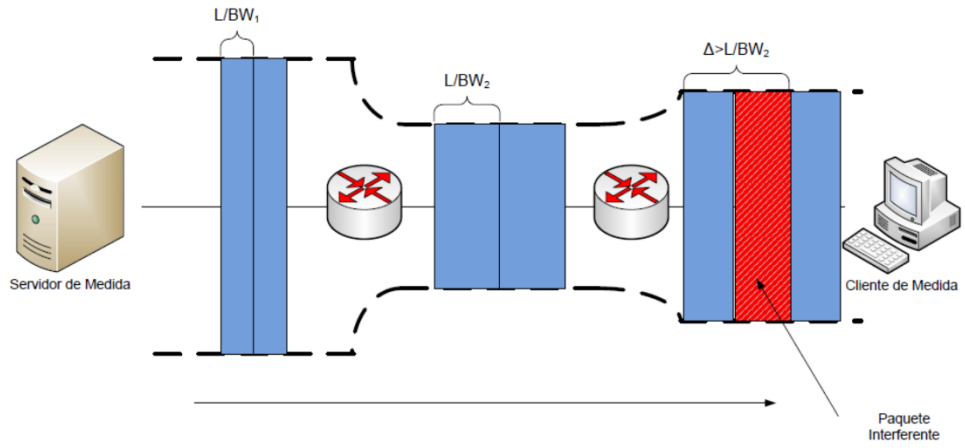


Figura 5.2 El segmento de color rojo de la parte derecha corresponde a un paquete perteneciente a otras aplicaciones y que se cola en medio del par de paquetes de la medida.

Para disminuir esta sensibilidad al tráfico interferente es necesario aumentar el número de paquetes de prueba (C. Dovrolis, 2004). Esta mejora de la técnica es lo que se conoce como tren de paquetes. Su funcionamiento es el siguiente:

- En lugar de enviar dos paquetes, enviamos un tren de N paquetes, de tamaño L, a la máxima velocidad.
- Desde el receptor, se calculan todos los tiempos entre llegadas de paquetes consecutivos del tren, a saber: $\Delta_1, \Delta_2, \Delta_3,..., \Delta_{N-1}$ con $\Delta_i = t_{i+1} - t_i$.
- Se estima el ancho de banda usando el mínimo de los Δ_i:

$$BW = \frac{L}{min_{i=1,...,N-1}\Delta_i}$$

De este modo, la medida sólo se verá afectada por el tráfico interferente cuando haya paquetes de otras aplicaciones entre cada uno de los $N - 1$ pares de paquetes del tren.

Figura 5.3 Dispersión del tren de paquetes cuando hay tráfico cruzado

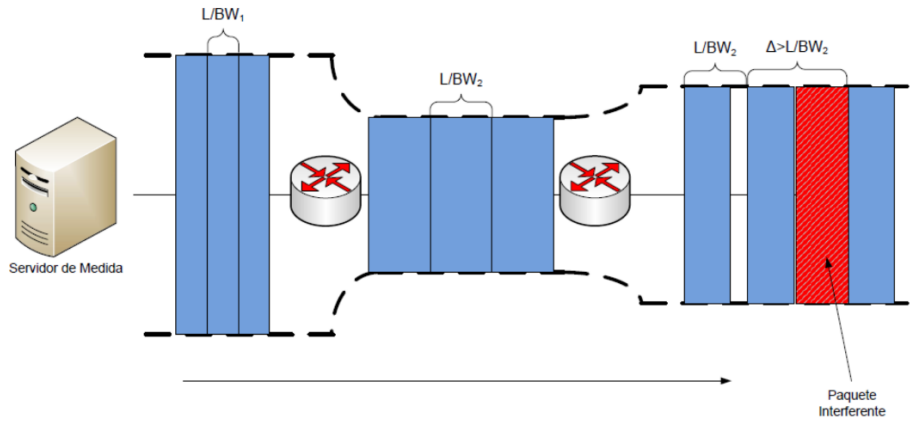


Figura 5.3 El segmento de color rojo de la parte derecha corresponde a un paquete perteneciente a otras aplicaciones y que se cola en medio del tren de paquetes de la medida.

Por ejemplo, en la Fig. 5.3 podemos observar como un tren de tres paquetes no se ve afectado por un único paquete interferente. El paquete interferente aumenta el interarrival entre el paquete 1 y el paquete 2. Sin embargo, no afecta al interarrival entre el paquete 2 y el paquete 3, con el cual podríamos obtener el ancho de banda del enlace de cuello de botella correctamente. Hay que tener en cuenta que la longitud del tren no se puede hacer arbitrariamente grande, ya que el tráfico de la medida sería demasiado intrusivo, llegando a deteriorar el servicio de la línea que se quiere medir.

5.02. Métodos basados en transferencias de ficheros. La idea del método es sencilla: medir cuánto tarda la transferencia de un fichero de tamaño conocido. De este modo, estimamos el ancho de banda del enlace como el cociente entre el tamaño del fichero y el tiempo que tarda en ser transmitido (ETSI, 2000). Dependiendo del nivel en el que sea medido (nivel físico, nivel de red, nivel de transporte o nivel de aplicación), nos proveerá el ancho de banda percibido en cada uno de esos niveles.

Por ejemplo, si el enlace a medir es un ADSL de 6 Mbps (a nivel físico) pero nosotros medimos el tiempo de descarga de un fichero a través de la web, la velocidad estimada será a nivel de aplicación y obtendremos un ancho de banda estimado menor de 6 Mbps (debido a las cabeceras de los protocolos inferiores: HTTP, TCP, IP, Ethernet). El método de referencia basado en esta técnica es el descrito en la guía de recomendación ETSI, El método es el siguiente:

- Se genera un fichero de contenido aleatorio de tamaño 8 veces al ancho de banda teórico del enlace a medir. Por ejemplo, si queremos medir un enlace de 1 Mbps, el fichero tiene que tener $8 \times 10^6\ bits = 10^6\ Bytes$.
- Se transfiere el fichero generado desde un servidor de referencia hasta el ordenador conectado al enlace que se quiere medir. Se mide el tiempo de la transferencia, y se estima el ancho de banda como:

$$BW = \frac{S}{t_{fin\ descarga} - t_{inicio\ descarga}}\ [bps]$$

Donde S es el tamaño del fichero en bits, $t_{inicio\ descarga}$ es el instante de tiempo (en microsegundos) en el que el equipo cliente introduce en la red toda la información necesaria para requerir la descarga del fichero de pruebas y $t_{fin\ descarga}$ es el instante de tiempo (en microsegundos) en el que el equipo cliente recibe el último bit del fichero solicitado. Merece la pena mencionar algunos matices necesarios para el buen funcionamiento del método, a saber:

- Es necesario usar algún algoritmo de resumen, como SHA-1 o MD5, que permita verificar la integridad y corrección del fichero. El tiempo consumido en esta verificación no debe ser contado como parte del tiempo de descarga.
- El contenido del fichero debe ser aleatorio para evitar el uso de algoritmos de compresión y optimización por parte del servidor web desde el que se hace la descarga, que reducirían el tamaño del fichero.

Esta técnica tiene una serie de claros inconvenientes, a saber:

- Tiempos más altos de medida: En el caso de que el enlace funcione a la velocidad esperada, la transmisión del fichero tarda en torno a 8 segundos y, en caso de degradación de la calidad, este tiempo aumenta. Hay que recordar que una medida con la técnica de pares de paquetes de un enlace de 1 Mbps dura del orden de los milisegundos.
- Intrusividad: Durante todo el tiempo que dura la medida, la descarga satura el enlace, lo que provoca la degradación de las posibles conexiones existentes.
- Alta sensibilidad al tráfico interferente: Al tratarse de una transferencia TCP (en particular HTTP), el throughput de la conexión se adapta al ancho de banda disponible del enlace. Así, el ancho de banda del enlace será compartido por todas las conexiones TCP concurrentes, reduciendo el ancho de banda medido. La Fig. 5.4 muestra de manera simplificada el comportamiento para los casos en los que hay ninguna, una y dos conexiones concurrentes(a, b y c, respectivamente).

Figura 5.4 Adaptación del ancho de banda al tráfico cruzado

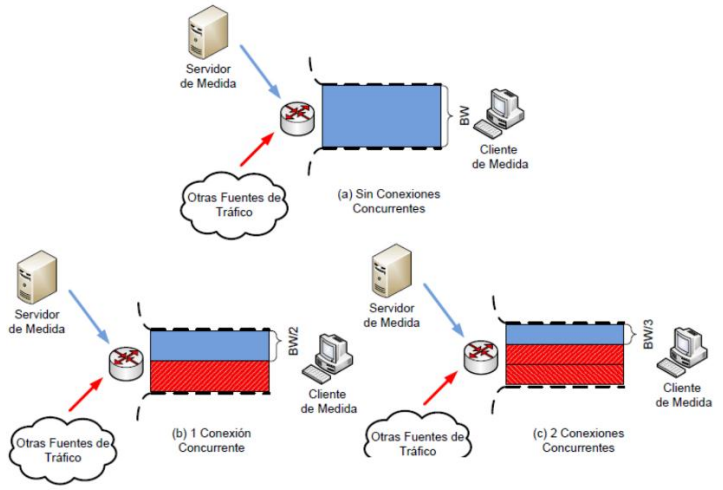


Figura 5.4 En la parte superior se observa la capacidad del canal sin conexiones concurrentes. En la parte inferior izquierda y derecha se observa como se reduce la capacidad del canal con una y dos conexiones concurrentes.

El comportamiento de flujos TCP en paralelo ha sido ampliamente estudiado en la literatura (T.J. Hacker, 2002) (E. Altman, 2006) (D. Lu, 2005). El mecanismo para evitar la congestión que implementa TCP provoca que la tasa de transferencia del flujo se vea afectada por las pérdidas de la siguiente medida (J. Padhye, 2000):

$$B = \frac{MSS}{RTT}\sqrt{\frac{k}{p}}$$

Donde MSS es el tamaño máximo de segmento, RTT es el retardo ida y vuelta, p es la probabilidad de error y k es una constante que depende de la implementación concreta de TCP. Al aumentar el tráfico interferente, aumentan las pérdidas y, por tanto, se degrada la tasa de transferencia del flujo TCP. Las Fig. 5.5 y 5.6 muestran un ejemplo de degradación de la tasa de transferencia que sufre un flujo TCP por la presencia de otros flujos TCP concurrentes y por la presencia de tráfico cruzado UDP, respectivamente.

Figura 5.5 Degradación de la tasa de transferencia de un flujo TCP

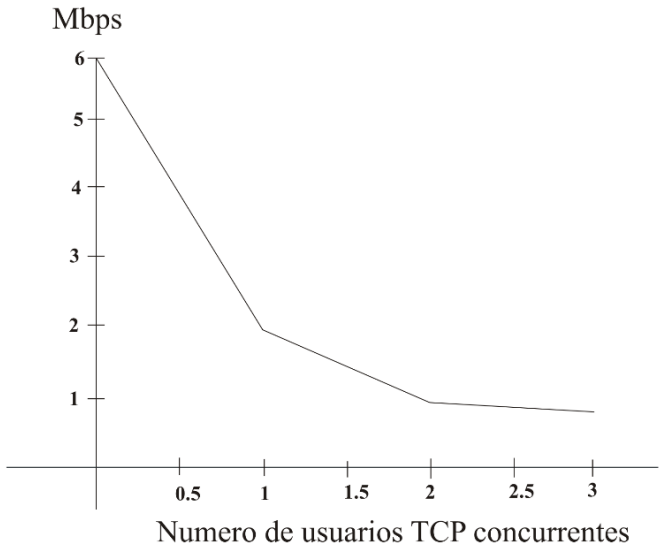


Figura 5.5 Diferencia de la degradación del rendimiento de un canal con TCP a medida que incrementa el numero de usuarios.

Figura 5.6 Degradación de la tasa de transferencia de un flujo UDP

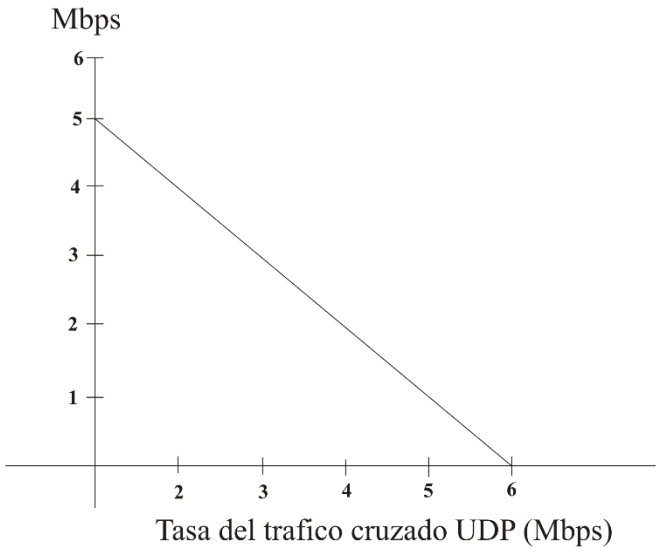


Figura 5.6 Diferencia de la degradación del rendimiento de un canal con UDP a medida que incrementa el trafico cruzado UDP.

Protocolo TCP

El Protocolo de Control de Transmisión (TCP abreviado en inglés) es un protocolo de capa superior a la IP. “La pregunta que surge es ¿Por qué necesitamos dos protocolos, IP y TCP?, mientras IP transmite datos entre computadoras individuales en Internet, las transferencias TCP lo hacen entre dos aplicaciones reales que se ejecutan en estos dos equipos, por lo tanto, IP se utiliza para la transferencia de datos entre ordenadores particulares” (James & Bramante, 2009). Una dirección IP es la única dirección de la interfaz de red de un ordenador, mientras que TCP utiliza un número de puerto como la dirección para discriminar la aplicación en concreto que va a recibir dicha información. Si tuviéramos que comparar esto con un sistema postal estándar, la dirección IP sería la dirección de la construcción y el número del puerto (la dirección en TCP) sería el nombre de un residente actual del edificio.

Figura 5.7 Diagrama físico de una comunicación a dos vias en TCP

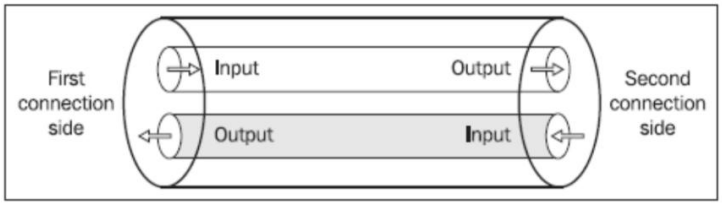


Figura 5.7 Cable que representa el medio fisico para la transmisión de los datos en TCP, que lo hace en full-duplex.

Una aplicación que utiliza TCP no tiene que preocuparse si los datos se pierden durante la transferencia, o si estos han sido modificados por un error de transferencia, debido a que TCP pide confirmación de cada segmento de información que llega al destino, por esta razón produce una sobrecarga significativa a medida que se incrementa el número de conexiones TCP tal como se presentó previamente en la Fig. 5.5.

5.03. Cabecera de un segmento TCP. La estructura de un segmento TCP se ilustra en la Fig. 5.8.

Figura 5.8 Segmento TCP

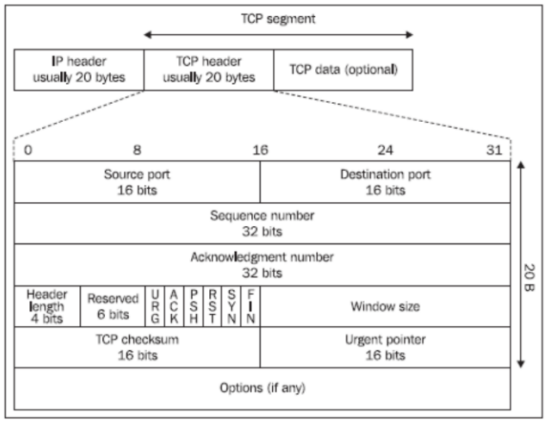


Figura 5.8 La trama TCP se ilustra con niveles en vertical debido a que ocupa mucho espacio con visualización horizontal.

Los campos del segmento TCP a saber:

- El puerto de origen, es el puerto de la fuente del segmento TCP, mientras que el puerto de destino es el puerto del destino del segmento TCP. Las cinco entradas; puerto de origen, puerto de destino, dirección IP de origen, dirección IP de destino, y el protocolo TCP, identifican de forma inequívoca la conexión particular en Internet en cualquier momento dado.
- El número de reconocimiento, expresa el número del siguiente Byte que el destino está listo para aceptar. En otras palabras, el destino confirma que recibió correctamente todo hasta el número de reconocimiento menos uno.
- La longitud de cabecera, especifica la cantidad de Bytes de la cabecera del segmento TCP en múltiplos de 32 bits (4 bytes), similar al formato de los encabezados IP.
- El tamaño de la ventana, especifica el incremento máximo de la secuencia de segmentos que se sigue aceptando el destino.
- El Puntero urgente, es válido sólo si el indicador URG está establecido. Esto indica el último byte de los datos urgentes.

Los indicadores siguientes corresponden al campo de la bandera del segmento:

- URG, bit que indica si el segmento TCP contiene datos urgentes.
- ACK, bit que indica si el Segmento TCP corresponde a una trama de confirmación de recibido válida (necesario en todos los segmentos, excepto el primer segmento con el que el cliente establece la conexión).
- PSH, esta bandera significa la función de impulso. Por medio de esta función, TCP permite a una aplicación especificar que los datos deben ser enviados inmediatamente. Cuando una aplicación con TCP solicita enviar datos, TCP debe enviar los datos que ha

acumulado sin espera para llenar el segmento. El Segmento TCP enviado está marcado por la bandera PSH.

- SYN, esta bandera sincroniza los números de secuencia y hace que la fuente empiece una nueva secuencia de conteo. En otras palabras, el segmento TCP contiene el número de secuencia del primer byte enviado (ISN).
- RST bit que indica que la conexión TCP se ha reseteado.
- FIN, sin más datos del remitente. Como en la transferencia de archivos, el indicador FIN sería el final del archivo. Debido a que TCP es una conexión completamente dúplex, la bandera FIN hará que el cierre de la conexión sea sólo en una dirección. En este sentido, no serán enviados más segmentos TCP con el indicador PSH (con excepción de la posible retransmisión).

5.04. Funcionamiento. El cliente genera un número aleatorio entre 0 y $2^{32} - 1$ que utiliza como número de secuencia inicial (ISN). En nuestro caso, hemos generado ISN = 145.165.778. El bit SYN se activa en el primer segmento TCP porque el cliente justo ha creado un número de secuencia inicial y que se encuentra en el campo de la cabecera correspondiente al número de secuencia. Durante esta conexión, el número de secuencia se incrementará de acuerdo a la cantidad de bytes enviados segmento tras segmento y también porque no se puede repetir el mismo número de nuevo si el lado transmisor no recibió confirmación. Por lo tanto, el cliente no puede establecer el mismo indicador SYN en cualquier otro segmento TCP durante la conexión.

Figura 5.9 Establecer una conexión TCP

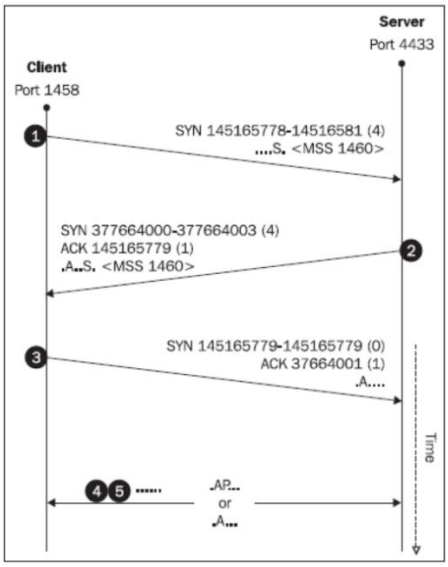


Figura 5.9 Proceso ilustrado del inicio de una conexión en TCP donde se producen confirmaciones del lado destino.

El segmento 1 inicia el proceso de la comunicación TCP, y por lo tanto no puede confirmar los datos recibidos, para ello el segmento 2, devuelve una respuesta con el campo del número de confirmación del recibido más uno, aunque este número no tiene un significado válido en toda la comunicación sino solo en el saludo protocolario (siendo rellenado con un cero binario en los segmentos siguientes) y por lo tanto el bit de este segmento ACK es 0 (el bit ACK se activa en cada segmento TCP hasta el fin de la conexión solo si ya se estableció el saludo y se inició la comunicación).

El segmento 2 tiene el bit SYN activo y el ACK inactivo, es muy específico para el saludo. Esta modalidad sirve de diagnóstico con el primer segmento de una conexión TCP. Si se desea prohibir la conexión en algunos clientes desde una dirección determinada, todo lo que se hace es filtrar todos los segmentos TCP de esa dirección que tengan el bit SYN activo y el ACK inactivo juntos. En este caso, el atacante no tiene ninguna oportunidad. Este mecanismo se utiliza a menudo para proteger una intranet de la Internet. Todo el proceso de inicio de la conexión se

ilustro en la Fig. 5.9. Una vez iniciada la comunicación se inicia un intercambio de segmentos TCP entre cliente y servidor, tal como se ilustra en la Fig. 5.10.

Figura 5.10 Intercambio de datos en TCP

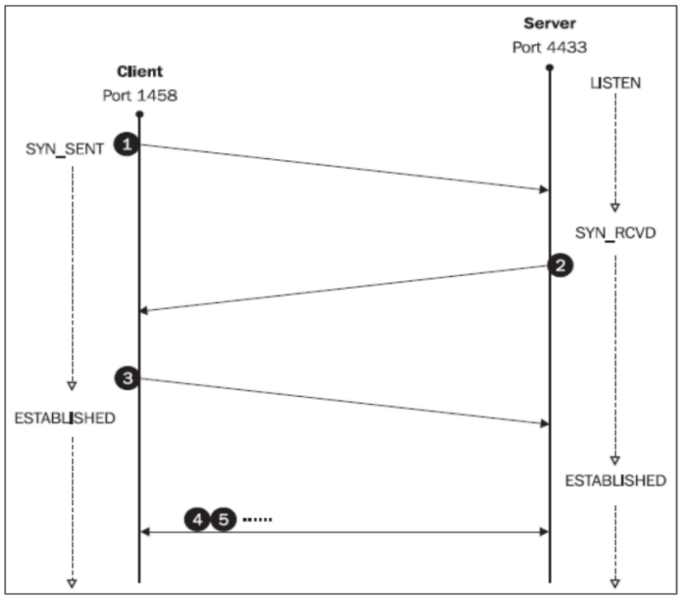


Figura 5.10 Proceso ilustrado del intercambio de segmentos durante la sesión de comunicación TCP.

Cuando la conexión está en el proceso de intercambio de datos, el cliente envía segmentos al servidor y este le devuelve una confirmación de cada uno sumando uno al número de confirmación del segmento del cliente. Cuando el cliente recibe la respuesta con el número de confirmación con la suma hecha entiende que el segmento llego bien y se dispone a enviar el siguiente, siempre respetando el orden de los datos, cuando el cliente no recibe la confirmación de algún segmento, lo reenvía las veces que sea necesario hasta recibir la confirmación pero sin exceder los 200ms porque si no se produce una desconexión. Por tal motivo, la sobrecarga de información del protocolo TCP hace de este ser muy demandante de tráfico y por ello, el exceso de las conexiones concurrentes pueden degradar los canales de comunicación, inalámbricos especialmente.

Métodos de análisis estadísticos aplicados en la medición del ancho de banda

5.05. Media. De acuerdo con (Guerra Bustillo, 2011), la media aritmética (o promedio aritmético) de un conjunto $X_1 \ldots X_n$ de n valores de una variable X se define como:

$$M(x) = \frac{X_1 + X_2 + \cdots X_n}{n} = \sum_{i=1}^{n} \frac{X_i}{n}$$

Empleando la media aritmética para conocer el valor de tendencia central en una transferencia de archivos a través de un canal inalámbrico, se calculan las muestras de la Tabla 1.

Tabla 1
Muestras de velocidad de una conexión inalámbrica ficticia

Medidas de Velocidad (MBps)				
7.9	8.1	8.43	8.32	8.38
8.2	8.26	8.36	8.4	8.32

La Tabla 1 se representa en un plano cartesiano, que se muestra en la Fig. 5.11.

Figura 5.11 Muestras de Velocidad de una Transferencia de Archivos en una Conexión Inalámbrica ficticia

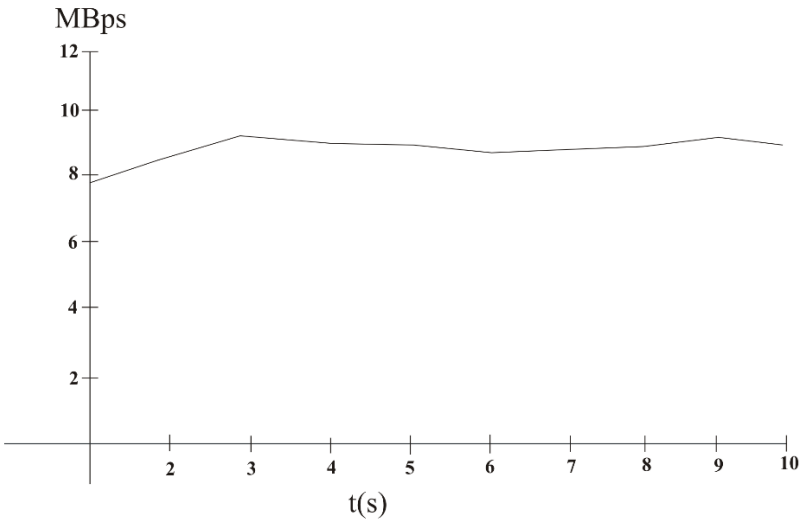


Figura 5.11 Velocidad medida en Bytes, de una transferencia de archivos durante 10 segundos.

Para comparar el efecto del cálculo de la media aritmética en dos conjuntos de muestras diferentes, se calcula la media con las muestras de la Tabla 2.

Tabla 2
Muestras de velocidad de una conexión inalámbrica ficticia No 2

Medidas de Velocidad (MBps)				
6.2	10.3	6.56	10.32	6.38
9.2	5.26	10.4	6.4	10.27

La Tabla 2 se representa en un plano cartesiano, que se muestra en la Fig. 5.12.

Figura 5.12 Muestras de Velocidad de una Transferencia de Archivos en una Conexión Inalámbrica ficticia No 2

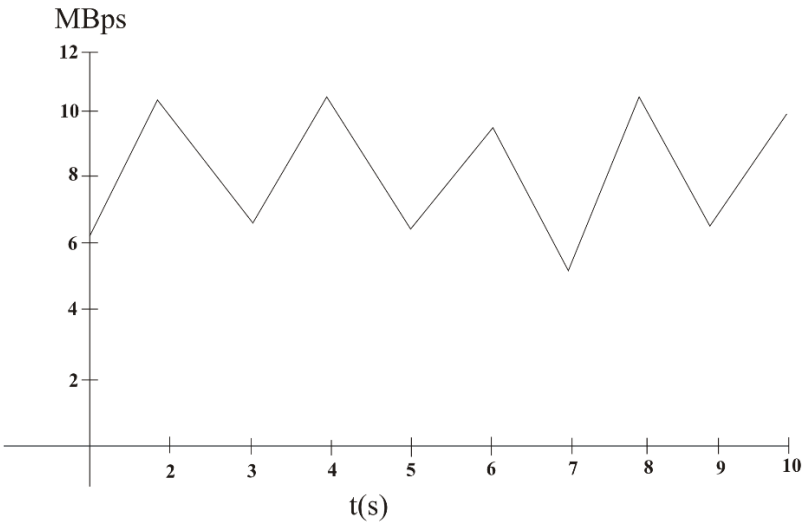


Figura 5.12 Velocidad medida en Bytes, de una transferencia de archivos durante 10 segundos.

Los cálculos de ambos conjuntos de muestras (Tabla 1 y Tabla 2) se muestran a continuación, para el primer grupo quedan:

$$M(1) = \frac{(7.9 + 8.1 + 8.43 + 8.32 + 8.38 + 8.2 + 8.26 + 8.36 + 8.4 + 8.32)MBps}{10}$$

$$M(1) = 8.26MBps$$

La unidad de medida de datos en informática es el Byte, la misma de los conjuntos de muestras de las dos tablas, por lo tanto se convierten a bits, que es la unidad de medida del ancho de banda en los canales de comunicación, por lo tanto la media se expresa en bits;

$$M(1) = (8.26MB\ x\ \frac{8b}{B})ps$$

$$M(1) = 66.08Mbps$$

Los cálculos del segundo grupo quedan:

$$M(2) = \frac{(6.2 + 10.3 + 6.56 + 10.32 + 6.38 + 9.2 + 5.26 + 10.4 + 6.4 + 10.27)MBps}{10}$$

$$M(2) = 8.12MBps$$

$$M(2) = (8.12MB\ x\ \frac{8b}{B})ps$$

$$M(2) = 64.96Mbps$$

Como se observa, los valores de $M(1)$ y $M(2)$ son casi similares, pero la pregunta que surge es, ¿Ambas señales son tan iguales como sus valores de $M(1)$ y $M(2)$? , en los planos cartesianos de las Fig. 5.11 y 5.12 a simple vista se puede saber que ambas señales tienen características diferentes, en tal caso, solo la media aritmética no sería suficiente para describir la señal inalámbrica.

5.06. Varianza. Continuando con el análisis del conjunto de valores de la Tabla 1 y la Tabla 2, se concluyó al final del numeral anterior que la media aritmética no es suficiente para describir globalmente la señal inalámbrica con base en los valores de velocidad resultantes. Otra característica importante de los conjuntos de datos es su dispersión, estas medidas se conocen con el nombre de estadígrafos de dispersión; siendo el más utilizado la *varianza*, la cual da una medida de la dispersión de los valores con respecto a su media aritmética y con ello se puede

saber si la señal presenta una transferencia de datos tendiendo a ser constante o por el contrario tendiendo a ser intermitente.

De acuerdo con (Guerra Bustillo, 2011), la varianza de un conjunto $X_1, X_2, \dots X_n$ de valores de una variable X se define como:

$$V(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - M(x))^2$$

La varianza es la media de las desviaciones al cuadrado de cada valor X_i, con respecto a la media aritmética del conjunto. Por tanto, mientras mayor sea la varianza más dispersos estarán los valores alrededor de la media y mientras más pequeña sea ella menos dispersión habrá, a continuación se calcula la varianza en el conjunto de muestras de la Tabla 1 y la Tabla 2.

$$V(1) = \frac{1}{10}.[(7.9MBps - 8.26MBps)^2 + (8.1MBps - 8.26MBps)^2 + (8.43MBps - 8.26MBps)^2 + (8.32MBps - 8.26MBps)^2 + (8.38MBps - 8.26MBps)^2 + (8.2MBps - 8.26MBps)^2 + (8.26MBps - 8.26MBps)^2 + (8.36MBps - 8.26MBps)^2 + (8.4MBps - 8.26MBps)^2 + (8.32MBps - 8.26MBps)^2]$$

$$V(1) = \frac{1}{10}.[0.2364MBps^2]$$

$$V(1) = 0.023MBps^2$$

$$V(2) = \frac{1}{10}.[(6.2MBps - 8.12MBps)^2 + (10.3MBps - 8.12MBps)^2 + (6.56MBps - 8.12MBps)^2 + (10.32MBps - 8.12MBps)^2 + (6.38MBps - 8.12MBps)^2 + (9.2MBps - 8.12MBps)^2 + (5.26MBps - 8.12MBps)^2 + (10.4MBps - 8.12MBps)^2 + (6.4MBps - 8.12MBps)^2 + (10.27MBps - 8.12MBps)^2]$$

$$V(2) = \frac{1}{10}.[40.81MBps^2]$$

$$V(2) = 4.08MBps^2$$

Con la varianza se pudo reconocer claramente la diferencia entre las señales de las gráficas de la Fig. 5.11 con $V(1) = 0.023MBps^2$ y la Fig. 5.12 con $V(2) = 4.08MBps^2$. La primera grafica muestra un canal de comunicaciones con tendencia estable y la segunda uno con intermitencia.

Figura 5.13 Varianza y Media Aritmética del Conjunto de Muestras de la Tabla 1

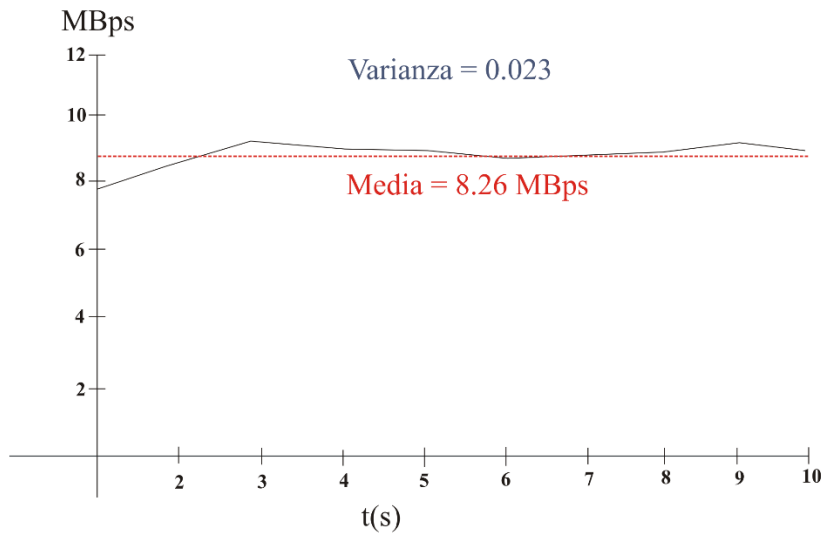


Figura 5.13 Varianza = 0.023, Media = 8.26MBps

Es claro que la velocidad promedio es una de las variables más importante para entender la capacidad de un canal de comunicaciones, pero la varianza permite conocer que tan fiable es esa velocidad promedio, a mayor varianza, mayor dispersión. Algunos servicios como el escritorio remoto o la video-vigilancia remota en vivo requieren implementarse en canales con

baja latencia y además que dicha latencia debe ser lo más constante posible, independiente de la velocidad promedio.

Figura 5.14 Varianza y Media Aritmética del Conjunto de Muestras de la Tabla 2

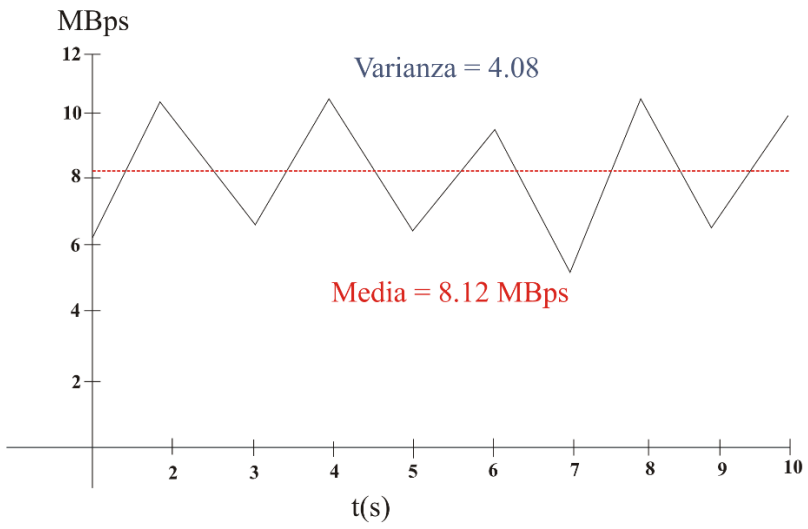


Figura 5.12 Varianza = 4.08, Media = 8.12 MBps

Para el servicio de escritorio remoto, las imágenes de la ubicación del puntero del mouse deben transferirse con retardos cortos para que el usuario conozca los detalles en tiempo real del manejo del equipo remoto. Aun si el promedio de velocidad fuese suficiente como para implementar el servicio de escritorio remoto, no serviría de nada si la latencia es muy variable y con retardos de transmisión muy prolongados porque la velocidad del canal variaría notablemente cada segundo (por lo tanto la varianza se incrementara) y eso es señal de que dicho canal no es lo suficientemente estable para hacer escritorio remoto y como consecuencia, el movimiento del puntero del mouse va a ser entrecortado y errático.

Con la video-vigilancia remota sucedería lo mismo, las imágenes de las cámaras de vigilancia deben grabar cada detalle en vivo durante un lapso prolongado de tiempo y aun si el

promedio de velocidad fuese lo suficientemente bueno para dicho servicio tampoco serviría si el canal presenta variaciones notables de velocidad cada segundo por causa de una latencia larga y muy variable porque sucederá que en determinados momentos las imágenes se verían mal por no actualizarse en los momentos justos y a la velocidad correcta y por lo tanto el DVR las grabaría de esa forma, o en su defecto se caería el servicio. En el evento indeseado de un robo, las imágenes al detalle serian la prueba para solucionar el caso o para impedir dicho robo pero si las imágenes no se reproducen correctamente y en el momento justo se puede correr el riesgo de sufrir un problema sin poder actuar oportunamente.

5.07. Desviación estándar y Coeficiente de variación. Repasando lo visto hasta el momento, en los numerales anteriores se explicó la media aritmética para conocer el valor de tendencia central de un conjunto de muestras, en este caso aplicado a las telecomunicaciones con dos conjuntos de muestras de velocidad de dos conexiones inalámbricas (ver Tabla 1 y Tabla 2), con la varianza se conoció la magnitud de dispersión de dichas muestras, que permitió saber si cada uno de los canales de comunicaciones tendían a ser estables o intermitentes. Ahora con la desviación estándar y el cociente de variación se puede saber en qué grado la varianza afecta la calidad de la señal de las conexiones inalámbricas o el conjunto de muestras a nivel general.

De acuerdo con (Guerra Bustillo, 2011), la desviación típica de un conjunto de valores $X_1, X_2, \dots X_n$ se define como la raíz cuadrada positiva de la varianza.

$$D(x) = \sqrt{V(x)}$$

Con el valor de $D(x)$ podemos conocer el rango de valores superior e inferior respecto a la media en el que pertenecen la gran mayoría de las muestras. A continuación se calcula la

desviación estándar en los conjuntos de muestras de la Tabla 1 y la Tabla 2, para el primer grupo queda:

$$D(1) = \sqrt{0.023 MBps^2}$$

$$D(1) = 0.1516 MBps$$

Para el segundo grupo queda:

$$D(2) = \sqrt{4.08 MBps^2}$$

$$D(2) = 2.01 MBps$$

Teniendo el valor de las desviaciones estándar $D(1)$ y $D(2)$ se calculan los coeficientes de variación. El coeficiente de variación de un conjunto de valores $X_1, X_2, \dots X_n$ se define como el cociente de la desviación estándar y la media de dicho conjunto.

$$CV(x) = \frac{D(x)}{M(x)}$$

Con bastante frecuencia este coeficiente de variación se expresa en tanto por ciento; para ello basta con multiplicar el resultado por 100. Se hallan los coeficientes en ambos grupos de muestras.

$$CV(1) = \frac{D(1)}{M(1)}$$

$$CV(1) = \frac{0.1516 MBps}{8.26 MBps}$$

$$CV(1) = 0.018$$

$$CV(2) = \frac{D(2)}{M(2)}$$

$$CV(2) = \frac{2.01 MBps}{8.12 MBps}$$

$$CV(2) = 0.247$$

Con el coeficiente de variación $CV(x)$ podemos ver la relación entre el promedio y la desviación estándar y con ello sabremos el porcentaje efectivo de variación de la media aritmética. Al asumir el cien por ciento como el valor total del promedio, se calcula el porcentaje de *Estabilidad* de la media aritmética invirtiendo el cálculo del coeficiente de variación y expresándolo de manera porcentual, a saber:

$$\%Est(x) = 100 - CV(x).100$$

Se calcula con $CV(1)$ y $CV(2)$:

$$\%Est(1) = 100 - CV(1).100$$

$$\%Est(1) = 98.2\%$$

Para el primer conjunto de muestras, $\%Est(1)$ indica que el 98.2% del promedio es efectivo, es decir que el conjunto de muestras se acercan bastante al promedio y el conjunto de valores tiende a ser casi el mismo, por lo cual es una conexión estable, el cálculo continúa para el segundo conjunto:

$$\%Est(2) = 100 - CV(2).100$$

$$\%Est(2) = 75.3\%$$

En el segundo conjunto de muestras la efectividad del promedio es del 75.3%, en este caso estaría en el límite de lo aceptable para un canal de comunicaciones inalámbrico que soporte servicios que necesiten hasta el 60% del ancho de banda del promedio.

Figura 5.15 Coeficiente de Variación, Desviación Estándar y Estabilidad del Conjunto de Muestras de la Tabla 1

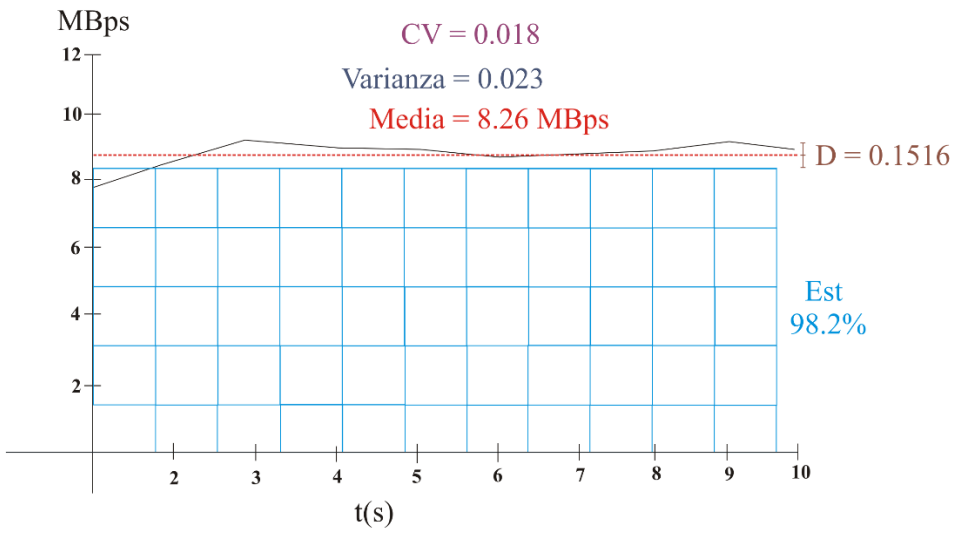


Figura 5.15 Varianza = 0.023, Media = 8.26 MBps, D = 0.1516 MBps, CV = 0.018 Est = 98.2%

Figura 5.16 Coeficiente de Variación, Desviación Estándar y Estabilidad del Conjunto de Muestras de la Tabla 2

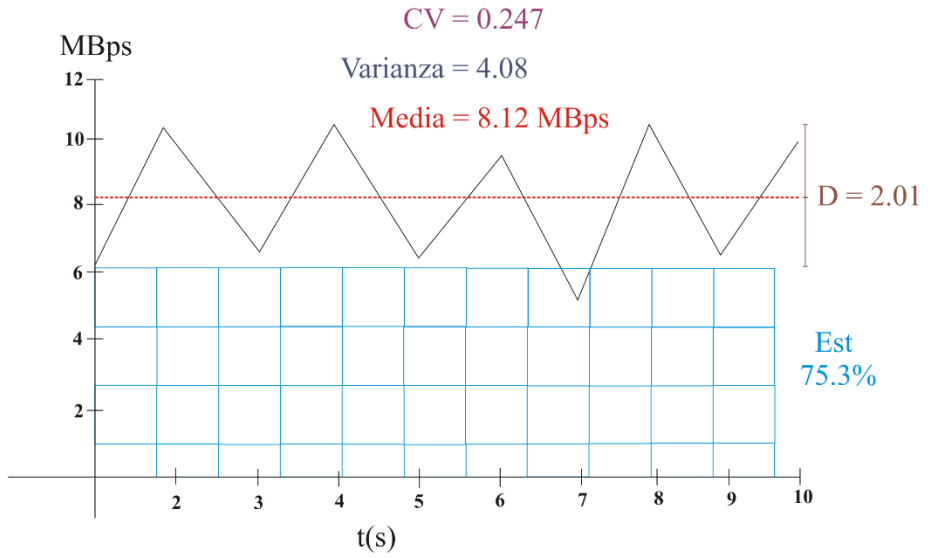


Figura 5.16 Varianza = 4.08, Media = 8.12 MBps, D = 2.01 MBps, CV = 0.247 Est = 75.3%

La media aritmética del segundo conjunto de muestras resulto en 8.12 MBps, por lo tanto, ese valor tiene un 75.3% de efectividad así que alrededor de 6.11 MBps de ese promedio son efectivos y se puede concluir que el canal correspondiente a las muestras de la Tabla 2 soporta servicios efectivamente pero que consuman menos de 6 MBps.

Aplicaciones para medir ancho de banda

En la informática y las telecomunicaciones existen muchas aplicaciones para medir ancho de banda, la diferencia de cada una está en cómo funcionan y también por otras características como: la plataforma, los protocolos, la configuración, los detalles de la medición, etc. En este numeral se exponen algunas aplicaciones que fueron consultadas, vale aclarar que el objetivo de buscar estas aplicaciones no es reemplazar el proyecto, sino brindar mayor conocimiento acerca de este tipo de herramientas para que se oriente bien el desarrollo del software con el fin de que cumpla con las necesidades de la empresa de energía.

Para los programas que se describirán en breve no se registran pruebas de funcionamiento debido a que simplemente se contemplan como opciones existentes en el mercado y fueron probadas solo con fines experimentales que no entran en materia dentro de este documento. Estas aplicaciones se plantearon como solución en la empresa de energía pero no fueron aprobadas con el argumento de que no permitían transmitir datos verificables para el usuario (como los archivos), sino que ellas mismas generan su propio tráfico y esto no gusto a los operarios de la red, sin embargo sirvieron de guía para este proyecto.

5.08. Iperf. Según la página oficial de esta aplicación, fue desarrollada por un grupo denominado NLANR/DAST y la idea de esta era servir como una herramienta moderna para medir al máximo el desempeño del ancho de banda de las conexiones TCP y UDP. Sin embargo la última versión oficial de este programa salió en el 2008, pero aun así iperf se ha mantenido como una de las herramientas más populares hoy en día de su clase.

Figura 5.17 Consola de Iperf

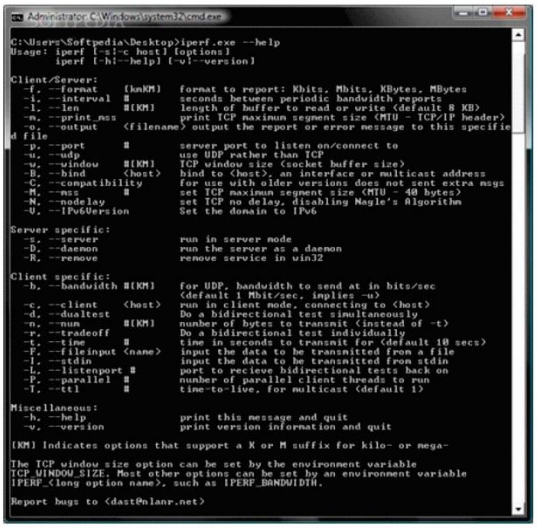


Figura 5.17 Iperf se maneja con comandos y tiene una gran variedad de funciones con los protocolos TCP y UDP.

Como se observa en la Fig. 5.17, iperf se ejecuta en consola y todo el programa se maneja por comandos. Las características de este son:

- Plataformas: Windows XP, vista y 7, Linux (necesita compilación previa)
- Tipo de aplicación: Consola
- Unidades de medida: kb, Mb, kB, MB
- Protocolos: TCP, UDP
- IPv6: Si
- Tipo de licencia: Libre

5.09. Packetcheck. Esta aplicación fue desarrollada por la empresa estadounidense GL COMMUNICATIONS, es una aplicación sencilla que permite conocer el comportamiento de las tramas de datos desde la capa 2 hasta la capa 4, es de uso comercial y su adquisición necesariamente debe ser con previa compra.

Figura 5.18 Ventana de controles de Packetcheck

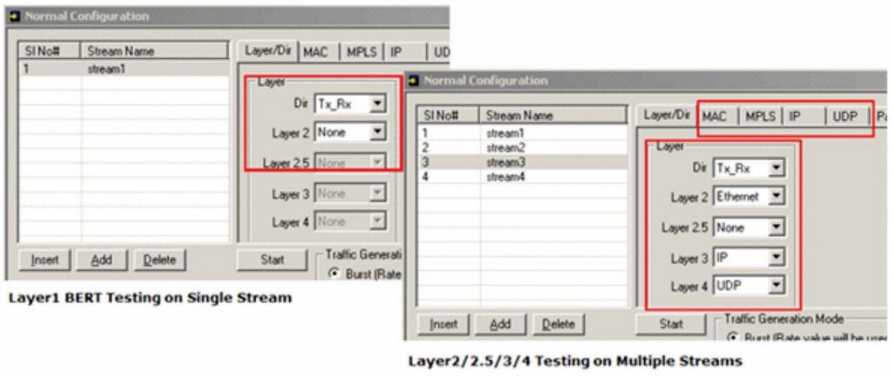


Figura 5.18 Paquetcheck se maneja con controles muy sencillos, su configuración permite monitorear las tramas entre la capa 2 y la capa 4.

Packetcheck permite hacer configuraciones en las capas 2, 3 y 4, y exporta los reportes en PDF o XML. Las características de este son:

- Plataformas: Windows XP, vista y 7
- Tipo de aplicación: Grafica
- Unidades de medida: kb, Mb
- Protocolos: UDP, en capa 2 tiene MPLS
- IPv6: Si
- Tipo de licencia: Comercial

5.10. Pingb. Una aplicación diferente del resto de su tipo porque utiliza el protocolo ICMP para medir el ancho de banda, gracias a los ecos ICMP. Se desconoce su creador y también su fecha de publicación.

Figura 5.19 Consola de Pingb

Figura 5.19 Pingb se maneja con comandos pero su única función es medir el ancho de banda basado en la técnica de ecos de ICMP.

La consola es el medio en que se maneja, era apenas lógico que usando ICMP debía ser así, permite conocer las respuestas tanto en bajada como en subida y calcular el ratio entre ambas. Las características de este son:

- Plataformas: Windows XP, vista y 7, Linux (necesita compilación previa)
- Tipo de aplicación: Consola
- Unidades de medida: kb
- Protocolos: ICMP
- IPv6: No
- Tipo de licencia: Libre

5.11. Netstress. Una herramienta relativamente nueva en el mercado que hace reportes gráficos del rendimiento del ancho de banda en interfaces cableadas e inalámbricas, tiene un aspecto cómodo y agradable para el usuario, fue desarrollada por la empresa NUTS ABOUT NETS, según la página oficial de la compañía, está antiguamente era una herramienta privada, es decir que solo era de servicio interno y luego la compañía decidió lanzarla al mercado.

Figura 5.20 Panel Gráfico de Netstress

Figura 5.20 Netstrees se maneja con controles gráficos intuitivos, en el panel aparecen dos gráficas, una superior muestra las medidas en el cliente y la inferior muestra las medidas en el servidor.

El programa tiene la gran ventaja que guarda en un historial las medidas de conexiones particulares y permite evaluar el rendimiento de dichas conexiones en el mediano o largo plazo para saber cuánto se han degradado, o por el contrario, como han ido mejorando, y con esta información hace un análisis cualitativo y cuantitativo del funcionamiento de la conexión y ofrece sugerencias al respecto para mejorar dichos inconvenientes. Si el análisis se hace sobre conexiones inalámbricas, esta herramienta tiene la capacidad de mostrar la relación señal a ruido y la potencia de recepción, los cuales se comparan para saber de qué manera influyen en el rendimiento del ancho de banda. Las características de este son:

- Plataformas: Windows XP, vista y 7
- Tipo de aplicación: Grafica
- Unidades de medida: kb, Mb, kB, MB
- Protocolos: TCP, UDP
- IPv6: No
- Tipo de licencia: Comercial

5.12. LANBench. Es una herramienta sencilla que diagnostica el rendimiento de las conexiones TCP. Fue creada por la compañía ZACH SAW, el consumo de CPU de esta aplicación es muy bajo, lo que le da una gran capacidad para llevar al máximo rendimiento las conexiones TCP.

Figura 5.21 Ventana de LANbench

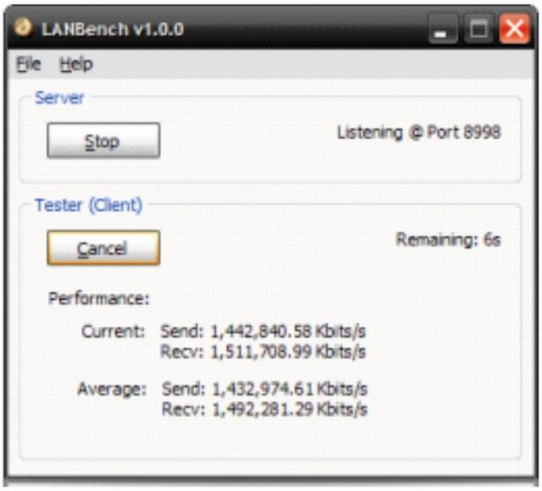


Figura 5.21 LANbench solo funciona con un par de botones, su sencillez lo hace muy práctico y su bajo consumo lo hace muy util en determinadas condiciones de operación.

El uso es fácil, solo basta con oprimir un botón para iniciar el servidor, además permite hacer una sobrecarga de tráfico con hasta 20 conexiones simultáneas. Las características de este son:

- Plataformas: Windows XP, vista y 7
- Tipo de aplicación: Grafica
- Unidades de medida: kb
- Protocolos: TCP
- IPv6: No
- Tipo de licencia: Libre

5.13. Comparación de las aplicaciones. A continuación se muestra en la Tabla 3 el cuadro comparativo que resume las características de los programas explicados anteriormente.

Tabla 3
Cuadro comparativo de las aplicaciones

	Plataformas			**Unidades**					**Protocolos**				
Aplicaciones	**WXP**	**WV**	**W7**	**L**	**Kb**	**Mb**	**kB**	**MB**	**TCP**	**UDP**	**ICMP**	**IPv6**	**Licencia**
Iperf	x	x	x	x	x	x	x	x	x	x		x	GPL
Packetcheck	x	x	x		x	x				x		x	Comercial
Pingb	x	x	x	x	x						x		GPL
Netstress	x	x	x		x	x	x	x	x	x			Comercial
LANBench	x	x	x		x				x				GPL

5.14. Finalidad del proyecto. Medir el ancho de banda con el protocolo TCP así mismo como lo hacen los programas descritos en la Tabla 3, pero con la diferencia en los siguientes aspectos:

- El programa de este proyecto debe venir compilado para que la instalación en Linux sea fácil y rápida. En la empresa de energía los operarios no disponen de suficiente tiempo para hacer procesos tediosos de compilación, ellos necesitan una aplicación que puedan usar inmediatamente, esta característica no la tienen las aplicaciones previamente descritas porque no todas son compatibles con Linux, y las que sí lo son, no vienen compiladas.
- Los operarios de la empresa de energía quieren verificar la integridad de los datos, como se mencionó al inicio de este numeral, se deben emplear archivos para que puedan comprobarse del otro lado de la comunicación, esta característica no la tienen las aplicaciones anteriores porque ninguna de ellas puede emplear archivos que sirvan de información para las pruebas.

- Los operarios de la empresa de energía necesitan que el programa pueda permanecer activo las veinticuatro horas para descargar o subir archivos entre cliente y servidor, esta función no está en ninguna de las aplicaciones anteriores.

Se reitera la aclaración que los programas consultados y descritos en el numeral anterior cumplen la función de medir ancho de banda pero dentro de la empresa de energía se consideró desarrollar e implementar una aplicación que se acomodara a los gustos y a la manera de trabajar de los operarios como se argumentó en este numeral.

Capítulo 6

Método

La conducción del trabajo se dividió en tres etapas basadas en el objetivo del proyecto, cada una de las etapas se desarrollan estrictamente de manera secuencial. Esta metodología no se basa en ninguna existente en particular pero si aplica los cuatro procesos fundamentales de cualquier proyecto, los cuales son: planificación, ejecución, control y conclusión. El proceso de control en este caso no aplica porque este proyecto no dura indefinidamente. Las dos primeras etapas de este proyecto son: “conocer el objetivo” y “aprender acerca del objetivo”, son la fase de planificación, mientras la tercera etapa es la “construcción hacia el objetivo” corresponde a la fase de ejecución y conclusión. Es importante aclarar que las especificaciones, el análisis y el diseño de las etapas y de la metodología se explicaron con detalles en el anteproyecto y no se tratara en este capítulo porque a este le corresponde solamente explicar el desarrollo del mismo con la implementación de la metodología previamente definida en el anteproyecto.

Etapas

El desarrollo del proyecto se orientó de acuerdo a las siguientes etapas.

Conocer el objetivo

Inicialmente se hace la descripción del problema, la solución y los alcances.

Aprender acerca del objetivo

Consiste en el aprendizaje de todos los temas correspondientes al problema y en la definición de todos los elementos necesarios para desarrollar su solución, tales como: herramientas, fuentes de información, y los medios de desarrollo.

Construcción hacia el objetivo

Corresponde al diseño y la elaboración del modelo que resuelva el problema, para ello se sigue con los siguientes pasos:

- Proponer mínimo dos modelos de solución teniendo en cuenta los criterios de la primera etapa (problema, solución y alcances).
- Evaluar cada modelo de acuerdo a los alcances y elegir el más conveniente.
- Dividir el modelo en partes ordenadas e independientes.
- Construir y probar cada una de las partes del paso anterior en orden lógico, cada parte debe tener variables de entrada y variables de salida que permitan enlazar unas con otras. A medida que las partes se vayan haciendo correctamente, se unirán como una cadena hasta que se complete el modelo.

Puesta en marcha del proyecto

Etapa primera

6.01.1. Descripción del problema. La sustentación de la problemática se encuentra expuesta en el capítulo previo de ‘Planteamiento del problema’.

6.01.2. Solución. Una manera fácil para medir el ancho de banda en múltiples canales de comunicación, consiste en la instalación de un servidor en un punto estratégico de la red. El servidor aloja información que cualquier cliente puede descargar y gracias a esto se puede conocer la velocidad de los canales por donde pasa la descarga. Esta solución es similar a los servidores que hacen pruebas de velocidad a través de Internet, pero con la diferencia que en esta solución el servidor no es externo sino local y por lo tanto las conexiones que evaluará también

serán locales dentro de la misma red sin importar que estén ubicadas en áreas geográficas extendidas.

Hacer un servidor de estas características implica programar una herramienta informática que mida el ancho de banda mediante la transferencia de archivos a través del protocolo TCP/IP. El servidor tendrá una nube de información donde uno o varios clientes puedan descargar archivos de su carpeta raíz. Para el uso de la aplicación, esta se instala primero en el equipo de referencia ejecutándose en modo servidor. Cualquier equipo de la red que quiera conectarse al servidor (equipo de referencia) debe tener instalada la misma aplicación pero ejecutada en modo cliente.

Luego de que ambos equipos estén conectados, es recomendable que la descarga sea con un archivo relativamente pequeño para que el tráfico dure poco tiempo pero también lo suficientemente grande para que permita obtener la máxima velocidad de transmisión posible. Otra consideración importante es que los canales involucrados en la prueba sean restringidos temporalmente para el uso de terceras partes mientras se hace la descarga, con el fin de que la medida no se vea afectada por la intrusión de tráfico ajeno a la prueba.

6.01.3. Alcances. El producto final debe cumplir con los siguientes requisitos:

- Medir la velocidad de bajada de una conexión TCP/IP (transferencia de servidor a cliente).
- Medir la velocidad de subida de una conexión TCP/IP (transferencia de cliente a servidor).
- Ser compatible para Windows 7, Windows server 2008/2011 y Linux, los cuales son los sistemas operativos usados por los operarios de la red de datos de la empresa de energía. Los requisitos mínimos de uso son: 1GB de memoria RAM, familias de procesadores con

tecnología igual o superior a los Pentium 4 de 2,4 GHz con tecnología HT, es importante que los procesadores tengan mínimo dos núcleos puesto a que el software del proyecto emplea dos hilos de procesamiento, uno de comunicación y otro de manejo de la información, los cuales se explicaran más adelante dentro de este capítulo.

- Funcionar en multicast (el servidor debe responder peticiones de varios clientes simultáneamente).

Etapa Segunda

6.02.1. Herramientas de trabajo

- Ordenador portátil hp dv2750.
- Ordenador portátil hp dv6700.

6.02.2. Medios

- Plataforma: Windows 7.
- Lenguaje: Java.
- Compilador: Java Develpment Kit.

6.02.3. Temas de estudio

- El modelo OSI.
- El protocolo TCP.
- Lenguaje de programación java.
- Ancho de banda y técnicas de medición de transferencia de datos.

Etapa tercera

6.03.1. Modelos de solución. Los modelos de solución se plantearon teniendo en cuenta los medios escogidos y el nivel de conocimientos para su desarrollo, cada uno tiene diferencias importantes en las funciones y el aspecto respecto a los demás. Los modelos planteados se presentan en la siguiente lista.

- Programa grafico avanzado.
- Programa grafico simple.
- Programa en consola.

6.03.2. Descripción de los modelos de solución. Antes de continuar a la presentación de los modelos, cabe aclarar que éstos están descritos solo en las características críticas de elección debido a que solo son simples bocetos y no son diseños complejos con especificaciones.

6.03.2.1. Modelo 1. El programa grafico avanzado tiene la función de presentar la velocidad en un plano cartesiano y hacer un análisis detallado de las muestras como; regresión, integración de la función de la gráfica y jitter.

Figura 6.1 Modelo 1. Programa Grafico avanzado con Visualización en Plano Cartesiano

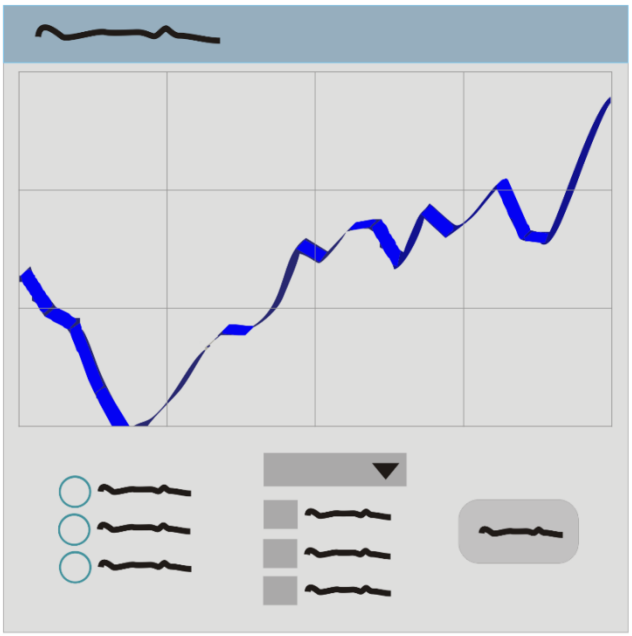


Figura 6.1 Representación grafica de la velocidad de transmisión de datos en el plano cartesiano de la parte superior. La parte inferior tiene los controles del formulario que permiten configurar las funciones del programa.

El grado de dificultad de este modelo es alto porque se deben programar dos partes: la parte lógica y las API de la parte gráfica, las cuales pueden requerir más código fuente que la parte lógica. Las herramientas reconocidas en el mercado para graficar variables como Matlab o Excel, no se aprueban dentro de la empresa de energía porque para el primer caso no se tiene la licencia, mientras para el segundo caso, Excel no se puede combinar directamente con java, pero si puede con Visual Basic y la empresa de energía no tiene la licencia de Visual Studio. A pesar de las dificultades previamente dicha, la terminación de éste modelo dejaría a disposición una herramienta estética y muy intuitiva aunque el consumo de CPU es alto.

6.03.2.2. Modelo 2. El programa grafico simple a diferencia del primero no representa la velocidad en un plano cartesiano sino en una tabla de registros y hace un análisis solo de la velocidad de transmisión.

Figura 6.2 Modelo 2. Programa Grafico Simple

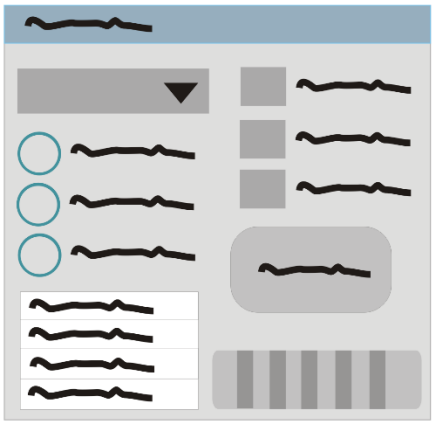


Figura 6.2 Los ajustes de la conexión se hacen con los controles del formulario. La tabla de registros aparece en la parte inferior izquierda y almacena las muestras de velocidad. En la parte inferior derecha hay una barra que muestra el progreso de la prueba.

El grado de dificultad de este modelo es intermedio y la terminación de éste dejaría a disposición una herramienta sencilla pero fácil de manejar aunque regularmente intuitiva. Está claro que el modelo 1 analiza con mayores detalles la comunicación que el modelo 2 y tiene un aspecto más profesional, pero el modelo 2 también es muy funcional porque como se mencionó antes, el manejo es más fácil, el consumo de CPU es más bajo y es más compatible que el modelo 1 porque usa los elementos gráficos básicos de java que ya vienen construidos, en cambio el modelo 1 necesita elementos gráficos más avanzados que no se encuentran construidos sino se deben diseñar y programar desde cero, lo que consumiría abismalmente el tiempo disponible del proyecto.

6.03.2.3. Modelo 3. El programa en consola tiene la función de visualizar la velocidad textualmente en la ventana de comandos. Éste modelo solo se maneja con el teclado e igual que el modelo 2 solamente analiza la velocidad de transmisión.

Figura 6.3 Modelo 3. Programa en Consola

Figura 6.3 Ventana de la consola que se maneja solamente con el teclado .

El grado de dificultad de este modelo es bajo y la terminación de éste dejaría a disposición una herramienta con el más bajo consumo de CPU comparada con las dos anteriores, bastante flexible pero poco estética.

6.03.3. Elección del modelo. La decisión radica en las ventajas y desventajas de los modelos respecto a los alcances. El modelo 1 es el programa más completo de los tres, cumple con todos los requisitos mínimos de los alcances, además hace otros análisis adicionales al ancho de banda con una presentación formidable pero entre las desventajas se encuentra el tiempo, debido a que por ser el mejor, exige programar las funciones gráficas y lógicas, otro inconveniente es la compatibilidad, por causa de que las aplicaciones graficas en java pueden variar significativamente su rendimiento con el cambio de sistema operativo o de versión de máquina virtual porque algunas bibliotecas graficas de java se adaptan mejor con unos sistemas operativos

que con otros, por lo tanto, para reducir este tipo de problemas hace que tome mucho más tiempo el perfeccionamiento de este modelo que los demás (Modelo 2 y Modelo 3).

El modelo 2 solo cumple con los requisitos mínimos de los alcances, utiliza controles gráficos básicos que no le dan un aspecto tan agradable como el modelo 1 pero consume menor cantidad de tiempo en el desarrollo aunque tiene el mismo inconveniente de compatibilidad de las bibliotecas graficas que el modelo 1 pero en un grado menor porque los controles gráficos básicos de este modelo tienen una mejor adaptación y rendimiento que los controladores gráficos avanzados del modelo 1.

El modelo 3 al igual que el modelo 2 solo cumple con los requisitos mínimos de los alcances y no es grafico sino que se carga en consola, el aspecto claramente es el menos intuitivo de los tres aunque el tiempo de desarrollo es mucho menor al de todos y también es altamente compatible con cualquier plataforma porque al ser de consola se ejecuta fácilmente porque no sería necesario ninguna librería grafica en particular y la consola de comandos es una herramienta que tienen todos los sistemas operativos (Linux, Windows y Mac OS).

Con base en los alcances, el modelo 3 fue el elegido porque a pesar de que solo cumpla con los requisitos mínimos y que los programas en consola sean poco atractivos, tiene bajo consumo de CPU y puede ejecutarse fácilmente en varias plataformas que tengan la máquina virtual de java sin presentar mayores inconvenientes, y así los usuarios que implementen el programa no dependerían de ninguna licencia en particular, cosa que puede ser perjudicial con el modelo 1 y el modelo 2. El otro aspecto fundamental es que el modelo 3 a pesar de ser el menos consumidor de tiempo, está ajustado al tiempo disponible que tuvo el proyecto para la fecha, lo que da por descartadas las opciones de los modelos 1 y 2.

6.03.4. Partes del modelo. El modelo se divide en tres partes. La Parte 1 es la función principal que contiene el menú, la presentación y el cierre del programa, es la parte que controla el acceso de todas las funciones. La parte 2 es la función que opera de cliente y la parte 3 es la función que opera de servidor. En el diagrama de la Fig. 6.4 se presenta el diagrama de las partes en la estructura del programa.

Figura 6.4 Diagrama de las Partes del Programa

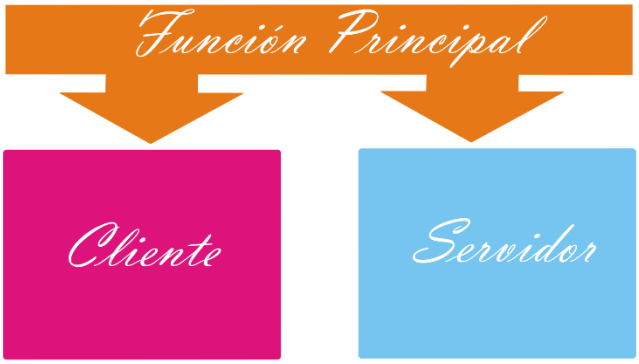


Figura 6.4 La ejecución inicia con la parte de la función principal (cuadro de colo naranja), que es la que organiza el acceso a las funciones del Cliente(cuadro de color rosa) y del Servidor (cuadro de color azul).

El estilo de programación que se implementó en el proyecto fue la programación orientada a objetos, por lo tanto, el programa se subdivide en funciones que se encuentran dentro de *paquetes* y *clases*. Los paquetes se identifican con las carpetas que agrupan los archivos del programa y las clases se identifican con los archivos de extensión *class* que se encuentran dentro de los paquetes.

6.03.4.1. Parte 1. El desarrollo de cualquier programa profesional debe tener una estructura lógica que sirva de columna vertebral y organice todas las funciones. En este proyecto la estructura que ordena todo es la Parte 1 y en ella se encuentra la función principal que es la que inicializa los

elementos principales del programa y organiza jerárquicamente todas las funciones. Los archivos que corresponden a la parte 1 se muestran en la Fig. 6.5.

Figura 6.5 Archivos de la Parte 1

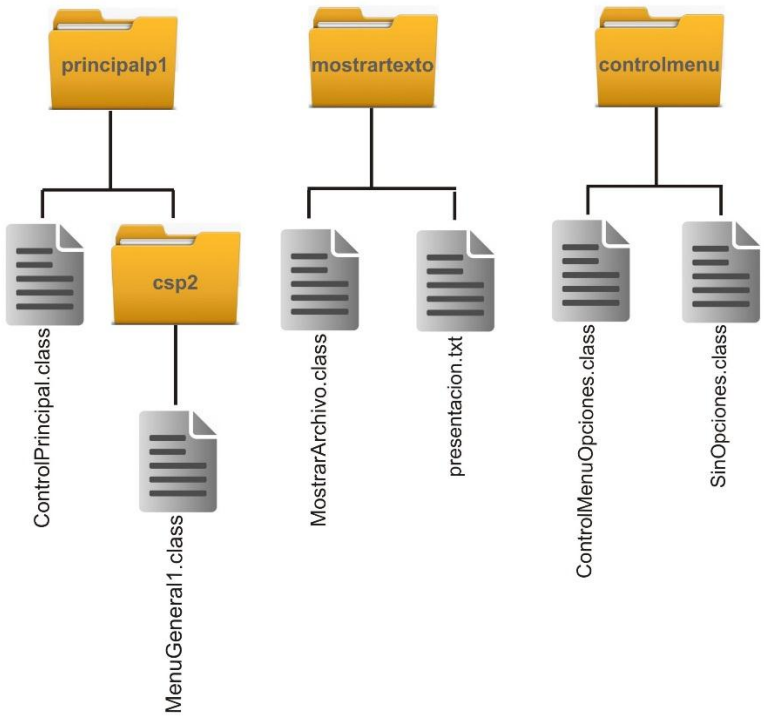


Figura 6.5 Los iconos de la parte superior simbolizan los archivos de la carpeta raíz, debajo de cada carpeta existen ramales que indican el contenido de cada una.

6.03.4.1.1 Paquetes y Clases de la Parte 1. Tal como se observa en la Fig. 6.5, hay tres paquetes que son: *principalp1*, *mostrartexto* y *controlmenu*, para continuar es importante aclarar primero que en este documento se explica la lógica de funcionamiento de los elementos (paquetes y clases) de las tres partes del programa, pero no se explica el código fuente, porque saber dicho código requiere que los lectores tengan conocimientos avanzados de programación que reducirían el entendimiento de muchos lectores interesados del área de ingeniería electrónica y también dichos temas de programación desenfocarían el tema principal del proyecto con relación al ancho

de banda. Para dar a entender lo que es una clase, es simple, es una parte del programa que cumple una o varias funciones particulares.

El paquete *controlmenu* contiene las siguientes clases:

- *ControlMenuOpciones.class*: clase que contiene las funciones que crean el menú de opciones del programa.
- *SinOpciones.class*: clase que contiene las funciones que manejan los posibles errores de la clase *ControlMenuOpciones.class*.

El paquete *mostrartexto* contiene las siguientes clases:

- *MostrarArchivo.class*: clase que contiene las funciones que visualizan en pantalla los caracteres de cualquier archivo de texto, en este caso, el archivo *presentación.txt* que se encuentra dentro del mismo paquete y contiene un mensaje introductorio de presentación del programa.

El paquete *principalp1* contiene las siguientes clases:

- *ControlPrincipal.class*: clase que contiene la función principal que se ejecuta primero en todo el programa y en ella se declaran todas las variables iniciales y se utilizan todas las clases de la parte 1.

El paquete *principalp1* contiene el sub-paquete *csp2* que contiene la siguiente clase:

- *MenuGeneral1.class*: clase que contiene las funciones que evalúan los datos de entrada que el usuario ingresa en el menú principal y con base en el resultado de dichas decisiones dirige el flujo del programa hacia donde haya escogido el usuario.

6.03.4.1.2 Funcionamiento de la Parte 1. La clase *ControlPrincipal.class* que se encuentra dentro del paquete *principalp1*, tiene la función principal del programa que es la primera en ejecutarse de todas. Dicha clase se encarga de armar todo el contenido básico y de organizar todas las

funciones del programa, por lo tanto, ésta invoca a las clases *ControlMenuOpciones.class* y *MostrarArchivo.class*, de ellas utiliza la primera para las funciones que crean el menú principal y de la segunda utiliza las funciones que visualizan el mensaje de presentación que se encuentra en el archivo de texto *presentación.txt* que está dentro del paquete *mostrartexto*. En el diagrama de la Fig. 6.6 se presenta el funcionamiento de la parte 1. Cuando el usuario escoge alguna de las opciones del menú principal, las funciones de la clase *ControlMenuOpciones.class* verifican que la entrada del teclado haya sido correcta y que corresponda a alguna de las opciones del menú, pero si es incorrecta le indicara al usuario el error. De ser correcta la entrada de datos, las funciones de la clase *ControlMenuOpciones.class* le informaran a las funciones de la clase *MenuGeneral1.class* la opción escogida y ellas direccionarán el flujo del programa hacia la parte de la opción que corresponda.

Figura 6.6 Diagrama de las Partes del Programa con las clases

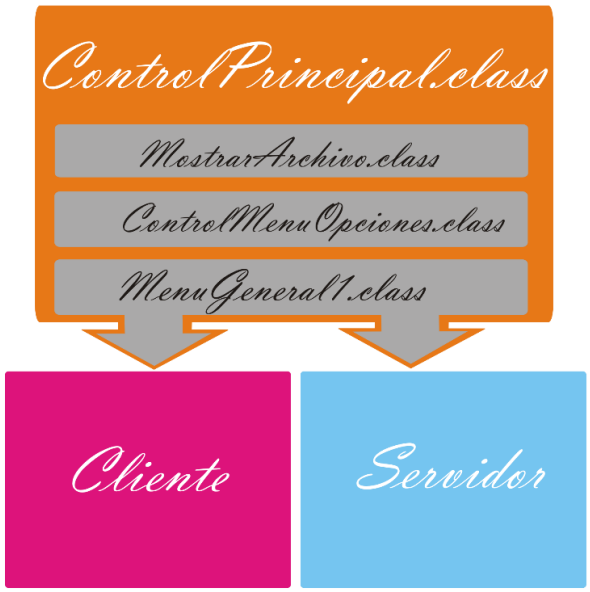


Figura 6.6 El mismo digrama de la Fig. 6.4 pero con los detalles de las clases que corresponden a la parte 1, dicha parte es contenida por la clase *ControlPrincipal.class* que internamente invoca a las clases *MostrarArchivo.class*, *ControlMenuOpciones.class* y *MenuGeneral1.class* que realizan los subprocesos introductorios y de control del programa, cada una de estas clases operan secuencialmente. Como se observa, la clase *MenuGeneral1.class* es la que dirige el flujo del programa hacia la función de Cliente (parte 2) o la función de Servidor (parte 3).

6.03.4.2 Parte 2 y Parte 3. La segunda y la tercera parte se explican en un solo numeral, la razón es porque ambas tuvieron que programarse simultáneamente durante todo el desarrollo del proyecto con el fin de sincronizar correctamente la transferencia de datos entre cliente y servidor, por lo tanto, el funcionamiento de estas partes será más entendible para el lector si estas se explica intercaladamente en el mismo numeral en vez de hacerlo independientemente en numerales separados.

Figura 6.7 Todos los Archivos Finales del Proyecto de las Tres Partes

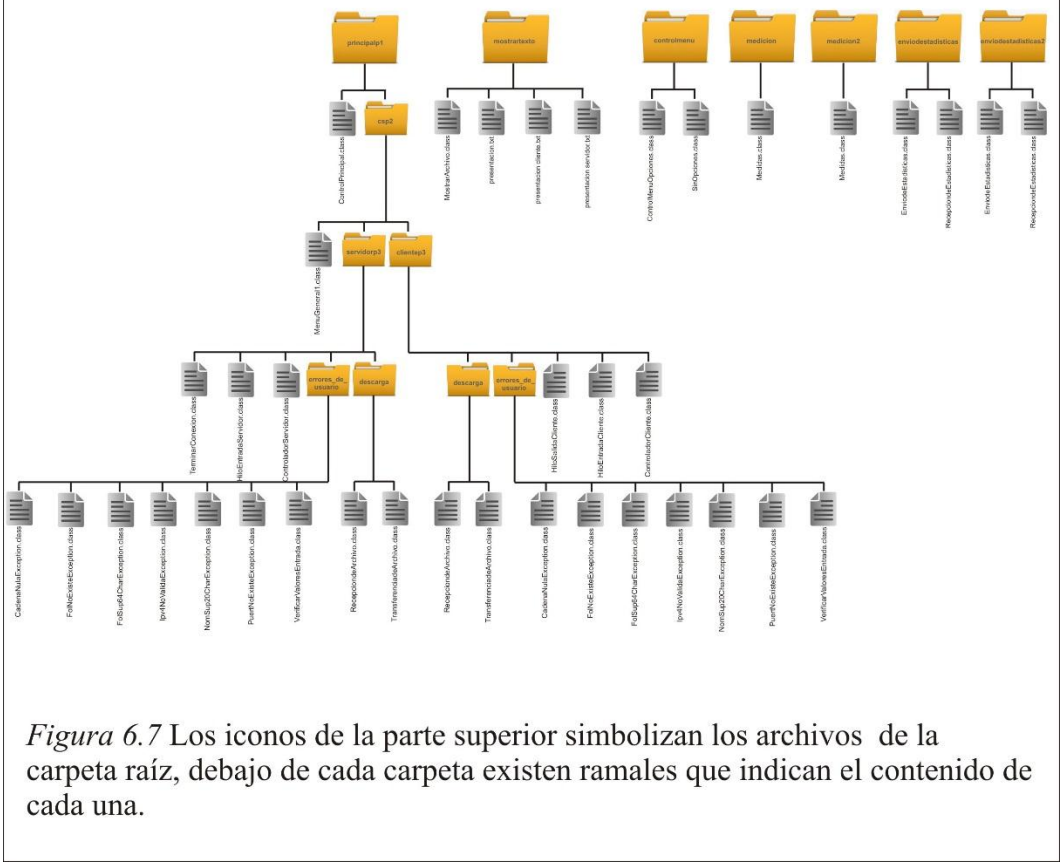


Figura 6.7 Los iconos de la parte superior simbolizan los archivos de la carpeta raíz, debajo de cada carpeta existen ramales que indican el contenido de cada una.

Como se observa en Fig. 6.7, la cantidad de paquetes y clases es amplia porque corresponde a todos los archivos resultantes del proyecto, a diferencia de la Fig. 6.5 que solo muestra los archivos hasta la parte 1 y en ella había tan solo tres paquetes, en cambio la Fig. 6.7

muestra un total siete paquetes en la carpeta raíz. En este numeral se explican los paquetes y las clases que solamente corresponden a la parte 2 (Cliente) y a la parte 3 (Servidor) porque los paquetes correspondientes a la parte 1 ya se trataron en el numeral anterior.

6.03.4.2.1 Paquetes y Clases de la Parte 2 y de la Parte 3. Primero se explican los paquetes y clases de la carpeta raíz porque son los principales.

Figura 6.7a Acercamiento de Vista de la Fig. 6.7 hacia los Paquetes y Clases de la Carpeta Raíz

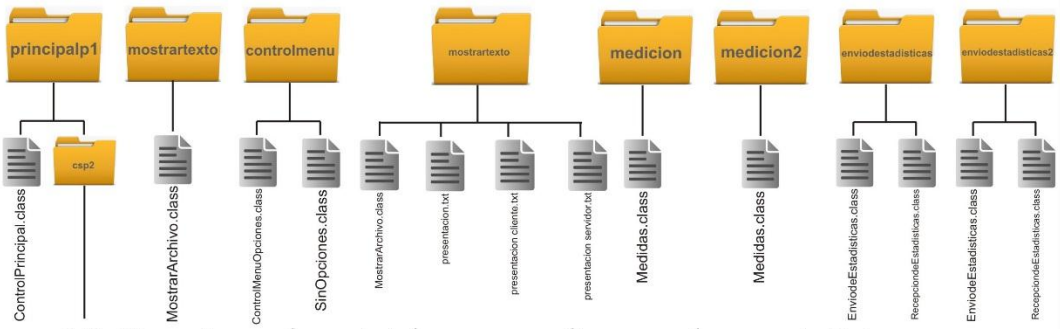


Figura 6.7a Paquetes y clases totales que resultaron en la carpeta Raíz.

De los siete paquetes, tres de ellos ya se explicaron (*principalp1*, *mostrartexto* y *controlmenu*) y quedan por explicar cuatro, los cuales son: *medición*, *medición2*, *enviodestadisticas* y *enviodestadisticas2*. Los paquetes *medicion* y *medicion2* son iguales porque ambos contienen la misma clase *Medidas.class*, la diferencia es que el paquete *medicion* es utilizado por el cliente, en cambio el paquete *medicion2* es utilizado por el servidor, esto permite distinguir mejor el rol de cada paquete en la programación del software.

- *Medidas.class*: clase que contiene las funciones que miden la velocidad de transferencia de datos.

Los paquetes *enviodestadisticas* y *enviodestadisticas2* son iguales porque ambos contienen las mismas clases *EnviodeEstadisticas.class* y *RecepciondeEstadisticas.class*, la

diferencia es que el paquete *enviodestadisticas* es utilizado por el cliente, en cambio el paquete *enviodestadisticas2* es utilizado por el servidor.

- *EnviodeEstadisticas.class*: clase que contiene la función que envía las medidas de velocidad entre cliente y servidor, si la medición se hace en el cliente, entonces esta función se activara en el cliente para enviar las medidas hacia el servidor o si la medición se hace en el servidor, entonces esta función se activara en el servidor para enviar las medidas hacia el cliente.
- *RecepciondeEstadisticas.class*: clase que contiene la función que recibe las medidas de velocidad entre cliente y servidor, si la medición se hace en el cliente, entonces esta función se activara en el servidor para recibir las medidas del cliente, o si la medición se hace en el servidor, entonces esta función se activara en el cliente para recibir las medidas del servidor.

Se puede decir que *EnviodeEstadisticas.class* y *RecepciondeEstadisticas.class* son inversas, es decir, a la parte que le corresponda hacer las medidas de velocidad (ancho de banda) deberá tener activa la función que está dentro de *EnviodeEstadisticas.class* para enviar las muestras de velocidad a la otra parte, mientras la otra parte deberá tener activa la función que está dentro de *RecepciondeEstadisticas.class* para recibir dichas muestras. De la carpeta raíz solo falta por explicar los nuevos archivos de texto del paquete *mostrartexto*, que ya se explicó en la parte 1, pero para la parte 2 y 3 aparecen nuevos archivos de texto tales como:

- *presentación cliente.txt*: archivo de texto que contiene el mensaje de presentación de la parte del cliente.
- *presentación servidor.txt*: archivo de texto que contiene el mensaje de presentación de la parte del servidor.

Todos los paquetes y clases de la carpeta raíz ya se trataron, ahora se explicaran los elementos que se encuentran dentro del paquete *principalp1* y corresponden exclusivamente a la parte del cliente y del servidor. El sub-paquete *csp2* contiene dos sub-paquetes: *clientep3* y *servidorp3*, el primero agrupa toda la información del cliente y el segundo del servidor.

Figura 6.7b Acercamiento de Vista de la Fig. 6.7 hacia los Sub-Paquetes del Paquete *principalp1*

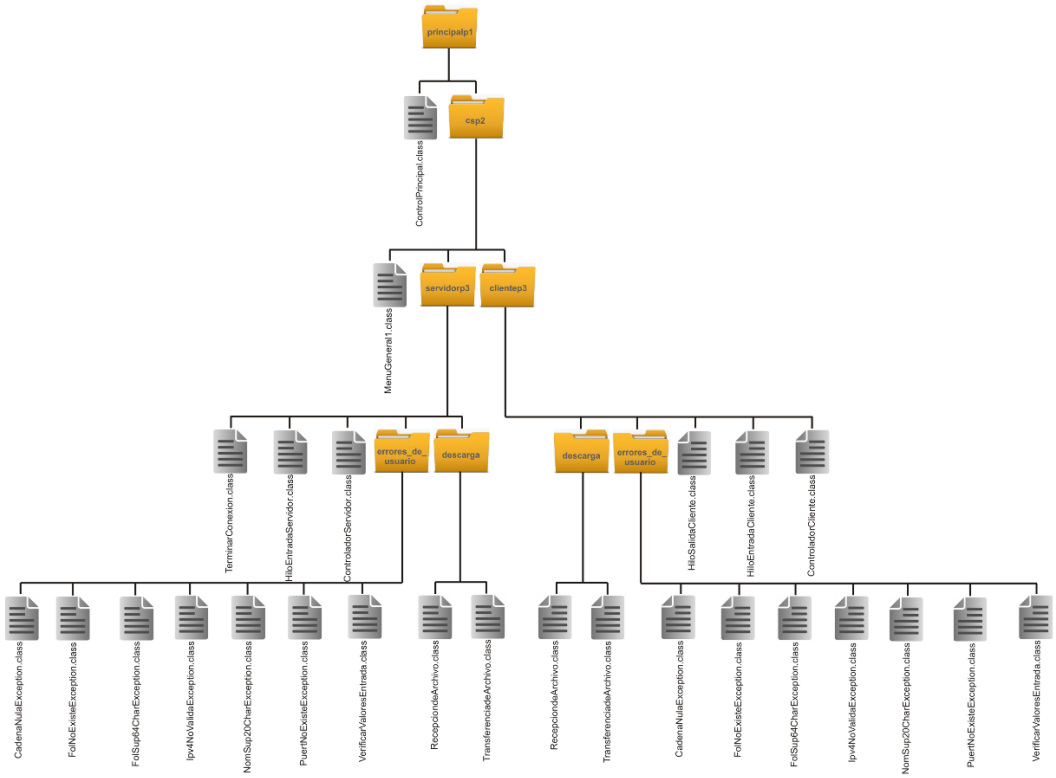


Figura 6.7b Sub-paquetes del paquete *principalp1*.

Ahora se inicia la explicación de los sub-paquetes y clases que son del cliente, por lo tanto, el sub-paquete *clientep3* y todo su contenido se muestra en la Fig. 6.8.

Figura 6.8 Archivos y Carpetas de la Parte del Cliente

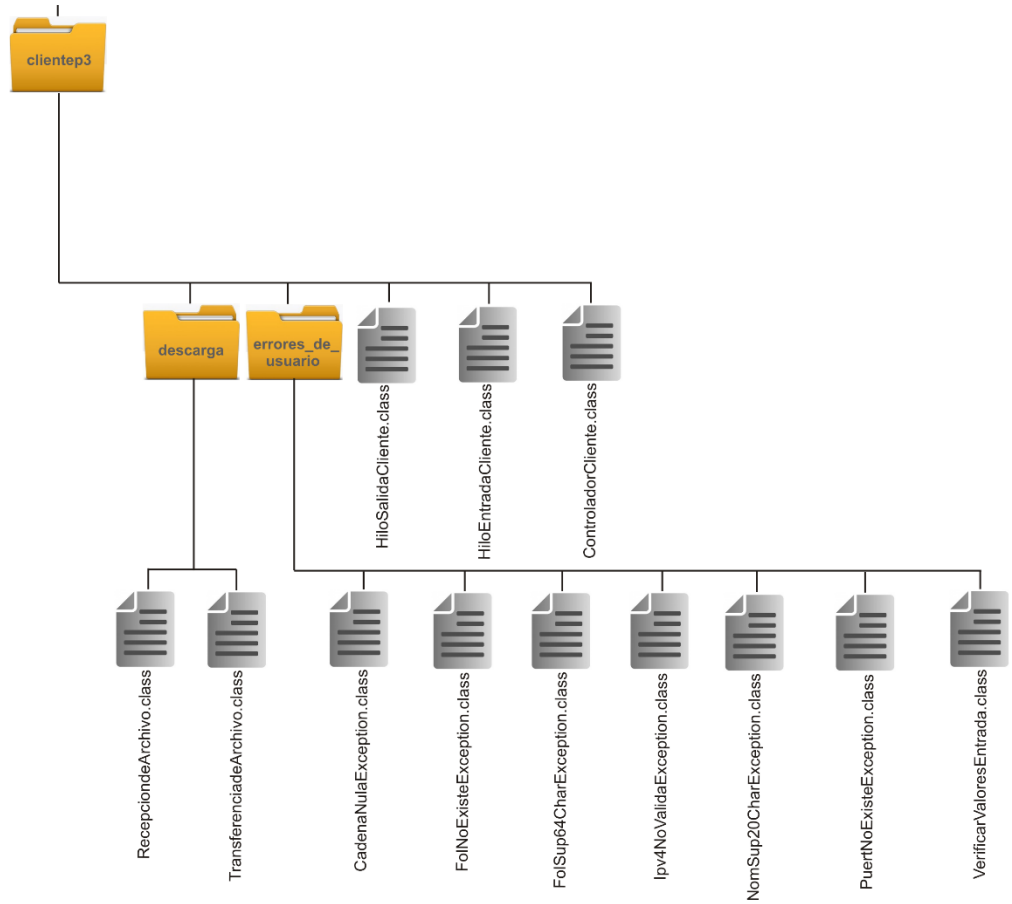


Figura 6.8 Diagrama que muestra los archivos solo de la parte del cliente.

El paquete *clientep3* contiene las siguientes clases:

- *ControladorCliente.class*: clase que contiene la función principal que controla la ejecución de todos los subprocesos que corresponden al cliente.
- *HiloEntradaCliente.class*: clase que contiene la función que controla la ejecución de los subprocesos que corresponden a la recepción de los datos.
- *HiloSalidaCliente.class*: clase que contiene la función que controla la ejecución de los subprocesos que corresponden al envió de los datos.

El paquete *clientep3* contiene dos sub-paquetes: *descarga* y *errores_de_usuario*, el primero contiene las clases que manejan la transmisión de datos y el segundo contiene las clases que manejan los errores.

El paquete *descarga* contiene las siguientes clases:

- *RecepciondeArchivo.class*: clase que contiene la función que hace la recepción de los datos de los archivos.
- *TransferenciadeArchivo.class*: clase que contiene la función que hace la transferencia de los datos de los archivos.

El paquete *errores_de_usuario* contiene las siguientes clases:

- *VerificarValoresEntrada.class*: clase que contiene la función que verifica la fidelidad de los datos insertados por el usuario cuando vaya a establecer conexión con el servidor.
- *PuertNoExisteException.class*: clase que contiene la función que verifica si el puerto ingresado por el usuario existe y está dentro del rango de valores permitidos.
- *NomSup20CharException.class*: clase que contiene la función que verifica que la entrada del nombre de usuario no supere los veinte caracteres.
- *Ipv4NoValidaException.class*: clase que contiene la función que verifica que la entrada de la dirección Ipv4 sea válida.
- *FolSup64CharException.class*: clase que contiene la función que verifica que la dirección de la carpeta asociada para almacenar los archivos tenga un nombre que no supere los sesenta y cuatro (64) caracteres.
- *FolNoExisteException.class*: clase que contiene la función que verifica que la carpeta asociada para almacenar los archivos exista.
- *CadenaNulaException.class*: clase que contiene la función que verifica que la entrada de datos no sea vacía.

En este punto se ha tratado todo el contenido de la parte del cliente. Ahora queda por tratar los elementos que corresponden a la parte del servidor. Los archivos dentro del paquete *servidorp3* se muestran en la Fig. 6.9.

Figura 6.9 Archivos y Carpetas de la Parte del servidor

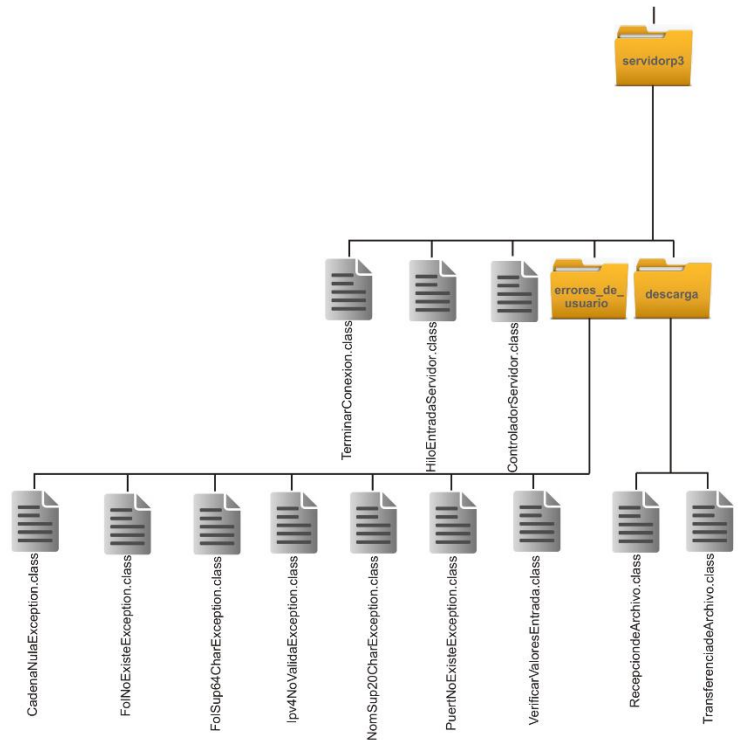


Figura 6.9 Diagrama que muestra los archivos solo de la parte del servidor.

Para la parte del servidor, el paquete *servidorp3* contiene las siguientes clases:

- *ControladorServidor.class*: clase que contiene la función principal que controla la ejecución de todos los subprocesos que corresponden al servidor.
- *HiloEntradaServidor.class*: clase que contiene la función que controla las funciones de atención a los clientes.
- *TerminarConexion.class*: clase que contiene la función que finaliza todos los subprocesos de atención a los clientes.

El paquete *servidorp3* contiene dos paquetes: *descarga* y *errores_de_usuario*, son exactamente iguales a los paquetes contenidos en *clientep3*, con la diferencia que en este caso éstos son utilizados por el servidor.

6.03.4.2.2 Funcionamiento. En la parte del cliente, la clase *ControladorCliente.class* que se encuentra dentro del paquete *clientep3*, contiene la primera función en ejecutarse, en ella se hace la entrada de datos necesaria para establecer la comunicación con el servidor, si ésta resulta satisfactoria entonces aparecen las clases *HiloEntradaCliente.class* e *HiloSalidaCliente.class*, la primera de ellas se encarga de procesar los datos que se reciben del servidor y la segunda se encarga de enviar los comandos al servidor, ambas clases se ejecutan simultáneamente porque la comunicación TCP/IP por ningún motivo puede presentar retardos ya que esto produciría una falla en tiempo de ejecución.

Cuando el cliente le envía un comando al servidor para la descarga de un archivo y el servidor le responde afirmativamente, la clase *HiloEntradaCliente.class* invoca a *RecepciondeArchivo.class* para dar inicio con la recepción del archivo. De igual forma si el cliente le envía un comando al servidor para la subida de un archivo y el servidor le responde afirmativamente, entonces la clase *HiloEntradaCliente.class* invoca a *TransferenciadeArchivo.class* para dar inicio con él envió del archivo hacia el servidor.

Ante la presencia de un error en cualquiera de las dos partes, intervienen las clases dentro del paquete *errores_de_usuario*, las cuales se encargan de controlar los errores que se puedan manejar, de lo contrario, si sucede un error grave que no se pueda manejar, como por ejemplo, un fallo de comunicación, entonces se produce una desconexión.

Figura 6.10 Diagrama de Funcionamiento de la Descarga de Archivos del Cliente con el Servidor con vista en el Cliente

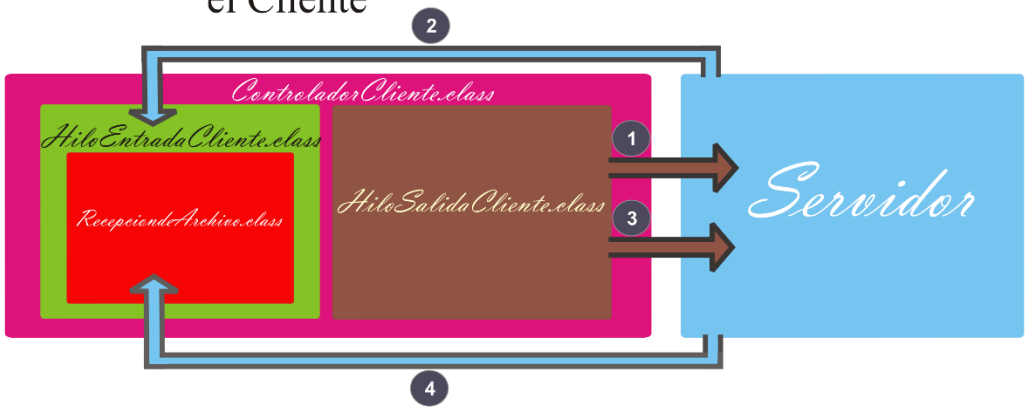


Figura 6.10 En el primer proceso, el cliente le envía al servidor una solicitud de la lista de archivos que el servidor tiene disponibles para compartir. En el segundo, el servidor le responde con dicha lista. El tercero consiste en que el cliente le envía al servidor el ID del archivo que quiere descargar. El cuarto proceso inicia la transeferencia del archivo desde el servidor hacia el cliente.

Simultáneamente mientras funciona el cliente, en la parte del servidor, la clase *ControladorServidor.class* que se encuentra dentro del paquete *servidorp3*, contiene la primera función en ejecutarse, dicha función escucha todas las peticiones que le llegan de los clientes y permite que cada cliente pueda conectarse simultáneamente con él servidor debido a que esta función invoca a la clase *HiloEntradaServidor.class*, la cual, crea subprocesos simultáneos e independientes para la atención de cada uno de los clientes y procesa los comandos de dichos clientes para controlar el servidor.

Cuando al servidor le llega un comando para descargar un archivo, éste activa la clase *TransferenciadeArchivo.class*, la cual se encarga de enviar el archivo hacia el cliente que corresponda. En cambio, cuando al servidor le llega un comando para recibir algún archivo, éste activa la clase *RecepciondeArchivo.class*, la cual se encarga de la recepción de los datos del archivo. La clase *TerminarConexion.class* se encarga de cerrar todas las conexiones con los clientes cuando el usuario quiera finalizar el programa.

Figura 6.11 Diagrama de Funcionamiento de la Descarga de Archivos del Cliente con el Servidor con vista en el Servidor

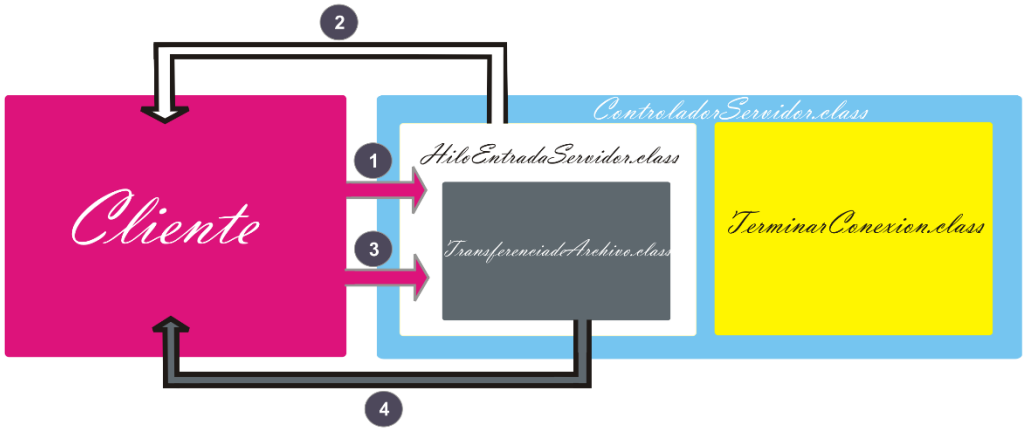


Figura 6.11 Explica el mismo proceso de la figura 6.10 pero con la diferencia que en esta se figura se muestra el funcionamiento desde el lado del servidor.

Figura 6.12 Diagrama de Funcionamiento de la Descarga de Archivos del Cliente con el Servidor con vista en el Cliente y en el Servidor

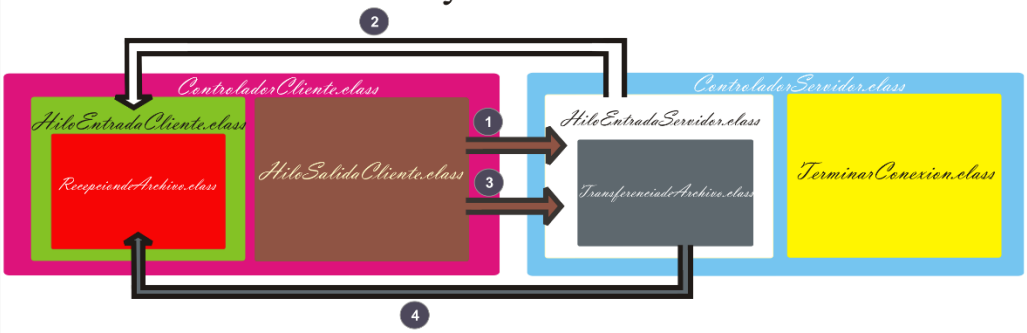


Figura 6.12 El proceso de las figuras 6.10 y 6.11 pero visto simultáneamente en el cliente y en el servidor.

Con la Fig. 6.12 queda expresada gráficamente la dinámica de la comunicación entre cliente y servidor para la transferencia de archivos.

6.03.5. Diagrama UML del Proyecto. En la explicación de los numerales anteriores acerca de las partes del proyecto, se presentaron diagramas en las Fig. 6.4, 6.6, 6.10, 6.11, 6.12, sin embargo dichos diagramas no corresponden con ninguna norma o estándar para modelar software. Todos los diagramas previos son simples simbolismos que pretenden dar a entender el flujo del programa y su modo de funcionamiento.

Es fundamental emplear diagramas que revelen con mayor detalle el modelo del programa y que obedezcan a un estándar para ser profundamente analizados por una comunidad de expertos, por lo tanto, a continuación se presenta un diagrama UML de paquetes y clases del programa, vale aclarar que los cuadros de las clases de este diagrama no muestran sus atributos porque harían al diagrama muy extenso y también más complejo. El diagrama UML siguiente representa la relación de los paquetes y las clases entre sí.

Figura 6.13 Diagrama UML del Proyecto

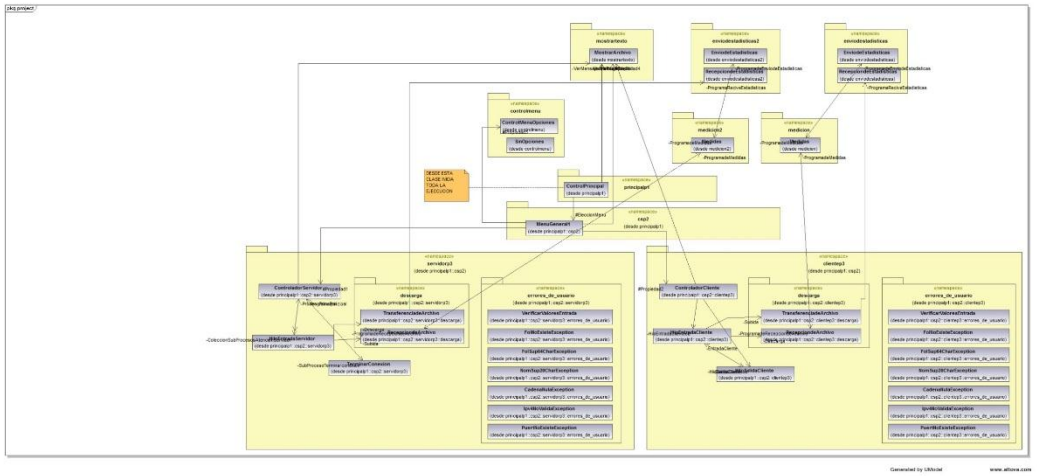


Figura 6.13 El diagrama UML contiene cuadros morados y amarillos que identifican a las clases y los paquetes.

Por limitaciones del tamaño de la página, los detalles del diagrama UML no se pueden observar claramente pero en el CD del proyecto está la imagen con la mejor resolución posible

para aquellos que quieran analizar el diagrama detalladamente, de igual forma está el archivo con extensión *.ump*, el cual contiene la información lógica del diagrama.

La interpretación del diagrama UML tiene los siguientes detalles:

- Los cuadros de color morado identifican las clases, mientras que los cuadros de color amarillo identifican los paquetes.
- Los cuadros amarillos de los paquetes encierran a los cuadros morados de las clases que le corresponden, sin embargo, por cuestiones de presentación y tamaño, los paquetes *principalp1* y *csp2* no encerraron sus paquetes y clases internas debido a que eran demasiadas pero se simbolizaron haciendo alusión a que estos son los contenedores principales de los paquetes y clases correspondientes a ellos.
- Las flechas del diagrama no indican el flujo del programa sino los vínculos entre las clases. La flechas unidireccionales indican que una clase invoco elementos de otra clase, pero la otra clase no invoco elementos de la primera. Las flechas bidireccionales indican que una clase invoco elementos de otra clase y de la misma forma la otra clase también invoco elementos de la primera, es decir que ambas clases se invocaron elementos entre sí.
- El punto de partida del programa está en la clase *ControlPrincipal.class* que se encuentra dentro del paquete *principalp1*.
- Las clases correspondientes al manejo de los errores que se encuentran en los paquetes *errores_de_usuario* y también la clase *SinOpciones.class* que está dentro del paquete *mostrartexto* , no tienen vínculos porque estas son activadas solo cuando ocurren excepciones o errores inesperados en el programa, mas no hacen parte de los procedimientos operativos del mismo.

Es apenas lógico que sea necesario ver el código fuente para entender mejor la coherencia del programa. El código fuente del programa viene en el CD del proyecto y puede ser

usado para que cada persona interesada modele el código en diagramas UML al gusto personal. Si queda alguna duda de la fidelidad del presente diagrama UML con relación al código fuente del programa aclaro que este se elaboró con el programa especializado en modelamiento de software conocido como “Altova UModel”, y lo genero directamente del código fuente, es decir, no fue una esquematización manual.

6.03.6 Estructura de archivos del código fuente. El código fuente del proyecto está en la carpeta “Código Fuente Programa Final TCPIPTest” (en el CD) y en ella está la carpeta “src”. Vale aclarar que una cosa es la estructura de archivos del código fuente y otra muy distinta es la estructura de clases del programa, porque la primera corresponde a los archivos de extensión *.java* que son parte del código fuente mientras que las clases son los archivos resultantes de la compilación de dicho código fuente, además, la explicación de cada una de las clases y su estructura de paquetes se trató con profundidad en los numerales anteriores correspondientes a las partes del proyecto.

Figura 6.14 Carpeta src

Figura 6.14 La carpeta src contiene el código fuente del proyecto.

Dentro de la carpeta “src”, se encuentran las carpetas que conforman la estructura de archivos del código fuente y estos archivos corresponden a los paquetes que subdividen el programa mediante carpetas.

Figura 6.15 Carpetas del código Fuente

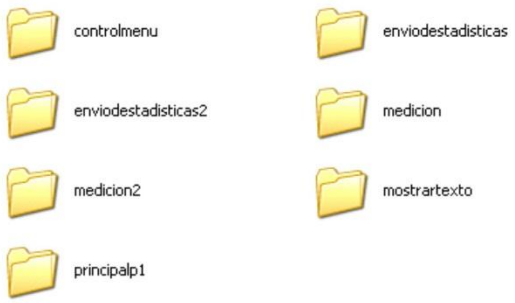


Figura 6.15 Las siete carpetas del código fuente corresponden a los siete paquetes del proyecto.

El paquete *controlmenu* contiene el archivo *ControlMenuOpciones.java*.

Figura 6.16 Código Fuente del Paquete ControlMenuOpciones

Figura 6.16

El archivo *ControlMenuOpciones.java* contiene el código de las clases *ControlMenuOpciones.class* y *SinOpciones.class*.

Figura 6.17 Código Fuente del Paquete EnviodeEstadisticas

Figura 6.17

Los paquetes (carpetas) *enviodestadisticas* y *enviodestadisticas2* son exactamente iguales, es decir, el código que contienen ambos es el mismo, el primero contiene el código que utiliza el cliente mientras el segundo contiene el código que utiliza el servidor.

El archivo *EnviodeEstadisticas.java* contiene el código de la clase *EnviodeEstadisticas.class* mientras el archivo *RecepciondeEstadisticas.java* contiene el código de la clase *RecepciondeEstadisticas.class*. Nótese que los nombres de los archivos *.java* del código fuente son iguales a los nombres de las clases que les corresponden, por convención esto se aplica en java pero no necesariamente en todos los casos.

Los paquetes (carpetas) *medicion* y *medicion2* son exactamente iguales, es decir, el código que contienen ambos es el mismo, pero el primero contiene el código que utiliza el cliente mientras el segundo contiene el código que utiliza el servidor.

Figura 6.18 Código Fuente del Paquete Medicion

Figura 6.18.

El archivo *Medidas.java* contiene el código de la clase *Medidas.class*.

El paquete *mostrartexto* contiene el archivo *MostrarArchivo.java*.

Figura 6.19 Código Fuente del Paquete Mostrartexto

Figura 6.19 El código fuente del paquete *mostrartexto* tiene cuatro archivos de texto que no son código fuente sino simples archivos de texto independientes, en cambio el archivo *MostrarArchivo.java* es el código fuente de este paquete.

Dentro del mismo paquete se encuentran archivos de texto como: *presentación.txt, presentacion servidor.txt, csp2.txt* y *presentación cliente.txt*. El archivo *MostrarArchivo.java* contiene el código de la clase *MostrarArchivo.class*.

El paquete *principalp1* contiene el archivo *ControlPrincipal.java*.

Figura 6.20 Contenido del Paquete Principalp1

Figura 6.20 Este paquete contiene el código de los modos de funcionamiento del cliente y del servidor.

El archivo *ControlPrincipal.java* contiene el código de la clase *ControlPrincipal.class*, además dentro del paquete *principalp1* está el paquete *csp2*. El paquete *csp2* contiene el archivo *MenuGeneral1.java*.

Figura 6.21 Contenido del Subpaquete Csp2

Figura 6.21 Este paquete agrupa el código exclusivamente del cliente y del servidor.

El archivo *MenuGeneral1.java* contiene el código de la clase *MenuGeneral1.class*, además dentro del paquete *csp2* están los paquetes *clientep3* y *servidorp3*.

El paquete *clientep3* contiene los archivos *ControladorCliente.java*, *HiloEntradaCliente.java* e *HiloSalidaCliente.java*.

Figura 6.22 Contenido del Paquete Clientep3

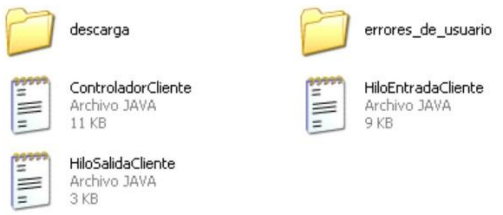


Figura 6.22

El archivo *ControladorCliente.java* contiene el código de la clase *ControladorCliente.class*. El archivo *HiloEntradaCliente.java* contiene el código de la clase *HiloEntradaCliente.class*. El archivo *HiloSalidaCliente.java* contiene el código de la clase *HiloSalidaCliente.class*. El paquete *clientep3* contiene dos paquetes que son: *descarga* y *errores_de_usuario*.

El paquete *descarga* contiene los archivos *RecepciondeArchivo.java* y *TransferenciadeArchivo.java*.

Figura 6.23 Contedino del Paquete Descarga

Figura 6.23

El archivo *RecepciondeArchivo.java* contiene el código de la clase *RecepciondeArchivo.class*. El archivo *TransferenciadeArchivo.java* contiene el código de la clase *TransferenciadeArchivo.class*.

El paquete *errores_de_usuario* contiene los archivos *VerificarValoresEntrada.java* y *CadenaNulaException.java*.

Figura 6.24 Contenido del Paquete Errores_de_usuario

Figura 6.24

El archivo *VerificarValoresEntrada.java* contiene el código de la clase *VerificarValoresEntrada.class*. El archivo *CadenaNulaException.java* contiene el código de seis clases, las cuales son: *CadenaNulaException.class*, *NomSup20CharException.class*, *FolSup64CharException.class*, *FolNoExisteException.class*, *Ipv4NoValidaException.class* y *PuertNoExisteException.class*.

Las preguntas que surgen son ¿Por qué *CadenaNulaException.java* tiene seis clases y no una?, también, ¿Porque *CadenaNulaException.java* tiene el nombre de una de las clases que contiene y de las demás no?, para la primera pregunta, el archivo *CadenaNulaException.java* agrupa el código que maneja los errores principales del programa en la parte del cliente para este caso, y dichos errores se clasificaron en varias clases y por lo tanto resulto mejor agrupar el código en un archivo.

Para la segunda pregunta, el nombre del archivo de código fuente por convención debe tener el mismo nombre de la clase que le corresponde, pero contiene varias clases, entonces se pueden hacer dos cosas, la primera es asignarle un nombre que generalice las clases que contiene, pero el problema de hacer eso es que al momento de hacer la compilación de todo el programa, el compilador no identifica a que clases especificas corresponde dicho archivo de código fuente y lo ignora, por lo tanto tocaría compilarlo de manera independiente del resto del programa y eso es molesto.

La segunda cosa es asignarle un nombre de cualquier clase que contenga, así el compilador identificaría el archivo de código fuente con alguna clase en particular y al momento

de evaluar el código, lograría identificar todas las clases adicionales que contiene, y por lo tanto las interrelacionaría con el resto del programa y todas serian compiladas.

Volviendo al paquete *csp2*, está el paquete *servidorp3*. El paquete *servidorp3* contiene los archivos *ControladorServidor.java*, *HiloEntradaServidor.java* y *TerminarConexion.java*.

Figura 6.25 Contenido del Paquete Servidorp3

Figura 6.25

El archivo *ControladorServidor.java* contiene el código de la clase *ControladorServidor.class*. El archivo *HiloEntradaServidor.java* contiene el código de la clase *HiloEntradaServidor.class*. El archivo *TerminarConexion.java* contiene el código de la clase *TerminarConexion.class* .El paquete *servidorp3* contiene dos paquetes que son: *descarga* y *errores_de_usuario*, los cuales son la copia de los mismos que hay en el paquete *clientep3* pero en este caso, estos pertenecen al paquete *servidorp3* porque aplican en el servidor. Con lo anterior se da por concluida la explicación de toda la estructura de archivos y carpetas del código fuente.

6.03.7. Código fuente de los procesos esenciales del software. Como se mencionó al inicio de la explicación de las partes, el código fuente no se trata por completo en este documento debido a que el entendimiento del mismo exige conocimientos avanzados de programación, sin embargo, el código que hace transmisión, recepción, medición de tráfico, cálculo de promedio, cálculo de varianza y cálculo de desviación estándar, se muestra y explica en este numeral pero solamente los fragmentos que realizan dichas funciones mas no el código completo de las clases

correspondientes, por lo tanto, las instrucciones sobrantes se omiten ya que se relacionan más con la estructura, el flujo y las variables internas del programa que con los procedimientos esenciales.

6.03.7.1. Fragmentos de código. A continuación se muestran los fragmentos de código de las clases que realizan los procedimientos esenciales.

6.03.7.1.1. Fragmento de código de transmisión. El código siguiente corresponde al procedimiento que hace la transmisión de datos. Este código se encuentra alojado en el CD del proyecto, dentro de la carpeta "Código Fuente Programa Final TCPIPTest" en *src\principalp1\csp2\clientep3\descarga\TransferenciadeArchivo.java*, y ocupa el espacio entre la línea 30 y la 36. Dicho código corresponde a la clase *TransferenciadeArchivo.class*.

```
this.TramadeSalida = this.SocketUsuario.getOutputStream();
this.Archivo = new FileInputStream(this.NombredelaCarpeta + "/" + this.NombredelArchivo);
byte[] buffer = new byte[this.SocketUsuario.getSendBufferSize()];
int bytesLeidos = 0;
while((bytesLeidos = this.Archivo.read(buffer))>0){
        this.TramadeSalida.write(buffer,0,bytesLeidos);
}
```

En el código anterior, la primera línea activa una función de java llamada "socket" que hace la transmisión TCP. En la segunda línea se declara una variable de trama de datos con el archivo que se va a transmitir y se definió en procesos previos. La tercera línea declara una variable que encapsula en bytes el archivo que se van a transmitir, la cuarta línea declara una variable que cuenta los bytes leídos del archivo que se va a transmitir. En la quinta está el ciclo que hace la transmisión de datos, el cual evalúa la cantidad de bytes leídos, si es mayor a cero, entonces continua la escritura de los bytes del archivo en el buffer de transmisión, si es cero entonces la transmisión finaliza.

6.03.7.1.2. Fragmento de código de recepción. El código siguiente corresponde al procedimiento que hace la recepción de datos. Este código se encuentra alojado en el CD del proyecto, dentro de la carpeta "Código Fuente Programa Final TCPIPTest" en *src\principalp1\csp2\clientep3\descarga\RecepciondeArchivo.java* y ocupa el espacio entre la línea 37 y la 43. Dicho código corresponde a la clase *RecepciondeArchivo.class*.

```
this.TramadeEntrada = this.SocketUsuario.getInputStream();
this.Archivo = new FileOutputStream(new File(this.NombredelaCarpeta + "/" + this.NombredelArchivoEntrada));
byte[] buffer = new byte[this.SocketUsuario.getReceiveBufferSize()];
while((this.bytesRecibidos = TramadeEntrada.read(buffer))>0){
        this.bytesAcumulados += this.bytesRecibidos;
        this.Archivo.write(buffer, 0, this.bytesRecibidos);
}
```

En el código anterior, la primera línea activa una función de java llamada "socket" que hace la recepción TCP. En la segunda línea se declara una variable de trama de datos que va a construir el archivo recibido. En la tercera línea se declara una variable que encapsula en bytes los datos recibidos de la transmisión. En la cuarta línea está el ciclo que hace la recepción, el cual evalúa si los bytes recibidos son mayor que cero, si se cumple la condición entonces se cuentan los bytes y se reciben los datos y con ellos se construye el archivo. Cuando los bytes recibidos son cero, entonces el proceso de recepción termina.

6.03.7.1.3. Fragmento de código que captura muestras del tráfico de la comunicación. El código siguiente corresponde al procedimiento que lee el buffer de recepción para saber la cantidad de bytes acumulados en la sesión de transferencia. Este código se encuentra alojado en el CD del proyecto, dentro de la carpeta "Código Fuente Programa Final TCPIPTest" en *src\medicion\Medidas.java*, y ocupa el espacio entre la línea 61 y la 67. Dicho código corresponde a la clase *Medidas.class*.

```
public void TomarMuestras() throws Exception{
    this.Muestras.add(this.ProgramadeRecepciondeArchivo.ObtenerbytesAcumulados());
```

```
    Thread.sleep(1000);
    this.Muestras.add(this.ProgramadeRecepciondeArchivo.ObtenerbytesAcumulados());
}
```

En el código anterior, la primera línea inicia la función de lectura de los bytes recibidos, dicha función se llama "TomarMuestras". En la segunda línea se leen los bytes acumulados en el buffer de recepción y dicho valor se aguarda en un vector llamado "Muestras". En la tercera línea el proceso entra en suspensión durante un segundo, con el fin de que las medidas de velocidad se midan en segundos. En la cuarta línea se repite el mismo proceso de la segunda línea, es decir, se hace una segunda captura de los bytes acumulados en el buffer de recepción y se guarda en un nuevo campo del vector "Muestras", es importante aclarar que dicho buffer incrementa su valor con el paso del tiempo porque en él se cuentan los bytes que se van acumulando desde el inicio hasta el final de toda la sesión de transferencia. En la quinta línea se cierra la función de lectura de bytes.

6.03.7.1.4. Fragmento de código que mide el tráfico de la comunicación. El código siguiente corresponde al procedimiento que calcula la cantidad de bytes que se reciben segundo a segundo en la sesión de transferencia. Este código se encuentra alojado en el CD del proyecto, dentro de la carpeta "Código Fuente Programa Final TCPIPTest" en *src\medicion\Medidas.java*, y ocupa el espacio entre la línea 81 y la 86. Dicho código corresponde a la clase *Medidas.class*.

```
public long MedirVelocidad() throws Exception{
                this.MuestrasdeVelocidad.add(this.Muestras.elementAt(this.Muestras.size() - 1) - this.Muestras.elementAt(this.Muestras.size() - 2));
                return this.MuestrasdeVelocidad.elementAt(this.MuestrasdeVelocidad.size() - 1);
}
```

En el código anterior, la primera línea inicia la función de medida de tráfico, dicha función se llama "MedirVelocidad". En la segunda línea se opera la resta entre el último dato agregado al vector "Muestras" con el anterior y el resultado se asigna a un nuevo vector llamado

"MuestrasdeVelocidad". Debido a que el vector "Muestras" almacena los datos que corresponden a la cantidad de bytes acumulados en la recepción, es necesario que se calcule la diferencia entre la última muestra del vector con la anterior para conocer la velocidad de captura por segundo y además de eso, dicha muestra no se debe perder, por lo tanto es necesario que se almacene en el vector "MuestrasdeVelocidad". Debido a que la segunda línea es larga, ocupa dos líneas (hasta la tercera). En la cuarta línea, el último valor almacenado en el vector "MuestrasdeVelocidad" es devuelto y en la quinta línea se cierra la función que mide la velocidad.

6.03.7.1.5. Fragmento de código que calcula el promedio. El código siguiente corresponde al procedimiento que calcula el promedio de velocidad de transmisión de datos. Este código se encuentra alojado en el CD del proyecto, dentro de la carpeta "Código Fuente Programa Final TCPIPTest" en *src\medicion\Medidas.java,* y ocupa el espacio entre la línea 97 y la 127. Dicho código corresponde a la clase *Medidas.class*.

```
public float MedirPromediodeVelocidad() throws Exception{
		long Sumatoria = 0;
		int ContadorMuestra = 0;
		ProcedimientodeCalculoPromedio:{
			if(this.MuestrasdeVelocidad.size() >= 4){
				for(long ValorVelocidad: this.MuestrasdeVelocidad){
					if(ContadorMuestra > 0 && ContadorMuestra < this.MuestrasdeVelocidad.size() - 1){
						Sumatoria += ValorVelocidad;
					}
					ContadorMuestra++;
				}
				this.CalculodePromedio = (float)Sumatoria/(this.MuestrasdeVelocidad.size() - 2);
				break ProcedimientodeCalculoPromedio;
			}
			else{
				for(long ValorVelocidad: this.MuestrasdeVelocidad){
					Sumatoria += ValorVelocidad;
				}
				this.CalculodePromedio = (float)Sumatoria/this.MuestrasdeVelocidad.size();
				break ProcedimientodeCalculoPromedio;
			}
		}
		return this.CalculodePromedio;
```

}

En el código anterior, en las primeras tres líneas se declaran las variables fundamentales del proceso, la que almacena la sumatoria del promedio y la que cuenta las muestras que se van sumando. En la línea 5 hay un condicional que divide el proceso en dos, el objetivo del condicional es evaluar si la cantidad de muestras de la transferencia de datos supero las cuatro muestras de velocidad. Si se cumple la condición entonces la transferencia tuvo más de 4 muestras, por lo tanto, el proceso recorre la colección de muestras capturadas de la medición haciendo la sumatoria de todas las muestras excepto la primera y la última, la razón de esto es porque en una transferencia de archivos, la primera y la última muestra no miden el verdadero "overhead" de la comunicación y por lo tanto se ignoran.

Si el condicional es falso entonces el programa se dirige a la línea 15 donde está la instrucción "else", por lo tanto la transferencia fue corta y tuvo 4 muestras de velocidad o menos, por lo tanto, ninguna muestra se puede ignorar sino se suman todas. Al finalizar ambos procesos, tanto el afirmativo como el negativo, se calcula el promedio y se devuelve el valor final del mismo.

6.03.7.1.6. Fragmento de código que calcula la varianza y la desviación estándar. El código siguiente corresponde al procedimiento que calcula la varianza y la desviación estándar. Este código se encuentra alojado en el CD del proyecto, dentro de la carpeta "Código Fuente Programa Final TCPIPTest" en *src\medicion\Medidas.java*, y ocupa el espacio entre la línea 129 y la 162. Dicho código corresponde a la clase *Medidas.class*.

```
public float DesviacionEstandar() throws Exception{
		float Sumatoria = 0.0f;
		int ContadorMuestra = 0;
		ProcedimientodeCalculoDesviacionEstandar:{
			if(this.MuestrasdeVelocidad.size() >= 4){
```

```
                    for(long ValorVelocidad: this.MuestrasdeVelocidad){
                        if(ContadorMuestra > 0 && ContadorMuestra < this.MuestrasdeVelocidad.size() - 1){
                this.MuestrasdeVarianza.add((float)Math.pow((float)(ValorVelocidad - this.CalculodePromedio), 2));
                        }
                        ContadorMuestra++;
                    }
                    break ProcedimientodeCalculoDesviacionEstandar;
                }
                else{
                    for(long ValorVelocidad: this.MuestrasdeVelocidad){
                this.MuestrasdeVarianza.add((float)Math.pow((float)(ValorVelocidad - this.CalculodePromedio), 2));
                    }
                    break ProcedimientodeCalculoDesviacionEstandar;
                }
            }
            for(float ValorVarianza: this.MuestrasdeVarianza){
                Sumatoria += ValorVarianza;
            }
            this.Varianza = (float)Sumatoria/this.MuestrasdeVarianza.size();
            this.CalculodeDesviacionEstandar = (float)Math.sqrt(this.Varianza);
            return this.CalculodeDesviacionEstandar;
}
```

En el código anterior, en las primeras tres líneas se declaran las variables fundamentales del proceso, la que almacena la sumatoria de las muestras de la varianza y la que cuenta las muestras que se van sumando. En la línea 5 hay un condicional hay un condicional que divide el proceso en dos, el objetivo del condicional es evaluar si la cantidad de muestras de la transferencia de datos supero las cuatro muestras de velocidad. Si se cumple la condición entonces la transferencia tuvo más de 4 muestras, por lo tanto, el proceso recorre la colección de muestras capturadas de la medición haciendo la operación de las muestras de velocidad con el promedio, excepto con la primera y la última muestra, la razón de esto es porque en una transferencia de archivos, la primera y la última muestra no miden el verdadero “overhead” de la comunicación y por lo tanto se ignoran.

Si el condicional es falso entonces el programa se dirige a la línea 14 donde está la instrucción “else”, por lo tanto la transferencia fue corta y tuvo 4 muestras de velocidad o menos, por lo tanto, ninguna muestra se puede ignorar y entonces se operan todas las muestras de

velocidad con el promedio. Al finalizar ambos procesos, tanto el afirmativo como el negativo, se calcula la varianza con un ciclo que suma todas las muestras de la operación previa y con el resultado de dicha sumatoria se calcula la varianza y posteriormente la desviación estándar. Al final se devuelve el valor de la desviación estándar.

6.03.8. Proceso de compilación. Como se mencionó antes, en el CD del proyecto está el código fuente del software dentro de la carpeta "Código Fuente Programa Final TCPIPTest". El proceso de compilación de un programa en java consiste en convertir los archivos del código fuente que tienen el formato *.java* en archivos *.class,* los cuales contienen el famoso código de bytes que es ejecutado por el intérprete de java, vale aclarar que en java no se producen archivos ejecutables porque los archivos *.class* no se pueden ejecutar por sí mismos como lo hacen los archivos *.exe* generados por C o C++, sino que son estáticos y necesitan ser interpretados por la máquina virtual de java, es decir, que si la máquina virtual de java no está instalada en la computadora, no se pueden ejecutar aplicaciones de java en ella. La explicación del proceso de instalación de la JDK (máquina virtual y compilador) está en el manual del software que también se encuentra en el CD del proyecto. El proceso de compilación es el más importante de todos, ya que sin este, no podrían implementar el código otros estudiantes o ingenieros interesados. Siga los siguientes pasos:

1. Insertar el CD del proyecto y buscar la carpeta llamada "Código Fuente Programa Final TCPIPTest" y copiarla en el escritorio de su computadora.
2. Ingresar en la carpeta copiada al escritorio y dentro de ella crear una nueva carpeta llamada "bin", tal como se observa en la imagen siguiente.

Figura 6.26 Carpetas del código Fuente en el CD

Figura 6.26

3. Abrir la consola de Windows y en ella dirigirse dentro de la carpeta del proyecto que se copió al escritorio.

Figura 6.27 Ingresando a la carpeta del código fuente a travez de la consola

Figura 6.27

4. En la consola insertar el siguiente comando:

   "javac -d bin -cp src src\principalp1\ControlPrincipal.java"

Figura 6.28 Ingresando el comando de compilación en la consola

Figura 6.28

Esto es lo que significa cada instrucción:

- javac: es la instrucción para compilar un archivo. Independientemente de las demás opciones, esta instrucción se utiliza al inicio cuando se desea compilar cualquier archivo de java.

- -d: indica el lugar donde se desean colocar los archivos con extensión *.class* generados una vez realizada la compilación. En este caso, se está indicando que dichos archivos deben ser colocados dentro de la carpeta llamada "bin", que es la carpeta que se creó al inicio junto a la carpeta "src" (carpeta que contiene el código fuente) y el objetivo de su creación es agrupar los archivos generados de la compilación para que queden apartados del código fuente.
- -cp: indica el directorio donde se deben buscar las clases que se van a compilar. Esta opción se utiliza cuando se desean compilar clases ubicadas en carpetas distintas. Delante de esta instrucción se coloca la carpeta o lugar desde el cual se debe comenzar la búsqueda del código fuente. En este caso se está indicando que se debe comenzar a buscar desde la carpeta llamada "src". A continuación se escribe la ruta completa que lleva al compilador hacia la clase principal o la clase que contiene el método "main" que en este proyecto es la clase ControlPrincipal.java que se encuentra dentro del paquete *principalp1*. Es muy importante que durante la compilación se escriba la extensión *.java* del archivo que contiene el método "main".

Cuando se realiza el proceso de compilación, la consola da un salto de línea sin mostrar nada y finaliza cuando aparece nuevamente la ruta de la misma. Si durante el proceso de compilación aparecen mensajes, significa que hay errores.

5. Ingresar en la carpeta “bin” y observar todas las clases que se crearon en ella.
6. Copiar todos los archivos.txt que se encuentran en la dirección *src\mostrartexto* y pegarlos en la dirección *bin\mostrartexto*, tal como se muestra en las siguientes imágenes.

Figura 6.29 Moviendo los Archivos de Texto Independientes

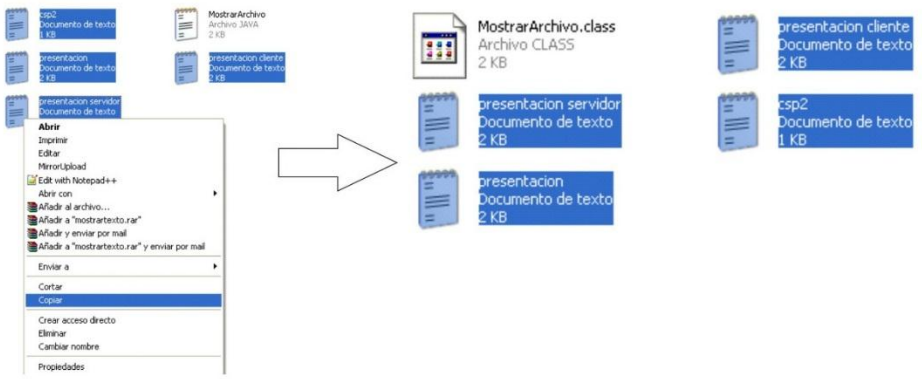


Figura 6.29

La generación de archivos .class en la carpeta “bin” durante el proceso de compilación no incluye los archivos de texto del paquete *mostrartexto* debido a que son simples archivos que se leen dentro del programa y solo actúan como elementos externos más no como código propiamente dicho, por lo tanto deben trasladarse manualmente al directorio *mostrartexto* de la carpeta “bin”.

7. En la consola insertar el siguiente comando:

“java -cp bin principalp1.ControlPrincipal”

Esto es lo que significa cada instrucción:

- java: es la instrucción que se coloca al principio siempre que se va a ejecutar un archivo ya compilado.
- -cp (o -classpath): cuando se ejecuta un programa, esta instrucción indica el lugar donde se debe buscar el archivo con extensión *.class* que el intérprete de java utiliza para ejecutar el proyecto. A continuación se indica la carpeta donde se debe buscar al archivo *.class* que servirá para correr el programa, que inicia con la carpeta "bin" (nótese que en la secuencia de compilado del paso 4, la instrucción -cp está seguida de "src"), seguida de un espacio en

blanco y a continuación la ruta para encontrar a dicho archivo, separando cada carpeta que conforma la ruta ahora con un punto en vez de una diagonal como se hizo en el proceso de compilación. Esta ruta dentro de la carpeta "bin" se creó automáticamente en el proceso de compilación y es idéntica a la ruta que se usó en la carpeta "src" para compilar el programa. Al final, se escribe el nombre del archivo principal, que en este caso es “ControlPrincipal” sin la extensión *.java*. El resultado se observa en la imagen siguiente.

Figura 6.30 Ejecución del Programa

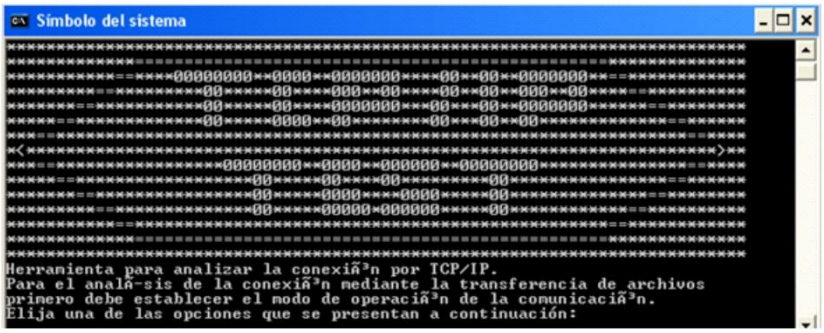


Figura 6.30

Como se observa en la imagen anterior, el programa está en ejecución, en este punto de la explicación ya se pueden hacer y probar modificaciones en el código del proyecto.

Capítulo 7

Resultados

El funcionamiento del programa final y las medidas de velocidad que este realizo en los canales de la empresa de energía, se presentan en este capítulo.

Configuración y Funcionamiento del Servidor

Figura 7.1 Inicio del Programa

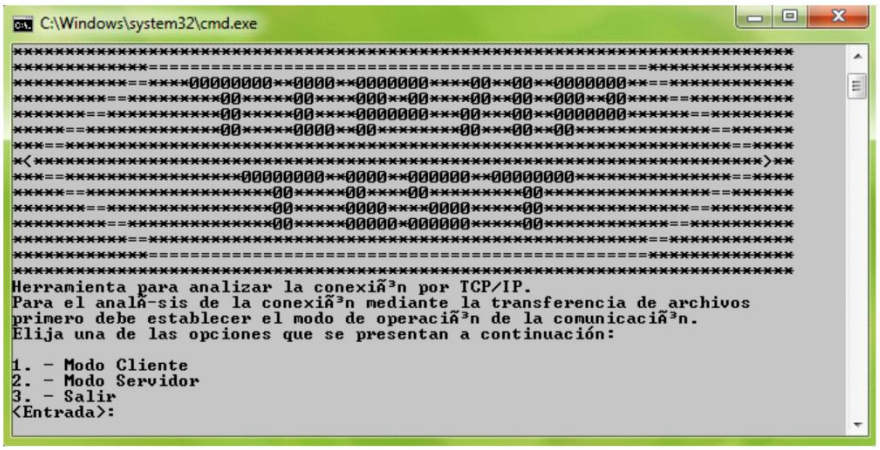
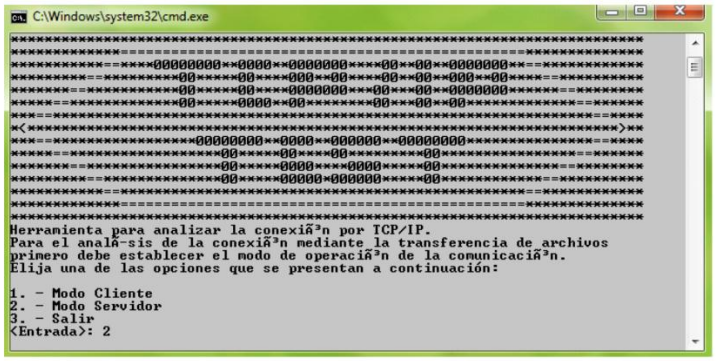


Figura 7.1 El inicio del programa muestra el letrero de presentación y un enunciado que explica los dos modos de funcionamiento que tiene, la primera opción escoge el modo cliente y la segunda el modo servidor, la tercera decide salir del programa.

Figura 7.2 Selección del modo Servidor

(a)

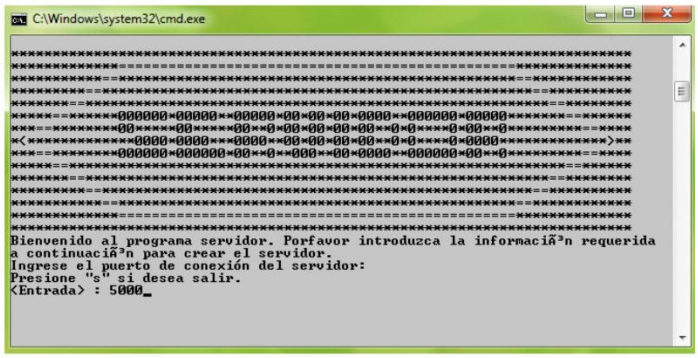


(b)

Figura 7.2 (a) Ingresa la opción 2 que selecciona el modo servidor, el cual debe ser el primero en activarse para que se puedan conectar los equipos.
(b) La función del servidor se presenta con un letrero de presentación y debajo de este hay un enunciado que guía al usuario para ingresar los datos de entrada.

Figura 7.3 Asignación del puerto de conexión

(a)



(b)

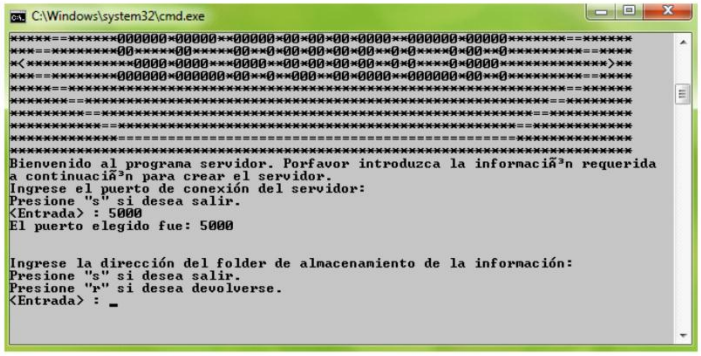
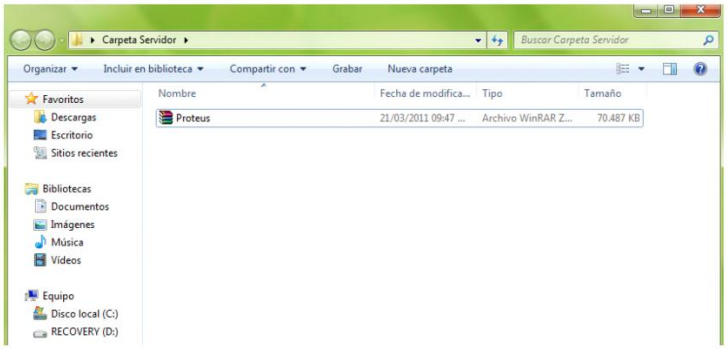


Figura 7.3 (a) Selección del puerto, el rango permitido esta entre 5000 y 65535. (b) Al presionar "Enter", pide el siguiente dato que corresponde a la dirección del folder donde estan los archivos que va a compartir con los clientes.

Figura 7.4 Asignación de la carpeta con archivos para compartir

(a)



(b)

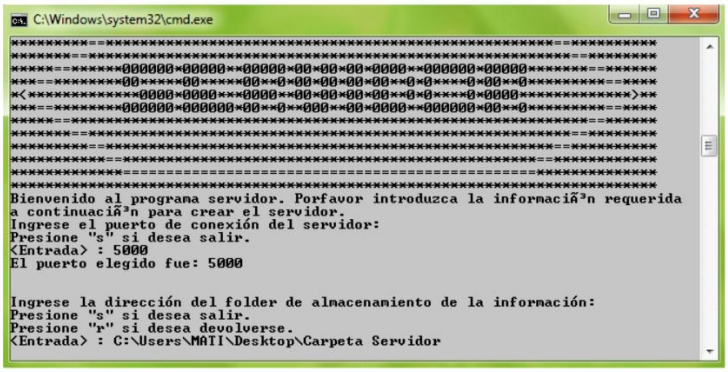


Figura 7.4 (a) Carpeta que va a compartir el servidor en la que se encuentra tan solo un archivo llamado "Proteus.zip" que es la información que sirve de elemento para la prueba . (b) Insertando la ruta en la linea de comandos donde se encuentra la carpeta de la figura (a).

Figura 7.5 Servidor esperando peticiones

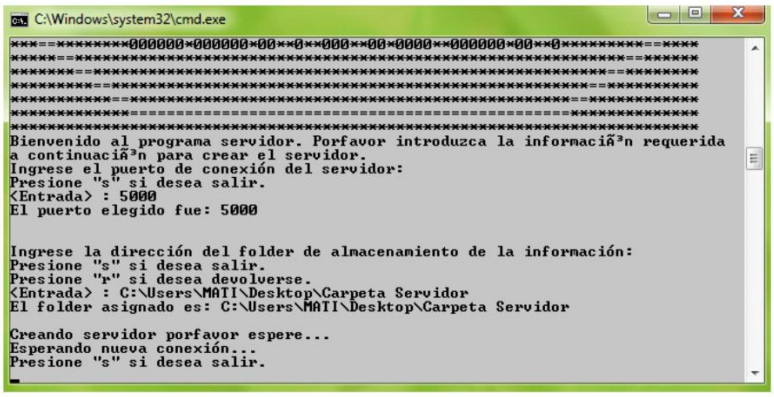


Figura 7.5 Una vez que se introducen los datos del puerto y la ruta de la carpeta con los archivos, el servidor permanece esperando las peticiones de los clientes.

Estableciendo Conexión

Figura 7.6 Activación del modo cliente

(a)

(b)

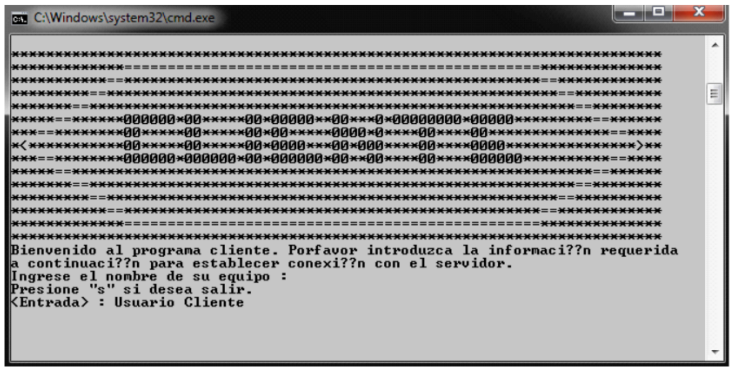


Figura 7.6 (a) La función del cliente se presenta con un letrero de presentación y debajo de este hay un enunciado que guía al usuario para ingresar los datos de entrada. (b) Insertando el nombre del usuario que corresponde al primer dato que solicita el cliente, en este caso es llamado como “Usuario Cliente”.

Figura 7.7 Asignación del puerto de conexión

(a)

(b)

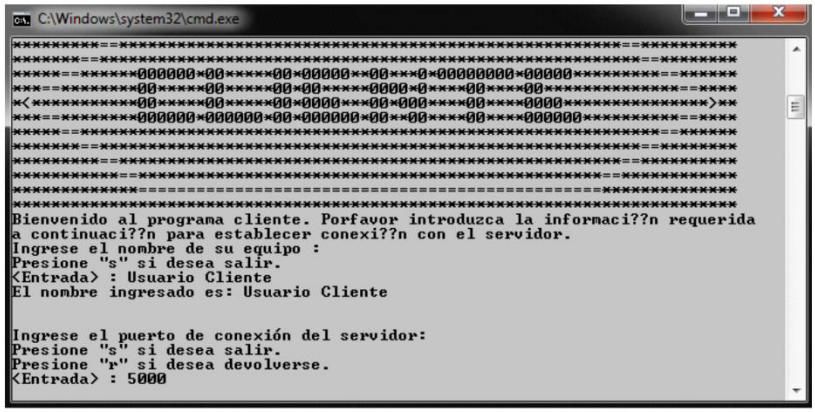
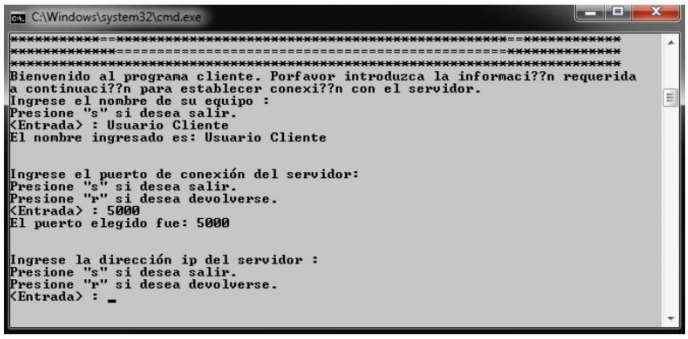


Figura 7.7 (a) El cliente solicita el ingreso del puerto del servidor al que quiere conectarse, en este caso seria 5000. (b) Ingreso del puerto 5000, mismo que se le asigno al servidor en la Fig. 7.3.

Figura 7.8 Insertando la dirección Ip

(a)



(b)

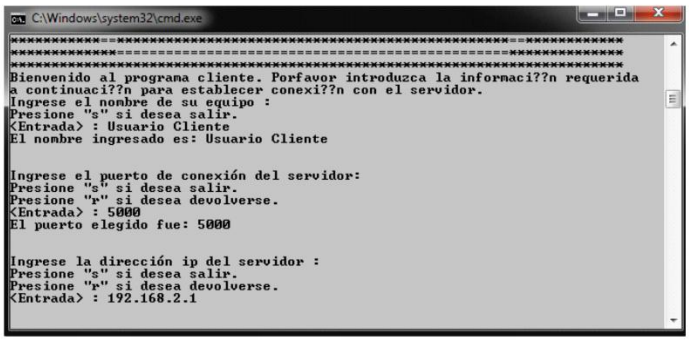
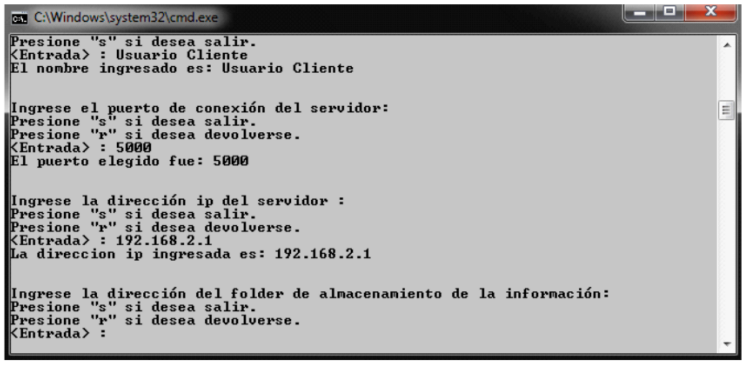


Figura 7.8. (a) Solicita la dirección Ip del servidor. (b) Ingresa la dirección Ip del servidor.

Figura 7.9 Ingresando la ruta de la carpeta donde el cliente va almacenar los archivos descargados del servidor

(a)



(b)

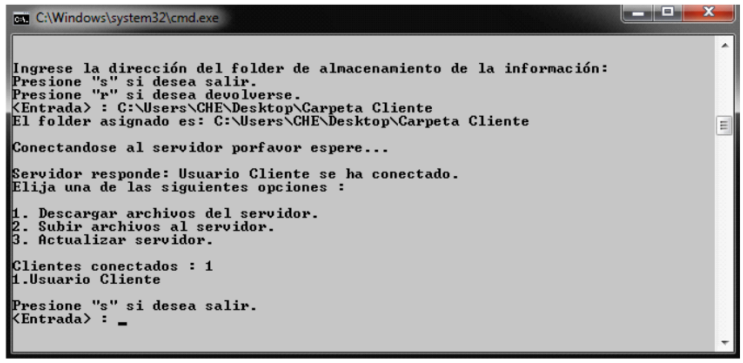


Figura 7.9 (a) Solicita la ruta de una carpeta que almacene los archivos descargados del servidor. (b) Una vez que se ingresa la ruta de dicha carpeta se establece conexión inmediata con el servidor, donde aparece un menu con tres opciones que el servidor le presenta al cliente y se explican mas adelante.

Figura 7.10 Menu de funciones

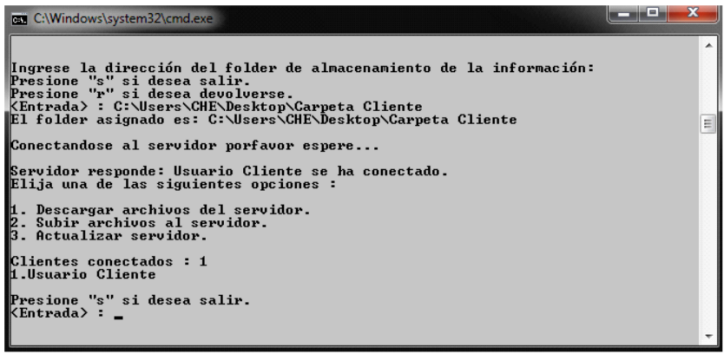
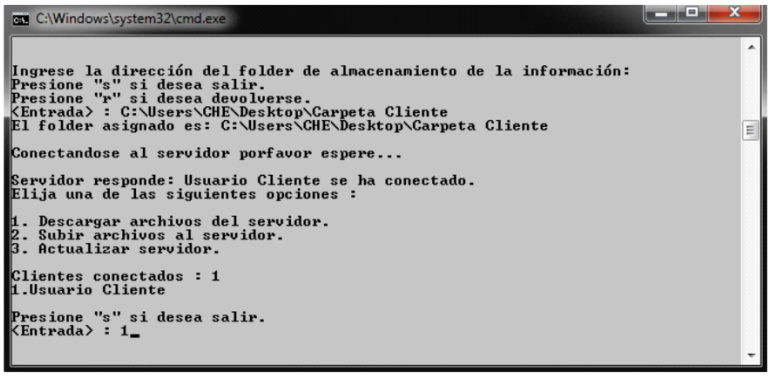


Figura 7.10 Una vez establecida la conexión entre cliente y servidor aparece un menu con tres opciones las cuales son, descargar archivos, subir archivos y actualizar servidor.

Midiendo el Ancho de Banda

Figura 7.11 Seleccionando la Opción de descargar Archivos.

(a)



(b)

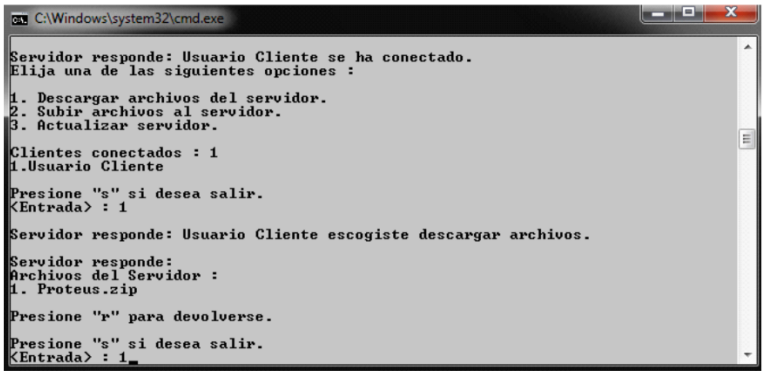


Figura 7.11 (a) Selección de la opción 1 que corresponde a la descarga de archivos. (b) Luego de seleccionar la opción de descarga de archivos, se despliega la lista de los archivos de la carpeta del servidor, entre ellos solo aparece “proteus.zip” tal como se muestra en la figura 4, por lo tanto se ingresa la unica opción existente (1) que corresponde a dicho archivo para dar inicio con la descarga.

Figura 7.12 Descarga del Archivo

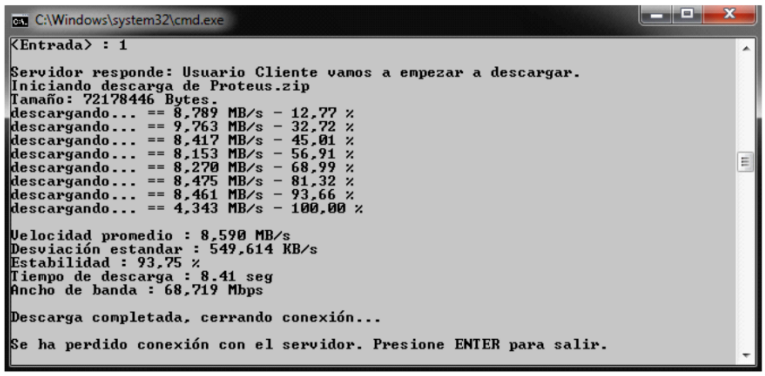


Figura 7.12 Cada segundo se visualizan muestras de velocidad y porcentaje. Al finalizar la transferencia se muestran los resultados finales de la velocidad promedio, la desviación estandar, el porcentaje de estabilidad, el tiempo de descarga y el ancho de banda.

Figura 7.13 Verificación

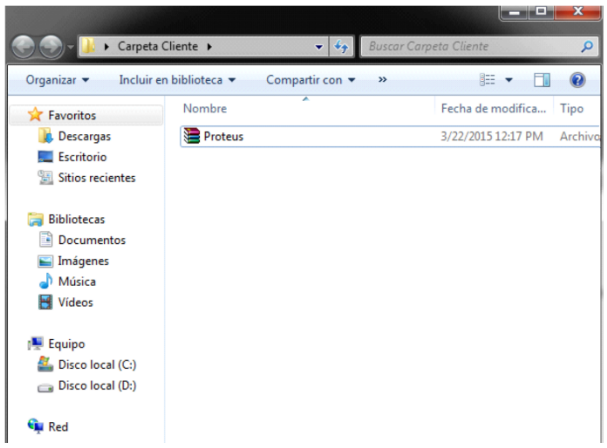
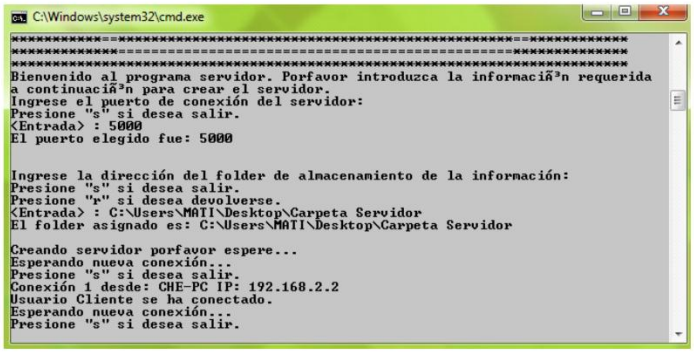


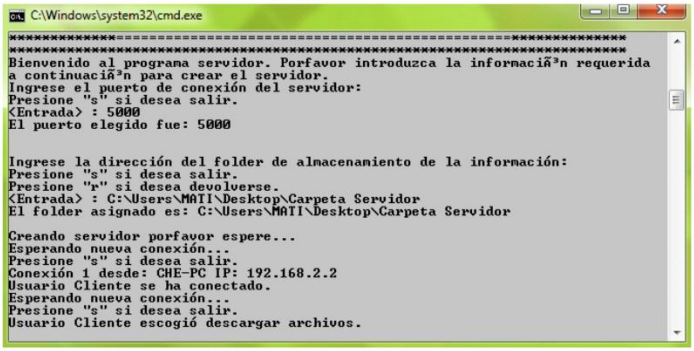
Figura 7.13 Una vez terminada la descarga se verifica la integridad del archivo de prueba, en este caso “proteus.zip” que se guarda en la carpeta que el cliente definio en la figura (9).

Figura 7.14 Visualización de eventos en el servidor

(a)



(b)



(c)

Figura 7.14 (a) El servidor visualiza todo lo que el cliente le ordena, en este caso se muestra un mensaje que indica que hosts se han conectado a el. (b) Visualiza las desiciones del cliente llamado “Usuario Cliente” correspondiente a la figura (6). (c) Visualiza las muestras de velocidad de descarga de “Usuario Cliente” e informa cuando este completa la descarga.

Medidas de velocidad de los canales de comunicación de EBSA

Tabla 4

Medidas de velocidad de canales inalámbricos de EBSA

Canal	Velocidad Media (Bps)	Ancho de Banda (bps)
Oficina – Subestación Sta. María	546.6K	4.27M
Oficina – Subestación Garagoa	5.28M	42.24M
Oficina – Subestación Villa de Leyva	1.4M	11.2M
Oficina – Subestación Belencito	3.31M	26.48M

Capítulo 8

Conclusiones

- La medición del throughput en el canal cableado patronado fue exitosa y como consecuencia de esto se cumplió con el objetivo general.
- El programa final cumplió con todos los requisitos mínimos y su mejor rendimiento se produce cuando la máquina virtual que lo carga es de la misma versión que aquella en que se compiló, en cambio, cuando se carga con versiones desactualizadas, el rendimiento presenta leves caídas en la velocidad de transmisión de datos. Una solución para tal inconveniente es volver a compilar el código fuente con versiones antiguas (versión 6 o anterior) de la máquina virtual sobre plataformas también antiguas como Windows XP para que sea más compatible. Con la implementación de los ajustes previamente expuestos, al usarse el programa en computadoras más potentes y con versiones más recientes de la máquina virtual, lograría conservar el rendimiento o inclusive mejorarlo.
- El libro de programación que se utilizó como referencia de estudio (Schildt Herbert, *Fundamentos de java 3° ed*, Mc Graw hill) resulto ser excelente porque con este se entendieron los temas con facilidad y además suministro más del 90% de la información necesaria para desarrollar el proyecto.
- Las interfaces de red de los equipos cliente y servidor deben tener una capacidad de transmisión de datos superior que la del canal que van a medir, de lo contrario la medida resultara incorrecta por las limitaciones de velocidad que puedan presentarse en dichas interfaces de red.
- La generación de trafico de los equipos cliente y servidor es directamente proporcional a la capacidad de CPU y memoria RAM de los mismos. Cada equipo debe procesar los bytes del archivo de la transmisión porque el lado transmisor debe leer los bytes del

archivo segundo a segundo para poderlos transmitir mientras el lado receptor debe construir el archivo segundo a segundo en la medida que recibe los datos, por lo tanto, si la capacidad de CPU y memoria RAM de los equipos de la medición es muy baja, se puede presentar un cuello de botella independiente del canal y de las interfaces de red, por lo tanto, los equipos de la medición deben ser lo suficientemente buenos como para superar la capacidad de transmisión de datos del canal que van a medir.

- Un programa que haga transmisión de datos obligatoriamente debe tener el proceso de comunicación funcionando de manera simultánea e independiente a los procesos que manejan la información, debido a que el proceso de comunicación no debe tener retardos por pequeños que sean porque dichos retardos limitaran las capacidades de generación de tráfico del equipo y afectaran la calidad de la comunicación.
- La conexión entre la oficina y la subestación Sta. María no es recomendable para la implementación de servicios de video IP porque de acuerdo a los estándares de MPEG4, con una calidad de movimiento mayor al 50% y 30 fps se consumen 1.5 Mbps por cada cámara, y teniendo en cuenta que el ancho de banda de los enlaces inalámbricos es variable con puntos altos y bajos en diferentes tiempos, aun si los 1.5 Mbps están por debajo de los 4.27 Mbps que indico la prueba, hay que tener en cuenta el “overhead” de las conexiones TCP que puede bajarle el rendimiento al resto de servicios que consume la subestación y aun si fuese funcional tan solo soportaría máximo una sola cámara que resultaría no viable para vigilar adecuadamente toda una subestación.
- La conexión entre la oficina y la subestación de Garagoa es recomendable para la implementación de video IP de acuerdo a los estándares de MPEG4, sin embargo es importante tener en cuenta que el ancho de banda de los enlaces inalámbricos es variable con puntos altos y bajos en diferentes tiempos. Aunque los resultados de la prueba

indicaron 42.27 Mbps se recomienda instalar tan solo un máximo de cinco cámaras debido a que el “overhead” en las conexiones TCP tiende a multiplicarse con cada dispositivo que se agrega al canal (previamente visto en el capítulo del “Marco teórico”) y no se desea degradar en ningún aspecto la calidad de los otros servicios de red que se comparten en el canal de la subestación.

- La conexión entre la oficina y la subestación Villa de Leiva obtuvo unos resultados de 11.2 Mbps, sin embargo, para hacer una buena vigilancia en la subestación es necesario operar con mínimo 4 cámaras, las cuales no se recomiendan instalar en dicha subestación porque como se mencionó antes, el “overhead” en TCP se multiplica cuando se agregan nuevas conexiones concurrentes, y la capacidad de la subestación no permite que 4 cámaras en MPEG4 a 1.5 Mbps sean instaladas porque la degradación que podrían generar entre todas afectaría seriamente la calidad de otros servicios compartidos.
- La conexión entre la oficina y la subestación Belencito cumple con las capacidades necesarias para implementar el servicio de video IP, sin embargo estaría en el límite de los requisitos para soportar 4 cámaras IP bajo el estándar MPEG4, en tal caso habría que evaluar qué tipo de servicios se están implementando en dicha subestación actualmente para saber si son sensibles a la degradación en caso de que se compartan junto con video IP.
- El modo de medida controlada con el programa consiste en ir transfiriendo archivos pequeños del orden de los KB e ir paulatinamente haciendo transferencias con archivos más grandes en el orden de los MB hasta llegar al punto de degradación de la capacidad de la conexión pero sin exceso y con ello conocer el ancho de banda disponible.

Capítulo 9

Recomendaciones

El aprendizaje debe ser el primer paso en el desarrollo de un proyecto y el que más necesita tiempo. Para entender con claridad los temas de estudio es importante que se practiquen a medida que se estudian o si no, no tendrán la utilidad relevante al momento de la elaboración del proyecto.

Definir un solo estilo de programación en toda la aplicación. A medida que el proyecto crezca en tamaño será más complejo, y sin un estilo que lo caracterice será más difícil encontrar errores así como hacerle mejoras, y como consecuencia se presentaran deficiencias en el funcionamiento del programa. El estrés al intentar entender un programa extenso, carente de estructura y con muchos estilos de programación producirá desánimo y cansancio en los desarrolladores, cosa que puede poner en riesgo la terminación del proyecto.

El estudio de los temas debe guiarse solo por una fuente bibliográfica. El uso de muchos libros para estudiar es bueno pero solamente si se desean reforzar conceptos haciendo que los conocimientos adquiridos sean más íntegros y completos, pero no para orientar el estudio. Es importante tener en cuenta que el aprendizaje tiene su tiempo límite y por lo tanto debe estar claramente marcado con un inicio y un final, así que si la guía de estudio esta referenciada por muchos libros, puede desorientar el camino del aprendizaje y no concluir en ninguna parte, como consecuencia pueden quedar temas trascendentales sin entenderse con claridad o en el peor de los casos sin estudiarse, además, el consumo de tiempo sería abismalmente mayor que con una sola guía de estudio.

Referencias

C. Dovrolis, P. R. (2004). *Packet-dispersion techniques and a capacity-estimation methodology.*

D. Lu, Y. Q. (2005). *Modeling and taming parallel tcp on the wide area network.* Denver: Parallel and Distributed Processing Symposium.

E. Altman, D. B. (2006). *Parallel tcp sockets: Simple model, throughput and validation.* Barcelona: 25th IEEE International Conference on Computer Communications.

ETSI. (2000). *202 057-4 v.1.1.1: Speech processing, transmission and quality aspects (STQ); user related QoS parameter definitions and measurements; part 4: internet access.* Madrid.

Guerra Bustillo, C. W. (2011). *Estadística.* Editorial Félix Varela.

J. Padhye, V. F. (2000). *Modeling tcp reno performance:a simple model and its empirical validation.* Networking, IEEE/ACM Transactions on 8.

Jacobson, V. (1988). *Congestion avoidance and control.*

James, E., & Bramante, R. (2009). *Networking OSI,TCP,IP,LAN.* Indianapolis: Wiley Publishing Inc.

Keshav, S. (1991). *A control-theoretic approach to flow control.* Zurich.

Margarit, R. M. (2003). *Los medios de comunicación en la era de la globalización, en América Latina* (Vol. 2). San jose: Red Universidad Nacional de Costa Rica.

Murillo Morera, J. d. (2010). *Implementación de un servidor ftp utilizando el modelo cliente/servidor mediante el uso de sockets en lenguaje C UNIX con el fin de mejorar los tiempos de respuesta en la red.* San jose: Red Universidad Nacional de Costa Rica.

Paxson, V. (1996). *Measurements and analysis of end-to-end internet dynamics phd. thesis.* U.C. Berkeley: Berkeley.

Schildt, H. (2007). *Fundamentos de java.* Mexico D.F: Mc Graw hill.

T.J. Hacker, B. A. (2002). *The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network.* Fort Lauderdale: Parallel and Distributed Processing Symposium.