



UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA



INGENIERÍA DE PROTOCOLOS DE COMUNICACIONES

SERVICIO COMUNICACIONAL CON SEGURIDAD (APL4)

Pablo Fernández Cuadros

Héctor García Miguel

Índice

PARTE I	3
Introducción y objetivos	3
1. Introducción	4
1.1 Objetivos.....	4
PARTE II.....	5
Técnicas aplicables	5
2. Descripción del problema	6
2.1. Técnicas utilizadas.....	6
2.2. Funcionamiento y características del sistema	6
PARTE III	7
Desarrollo software	7
3. Diseño	8
3.1. Descripción de componentes.....	8
4. Implementación	9
5. Validación	12
PARTE IV	15
Conclusiones	15
6. Conclusiones	16
6.1 Conclusiones.....	16
6.2 Organización de la entrega	16

Parte I

INTRODUCCIÓN Y OBJETIVOS

1. INTRODUCCIÓN

El trabajo se encuadra en el ámbito de protocolos de comunicación entre entidades, orientados a la compartición de información de una manera segura. Sobre el desarrollo del proyecto se puede destacar el uso de las herramientas vista en clase como son Microsoft Visual Studio y la API de Windows sockets. Concretando en el sistema, es un servicio conversacional en la que dos usuarios compartirán información de la manera privada, con interfaz gráfica un intérprete de comandos.

1.1 Objetivos

El objetivo principal de este proyecto consiste conectar dos entidades para realizar una comunicación segura compartiendo una clave. Dicha comunicación se realizará utilizando las llamadas de *Windows sockets* vistas en la asignatura. Para que dicha comunicación sea segura se utiliza el algoritmo codificador *blowfish* diseñado por Bruce Schneier. El usuario interactuará con el sistema mediante un intérprete de comando conociendo tanto la dirección IP del usuario opuesto como los puertos de escritura y lectura. Obviamente las dos entidades introducirán una clave compartida para asegurar dicha comunicación.

- **Análisis del problema.** Se realiza un estudio de la problemática que se pretende solucionar con el sistema. Se revisan las técnicas de comunicación remota entre entidades mediante la API de *Windows sockets*.
- **Diseño del sistema.** Se pretende diseñar un sistema que cumpla con los requisitos propuestos que se repasarán en el apartado 2 del documento.
- **Implementación.** Se aborda la fase de implementación con el objetivo de realizar un código con las herramientas vistas en clase. Otro objetivo en esta parte del desarrollo, es que el sistema contemple todos los casos como en el que los usuarios envíen gran cantidad de información.
- **Validación.** Para verificar el buen funcionamiento del sistema se desarrollan pruebas específicas que prueban las funcionalidades del sistema de manera unitaria. Para probar la generalidad del sistema se plantean diversos ejemplos para probar la robustez y solidez del sistema.

Parte II

TÉCNICAS APLICABLES

2. DESCRIPCIÓN DEL PROBLEMA

2.1. Técnicas utilizadas

Debido a que se tomó como objetivo del proyecto utilizar las herramientas como la API de Windows sockets, empleamos el entorno de desarrollo Microsoft Visual Studio. En cuanto al lenguaje de programación se consideró adecuado implementar el código en C++ ya que aporta más flexibilidad para extender el sistema en otras funcionalidades extra.

El trabajo se desarrollo en un grupo de dos personas y para ello recurrimos a un repositorio para la compartición del código. Este repositorio es un sistema de control de versiones llamado *Subversion SVN* con el código alojado en *Google Code*.

2.2. Funcionamiento y características del sistema

El funcionamiento del sistema a desarrollar será el siguiente: Un sistema que realice funciones de servicio conversacional, tipo chat, entre dos usuarios con mecanismos de seguridad en el que ambos entidades conocerán una clave simétrica.

Estas dos entidades tendrán un comportamiento simultáneamente como cliente y servidor en el que cada uno abre dos sockets UDP. Uno para enviar y otro para la recepción de datos los cuales deberán ser consecutivos. Por cada envío que realice cualquiera de las dos entidades, el opuesto responderá con un asentimiento por encima de UDP. Para aportar ese mecanismo de seguridad, se utilizarán los métodos del algoritmo blowfish, el cual las entidades compartirán clave. Para ello se importará la biblioteca blowfish.lib.

En cuanto al protocolo de comunicaciones será UDP, con cada uno de los dos servidores, un servidor de tipo concurrente mediante la creación de hilos en cada entidad. Para utilizar las llamadas a la API de Windows sockets se utilizará la biblioteca ws2_32.lib

Como ya se ha comentado, todo el sistema ha sido implementado en el lenguaje de programación C++ debido nos ofrecía más flexibilidad de extender el proyecto a otras funcionalidades ya que C++ puede utilizar clases.

Parte III

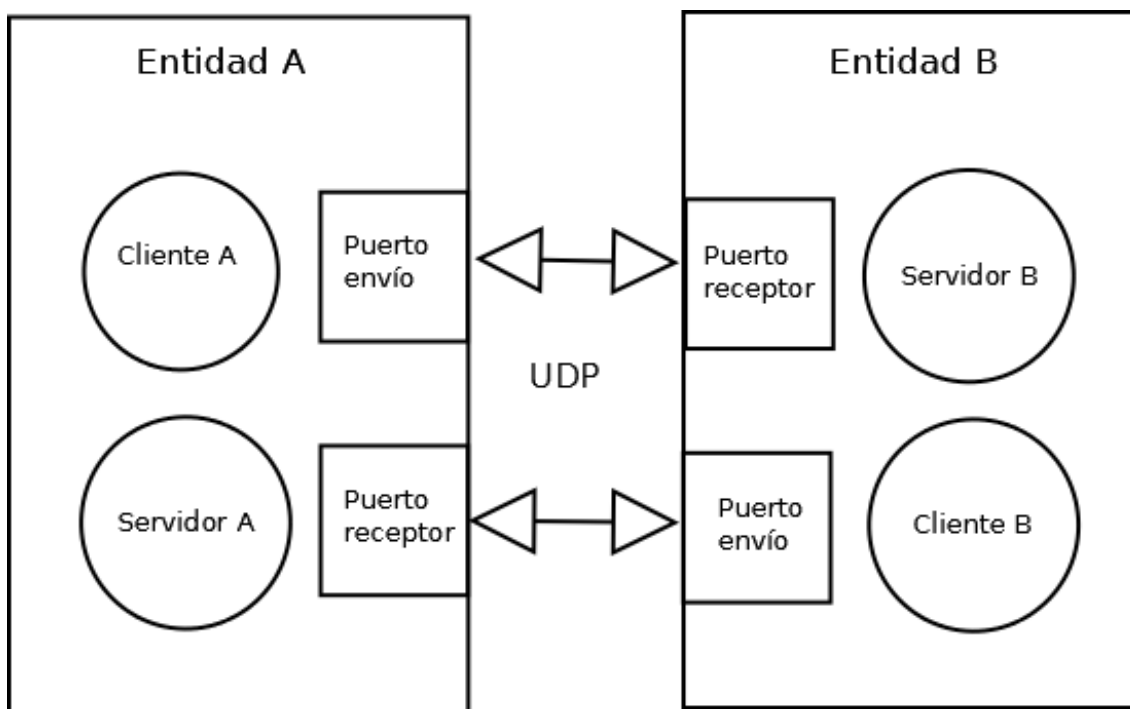
DESARROLLO SOFTWARE

3. DISEÑO

En este apartado se muestra una descripción de cómo resulta la estructura del sistema y más adelante una visión más concreta de la utilización de las herramientas ofrecidas por Windows sockets para la creación de canales de comunicación.

3.1. Descripción de componentes

Como ya se ha destacado en la descripción del problema, el sistema tendrá comunicación entre dos entidades, la cual cada una funcionará como servidor y servidor de la otra y viceversa. Para ello tanto cliente como servidor enviará y recibirá de cada puerto específico.



La comunicación entre entidades se realizará mediante datagramas por lo tanto será nivel *UDP*, por lo tanto se deberá tener en cuenta las restricciones que esto producirá. Por lo tanto y debido al planteamiento principal del trabajo, se tendrá en cuenta la recepción correcta de los mensajes asintiendo a la entidad de enviante.

4. IMPLEMENTACIÓN

A continuación se especificará la implementación del sistema en pseudocódigo para comprobar las funcionalidades y ver como se ha desarrollado.

Función principal: Crea los dos sockets, de recepción y de envío, asocia los puntos de acceso a sendos sockets y lanza dos hilos, uno para la recepción y otro para el envío, creando así un servidor concurrente con hilos.

```
// Declarar variables locales de la función main
// Recibir parámetros
// Inicializar contexto blowfish y windows sockets
// Inicializar puntos de acceso

// Crear socket de envío
// Crear punto de acceso local de envío
// Enlazar socket de envío a punto de acceso local de envío

// Crear socket de recepción
// Crear punto de acceso local de recepción
// Enlazar socket de recepción a punto de acceso local de recepción
// Crear punto de acceso remoto de envío

// Crear hilo de envío con función Send
// Crear hilo de recepción con función Receive

// Esperar hilo de envío
// Esperar hilo de recepción
```

Función Send para el hilo de envío: Realiza el envío del mensaje capturado por el interprete de comandos cifrándolo mediante blowfish.

```
// Declarar variables locales de la función Send

// Bucle infinito del hilo
// Obtener el siguiente carácter de la entrada estándar
// Si el carácter es fin de línea
// Si el tamaño del mensaje es mayor que cero
// Extender el buffer de envío
// Llenar el buffer de envío
// Si no existe mensaje por confirmar
// Calcular el tamaño de pad
// Reservar buffer para mensaje cifrado. Tamaño se extiende
// hasta multiplo de 64 bits / 8 bytes
// Rellenar inicio del buffer con el tamaño del mensaje
// Inicializar a 0 el buffer
// Copiar mensaje al buffer
// Calcular el md5 hash del mensaje
// Establecer que existe mensaje por confirmar. Guardando su
// hash en buffer reservado
// Cifrar el mensaje
```

```
// Proporcionar información por la salida estándar
// Declarar variables para bytes enviados y offset de envío
en buffer
// Actualizar el tamaño del mensaje incrementándolo con la
longitud de cabeza y el pad
// Transmitir el buffer por el socket de envío
// Liberar buffers reservados
// Extender el buffer de envío
// Llenar el buffer de envío
```

Función Receive para el hilo de recepción: Hilo con la llamada bloqueante `recvfrom` en el que recibe el mensaje, luego lo descifra para mostrarlo por pantalla

```
// Declarar variables locales de la función Receive

// Bucle infinito del hilo
// Inicializar buffer de recepción del socket a 0
// Recibir mensaje del socket de recepción
// Actualizar la longitud del mensaje recibido
// Aún no se ha recibido la cantidad necesaria
// Extender el buffer de recepción del mensaje
// Llenar el buffer de recepción del mensaje
// Calcular cantidad de bytes tras el delimitador
// Extender buffer de recepción del mensaje hasta el delimitador
// Llenar buffer de recepción del mensaje hasta el delimitador
// Reservar buffer para el mensaje descifrado
// Rellenar buffer para el mensaje descifrado con el mensaje cifrado
// Delimitar buffer para el mensaje descifrado
// Descifrar el mensaje
// No existe mensaje por confirmar
// Calcular el md5 hash del mensaje
// Proporcionar información por la salida estándar
// Reservar buffer para el mensaje cifrado
// Rellenar inicio del buffer con el tamaño del mensaje
// Inicializar a 0 el buffer
// Copiar mensaje al buffer
// Cifrar mensaje
// Declarar variables para bytes enviados y offset de envío en
buffer
// Inicializar el tamaño enviado del mensaje incrementándolo con
la longitud de cabeza
// Transmitir el buffer por el socket de envío
// Extender buffer de recepción del mensaje a la cantidad de
bytes tras el delimitador
// Llenar buffer de recepción del mensaje con bytes tras
delimitador
// Comprobar que el mensaje recibido es el la confirmación de uno
anterior
// Liberar el buffer del hash del mensaje por confirmar
// Proporcionar información por la salida estándar
// Extender buffer de recepción del mensaje a la cantidad de
bytes tras el delimitador
// Llenar buffer de recepción del mensaje con bytes tras
delimitador
// Proporcionar información por la salida estándar
```

```
        // Extender buffer de recepción del mensaje a la cantidad de
        bytes recibidos
        // Llenar buffer de recepción del mensaje con los bytes
        recibidos
// Liberar buffer de recepción del mensaje
// Liberar buffer de la confirmación de mensaje
```

5. VALIDACIÓN

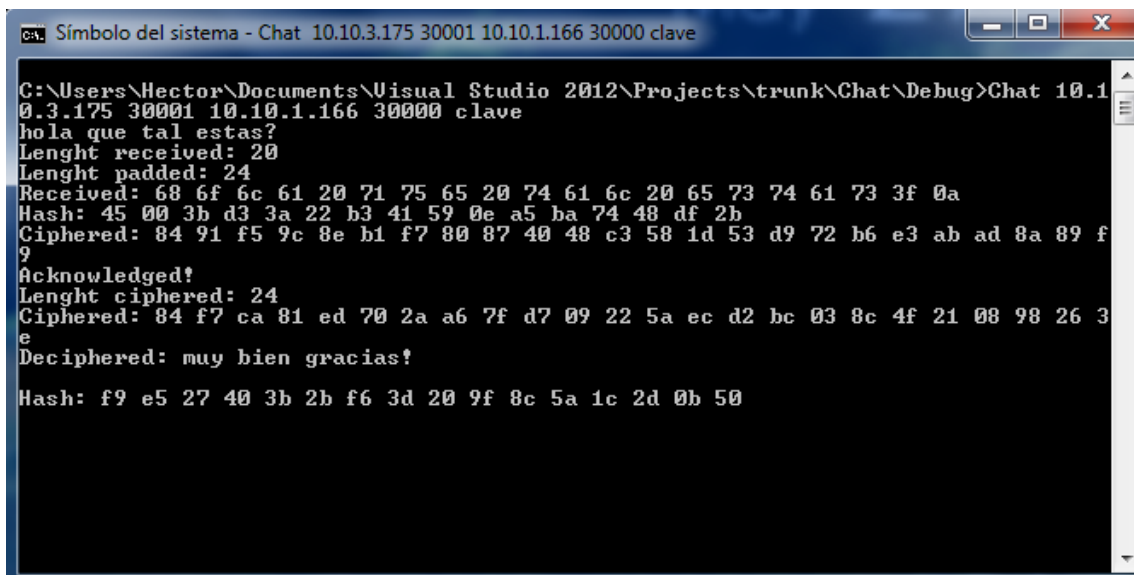
Para la ejecución del *Chat* cada entidad se deberá llamar a la aplicación con los siguientes argumentos:

Nombre de la aplicación: Chat.exe
Dirección de la IP local de la entidad A
Número de puerto de envío
Dirección de IP remota de la entidad B
Número de puerto para la recepción
Clave que será compartida entre las dos entidades

En cuanto a la realización de pruebas se comprobó el funcionamiento tanto en red local como en el exterior para cerciorarse del buen comportamiento del sistema.

Para verificar el funcionamiento del *Chat* observamos el tráfico de la red mediante la herramienta *Wireshark*. De este modo capturando el tráfico nos dio una visión gráfica y más clara del comportamiento.

Como se puede observar se inicia una conversación entre la entidad A 10.10.3.175 y la entidad B 10.10.1.166 en la red de la Facultad.

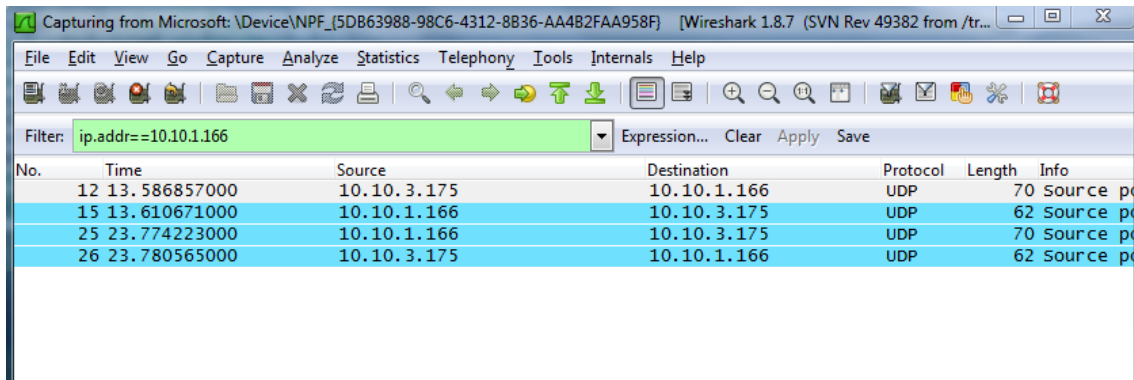


```
C:\Users\Hector\Documents\Visual Studio 2012\Projects\trunk\Chat\Debug>Chat 10.10.3.175 30001 10.10.1.166 30000 clave
hola que tal estas?
Lenght received: 20
Lenght padded: 24
Received: 68 6f 6c 61 20 71 75 65 20 74 61 6c 20 65 73 74 61 73 3f 0a
Hash: 45 00 3b d3 3a 22 b3 41 59 0e a5 ba 74 48 df 2b
Ciphared: 84 91 f5 9c 8e b1 f7 80 87 40 48 c3 58 1d 53 d9 72 b6 e3 ab ad 8a 89 f9
Acknowledged?
Lenght ciphared: 24
Ciphared: 84 f7 ca 81 ed 70 2a a6 7f d7 09 22 5a ec d2 bc 03 8c 4f 21 08 98 26 3e
Deciphared: muy bien gracias!
Hash: f9 e5 27 40 3b 2b f6 3d 20 9f 8c 5a 1c 2d 0b 50
```

En un primer instante la entidad A envía la cadena 'hola que tal estas?'. Inmediatamente la entidad B recibe esta cadena y envía un asentimiento.

Servicio comunicacional con seguridad (APL4)

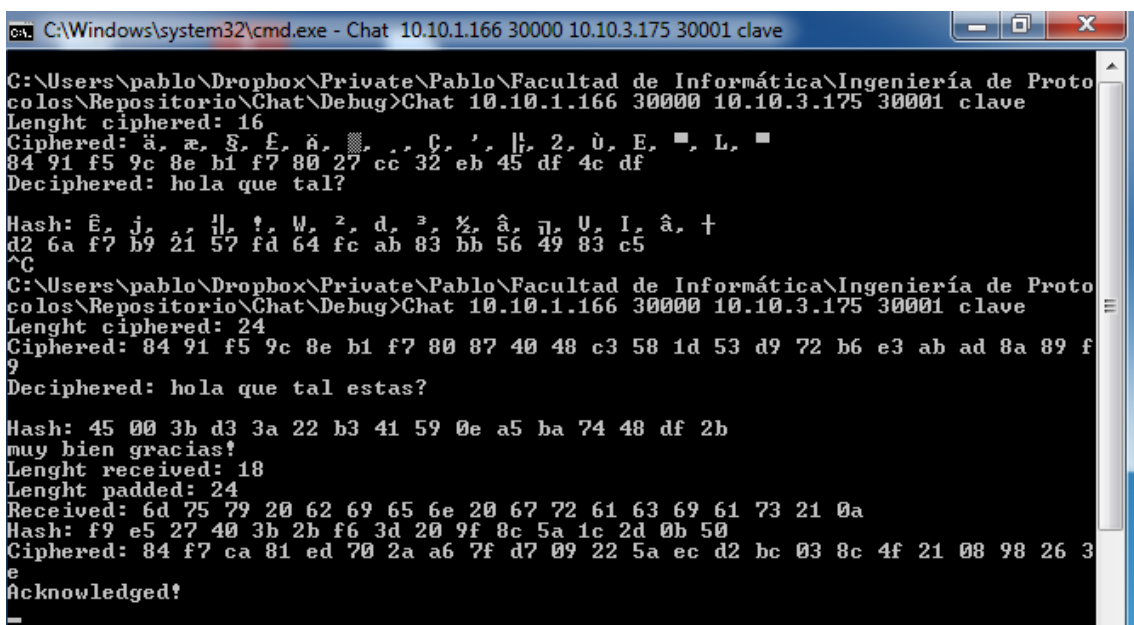
Como se puede ver en la captura de pantalla, capturamos el tráfico y comprobamos el flujo de información entre la entidad A y B en todo momento cifrada y el asentimiento de B a A.



No.	Time	Source	Destination	Protocol	Length	Info
12	13.586857000	10.10.3.175	10.10.1.166	UDP	70	Source port 30001
15	13.610671000	10.10.1.166	10.10.3.175	UDP	62	Source port 30000
25	23.774223000	10.10.1.166	10.10.3.175	UDP	70	Source port 30000
26	23.780565000	10.10.3.175	10.10.1.166	UDP	62	Source port 30001

También se realizó la operación contraria de que la entidad B envíe un mensaje al A 'muy bien gracias!', como se comprueba en la tercera línea y el asentimiento correspondiente de A a B en la cuarta fila.

Para comprobar vemos la ejecución de la entidad contraria:

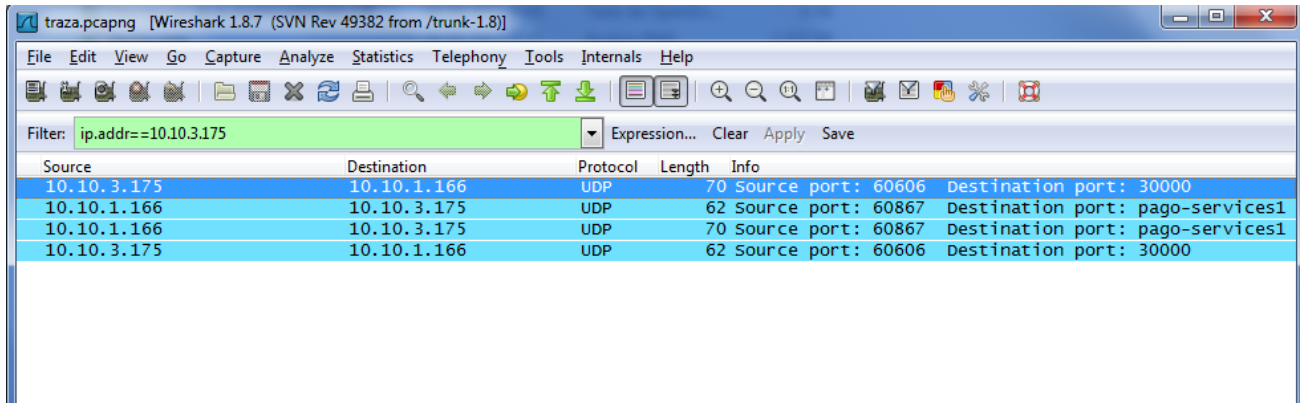


```
C:\Windows\system32\cmd.exe - Chat 10.10.1.166 30000 10.10.3.175 30001 clave
C:\Users\pablo\Dropbox\Private\Pablo\Facultad de Informática\Ingeniería de Protocolos\Repositorio\Chat\Debug>Chat 10.10.1.166 30000 10.10.3.175 30001 clave
Lenght ciphered: 16
Ciphered: ä, æ, §, £, ä,  , Ç, ', ll, 2, ù, E, ■, L, ■
84 91 f5 9c 8e b1 f7 80 27 cc 32 eb 45 df 4c df
Deciphered: hola que tal?

Hash: Ê, j,  , ll, !, W, ², d, ³, ½, â, ñ, U, I, â, +
d2 6a f7 b9 21 57 fd 64 fc ab 83 bb 56 49 83 c5
^C
C:\Users\pablo\Dropbox\Private\Pablo\Facultad de Informática\Ingeniería de Protocolos\Repositorio\Chat\Debug>Chat 10.10.1.166 30000 10.10.3.175 30001 clave
Lenght ciphered: 24
Ciphered: 84 91 f5 9c 8e b1 f7 80 87 40 48 c3 58 1d 53 d9 72 b6 e3 ab ad 8a 89 f
9
Deciphered: hola que tal estas?

Hash: 45 00 3b d3 3a 22 b3 41 59 0e a5 ba 74 48 df 2b
muy bien gracias!
Lenght received: 18
Lenght padded: 24
Received: 6d 75 79 20 62 69 65 6e 20 67 72 61 63 69 61 73 21 0a
Hash: f9 e5 27 40 3b 2b f6 3d 20 9f 8c 5a 1c 2d 0b 50
Ciphered: 84 f7 ca 81 ed 70 2a a6 7f d7 09 22 5a ec d2 bc 03 8c 4f 21 08 98 26 3
e
Acknowledged!
```

Y la captura de tráfico de la entidad contraria:



traza.pcapng [Wireshark 1.8.7 (SVN Rev 49382 from /trunk-1.8)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: ip.addr==10.10.3.175 Expression... Clear Apply Save

Source	Destination	Protocol	Length	Info
10.10.3.175	10.10.1.166	UDP	70	Source port: 60606 Destination port: 30000
10.10.1.166	10.10.3.175	UDP	62	Source port: 60867 Destination port: pago-services1
10.10.1.166	10.10.3.175	UDP	70	Source port: 60867 Destination port: pago-services1
10.10.3.175	10.10.1.166	UDP	62	Source port: 60606 Destination port: 30000

Parte IV

CONCLUSIONES

6. CONCLUSIONES

6.1 Conclusiones

Después de la realización de este proyecto se puede asegurar que se ha trabajado en el ámbito y herramientas explicadas en clase utilizando mecanismos de comunicación, en este trabajo especialmente *Windows sockets* aunque como ya se ha podido comprobar, es fácilmente compatible para otros sistemas.

Para futuros proyectos del mismo ámbito, seguro que se afrontará con una mayor facilidad y soltura a la hora de implementar los mecanismos de seguridad.

6.2 Organización de la entrega

Aparte de la realización de esta documentación se ha incluido en un CD el sistema, organizado en los siguientes elementos:

- Carpeta 'Proyecto en vs2012': contiene el proyecto desarrollado en *Visual Studio 2012*.
- Carpeta 'Código y librerías': contiene el archivo con el código del *Chat.cpp* y las librerías para *blowfish* y *md5*.
- Carpeta 'Ejecutable': Contiene el solo ejecutable del Chat.

