

**Instituto Politecnico Nacional**

**ESCOM “ESCUELA SUPERIOR DE CÓMPUTO”**

*REDES DE COMPUTADORAS*

*PRÁCTICA 2: Algoritmo CheckSum*

PROFE: Axel Moreno Cervantes

ALUMMNOS: Rojas Alvarado Luis Enrique

Miranda Sandoval Mario Alberto

GRUPO: 2CM10

**INTRODUCCIÓN**

El algoritmo CheckSumes un algoritmo para detección y corrección de errores en un transmisor que calcula un valor numérico según el número de bits establecidos o no configurados en un mensaje y lo envía junto con cada trama de mensaje. En el extremo del receptor, la misma función de suma de comprobación (fórmula) se aplica al cuadro de mensaje para recuperar el valor numérico. Si el valor de suma de comprobación recibido coincide con el valor enviado, la transmisión se considera exitosa y sin errores.

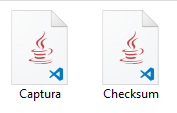
Un CheckSum no coincidente muestra que no se ha transmitido todo el mensaje. TCP / IP y User Datagram Protocol (UDP) proporcionan un recuento de suma de comprobación como uno de sus servicios.

Los algoritmos de CheckSum eficientes producen resultados diferentes con grandes probabilidades si los mensajes están dañados. Los bits de paridad y los dígitos de verificación son casos de suma de comprobación especiales adecuados para pequeños bloques de datos. Ciertos códigos de corrección de errores basados ​​en sumas de comprobación son incluso capaces de recuperar los datos originales.

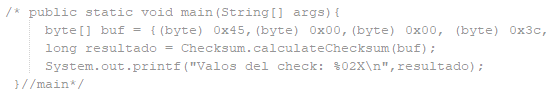
**DESARROLLO**

La práctica se desarrolló en el mismo archivo de Captura.java sólo que esta vez se agregó una clase que calcula el CheckSum, para esto debíamos identificar que las tramas fueran de primero de Ethernet, identificando su tipo mayor a 1500 y posteriormente que sea una IP con si tipo igual 2048.

Para el desarrollo de esta práctica se usaron los siguientes archivos, el Captura.java que se nos habia dado la práctica pasada (antes de la modificación), y este nuevo código el Checksum.java (dado para esta práctica), el cual contiene el algoritmo del checksum.



Lo primero que se hizo fue comentar el método main del Checksum.java para poder hacer uso de sus métodos y evitar la colisión de clases principales.



Eso fue todo por parte del Checksum, en el Captura.java se volvió a la versión original (sin modificar) y se tomó en cuenta lo siguiente:

Para enviar la trama correcta al Checksum, después de los 12 bytes de la dirección origen y destino nos encontramos con un byte de encabezado, que es 45, donde el 4 es la versión y el 5 es la longitud de la cabecera, al hacer la multiplicación de la longitud de la cabecera obtendremos el número de bytes que tendremos que mandar al método ya antes mencionado.

Para diferenciar el protocolo para la capa de transporte el byte numero 10 a partir del byte 45 (encabezado), nos dará el tipo de protocolo, los dos bytes siguientes pertenecen a los bytes de checksum, para verificar si es correcto se deben mandar todos los datos ordenados (en palabras de 16 bits) y debe de dar 0000 si no es así es incorrecto, para este caso los bytes de checksum se deben mandar como 0000 y al obtener el resultado de ese checksum se debe complementar a 1.

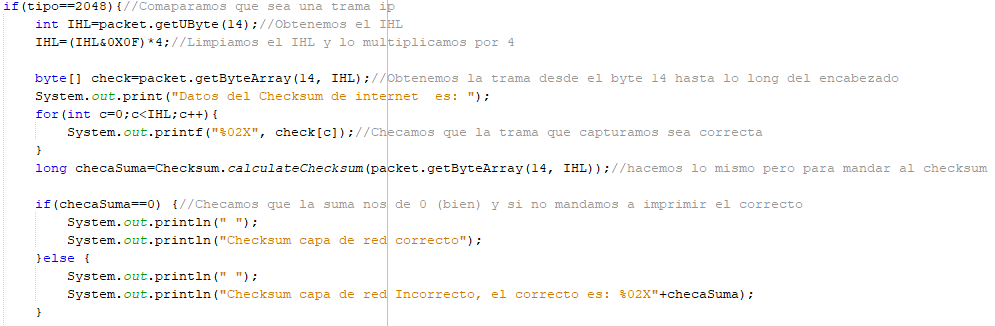
Y para comprobarlo solo se debe sumar el valor obtenido y su complemento, ese resultado al complementarlo a 1 debe dar 0000 y eso indicará que es el checksum correcto.

La modificación al código Captura.java fue la siguiente:

Checamos que sea una trama IP.



Comparamos que sea la trama IP el cual debe ser 2048 en decimal y preparamos las tramas para la capa de red.



Para la capa de transporte, el checkum se calcula a partir de la suma de un pseudo encabezado y el PDU de la capa de transporte.

El pseudo encabezado se conforma de la siguiente manera:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IP  Origen | + | IP  Destino | + | Cero’s | + | Protocolo | + | Longitud (Segmento TCP) |
| (4 bytes) |  | (4 bytes) |  | (1 byte) |  | (1 byte) |  | (2 bytes) |

La longitud de la cabecera es 5, y el byte protocolo mostrado es 06 por lo que indica que se trata de TCP (recordar que si el byte es 17 corresponde a UDP)

Finalmente, el pseudo encabezado quedaría de la siguiente manera:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IP  Origen |  | IP  Destino |  | Cero’s |  | Protocolo |  | Longitud (Segmento TCP) |
| c192c4ec | + | 93530797 | + | 00 | + | 06 | + | 05 |

El PDU de transporte se forma de la siguiente manera:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Longitud | - | ( IHL | \* | Versión) |
| (2 bytes) |  | (4 bits) |  | (4 bits) |

Donde:

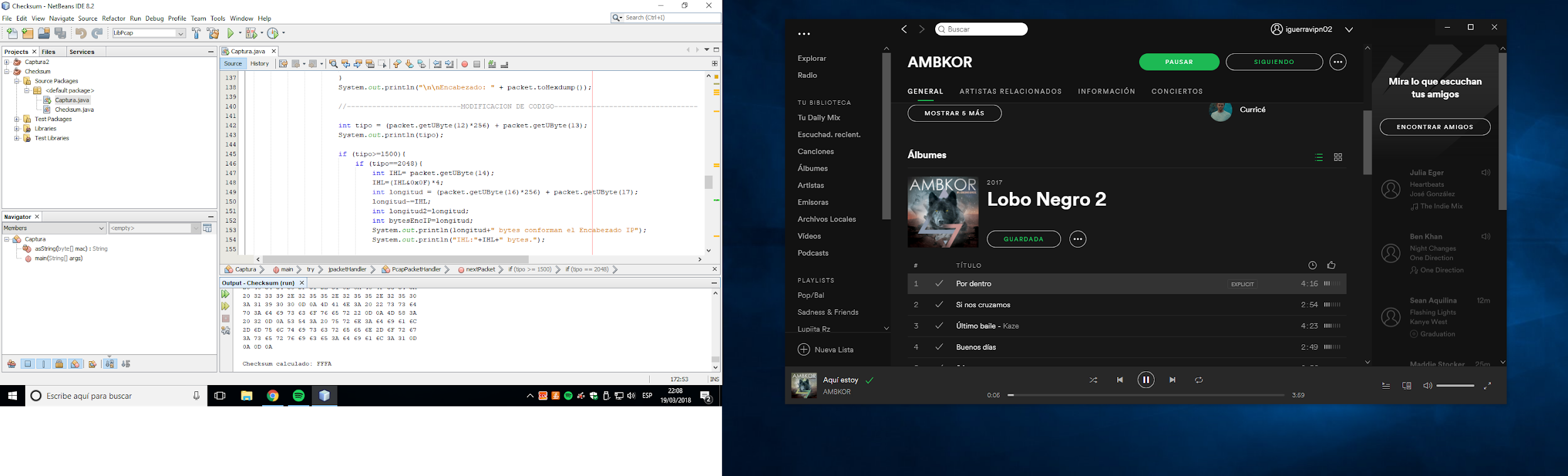
El 4 es la versión, el 5 representa el IHL o longitud de cabecera. Al multiplicarlos obtenemos como resultado 20.

Los bytes 00 2c representan la longitud total del paquete, que es equivalente a 44 en decimal.

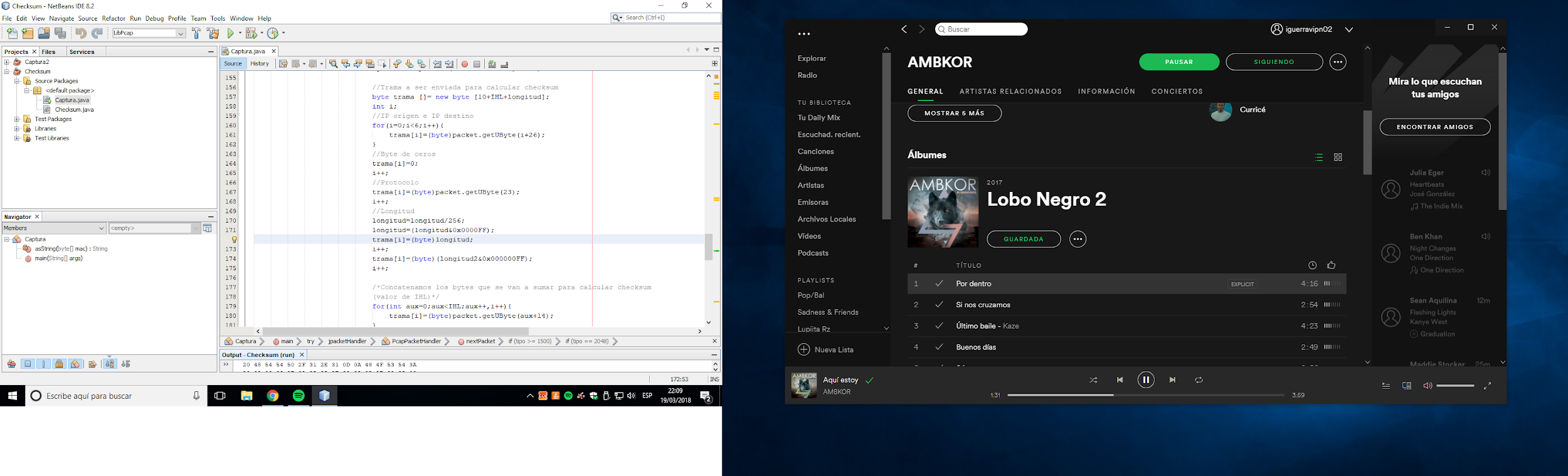
Por lo tanto, el PDU de transporte queda de la siguiente manera:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Longitud |  | ( IHL |  | Versión) |
| 002c | - | (5 | \* | 4) |

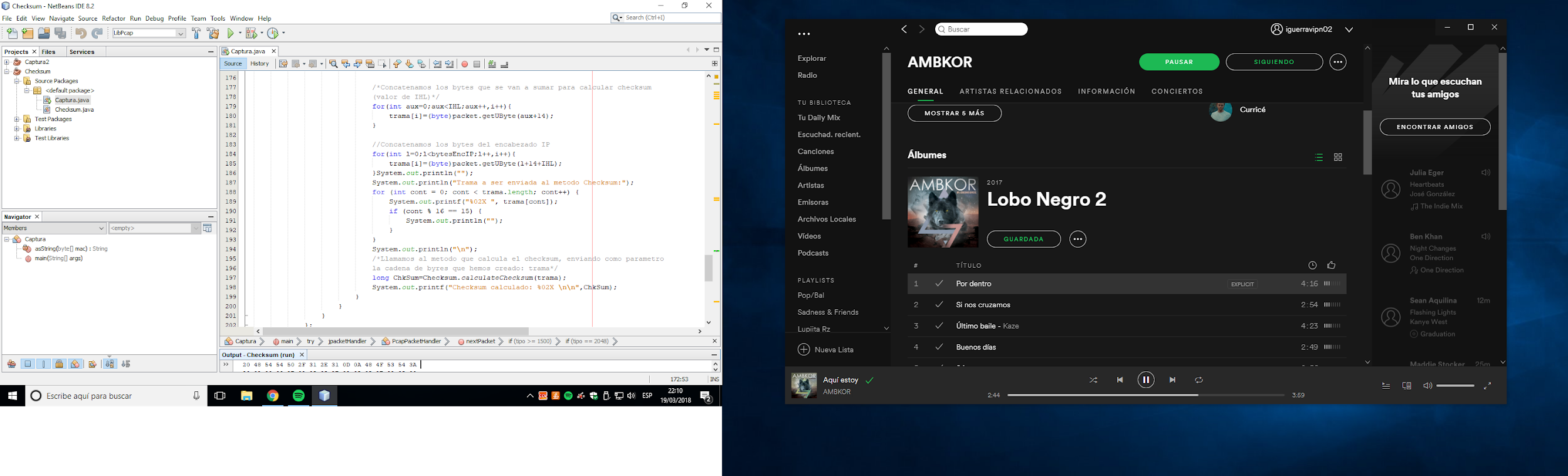
Finalmente, al sumar el pseudo encabezado y el PDU obtenemos el checksum.



Se crea el encabezado IP.



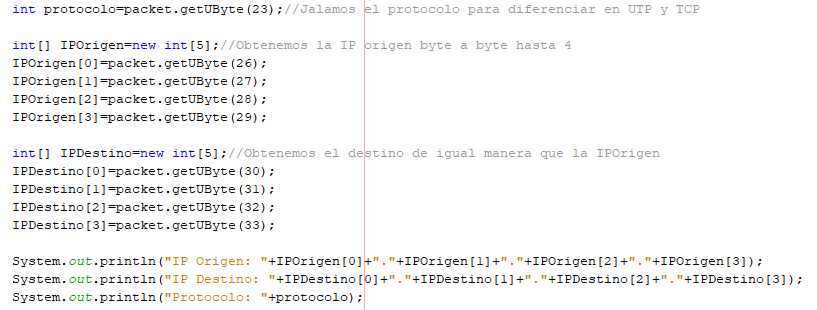
Después se crea la nueva trama y se va llenando con la información.

La Ip origen y destino, los bytes de ceros, el protocolo y su longitud.

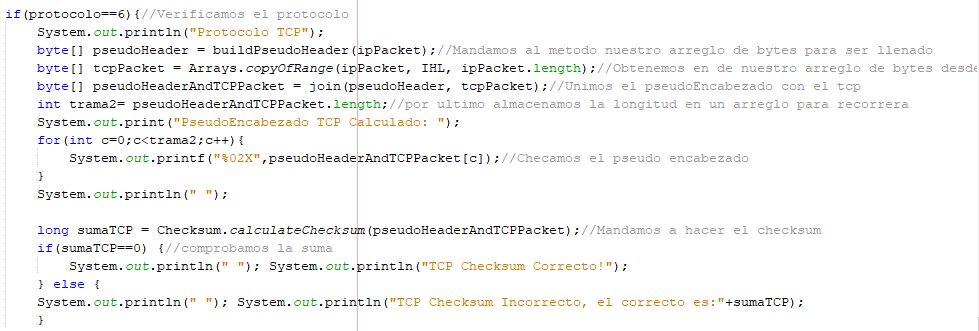
Se concatena con la trama anterior más el encabezado IP, y es la trama que se enviará al método checksum.

Y el código implementado fue el siguiente:

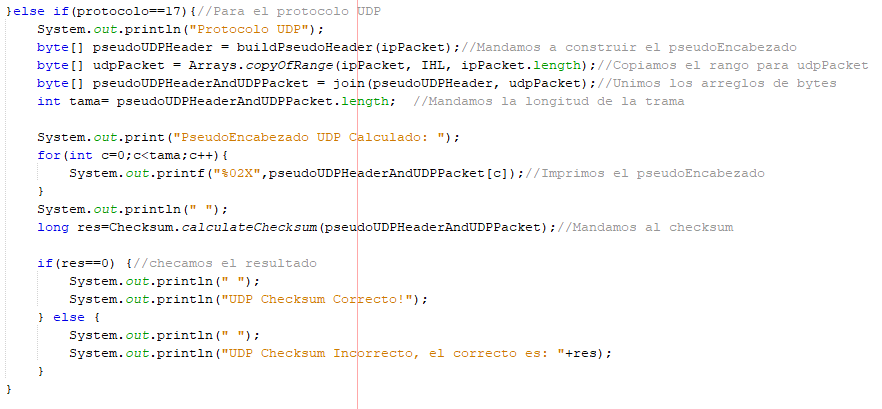
Para la capa de transporte.



Para el protocolo TCP.

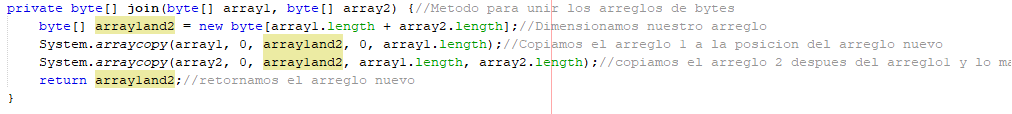


Para el protocolo UDP.

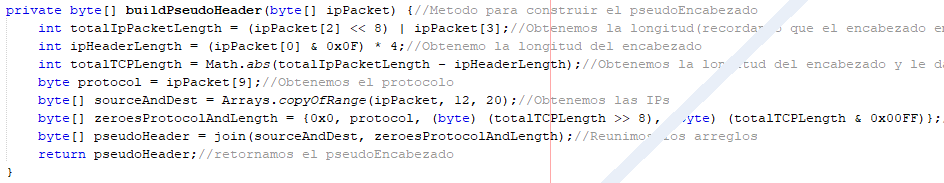


Métodos usados para facilitar el manejo de los encabezados.

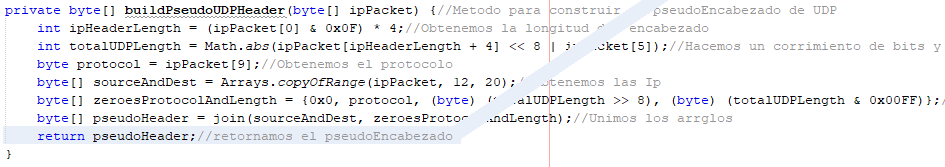
Unión de los arreglos de bytes.

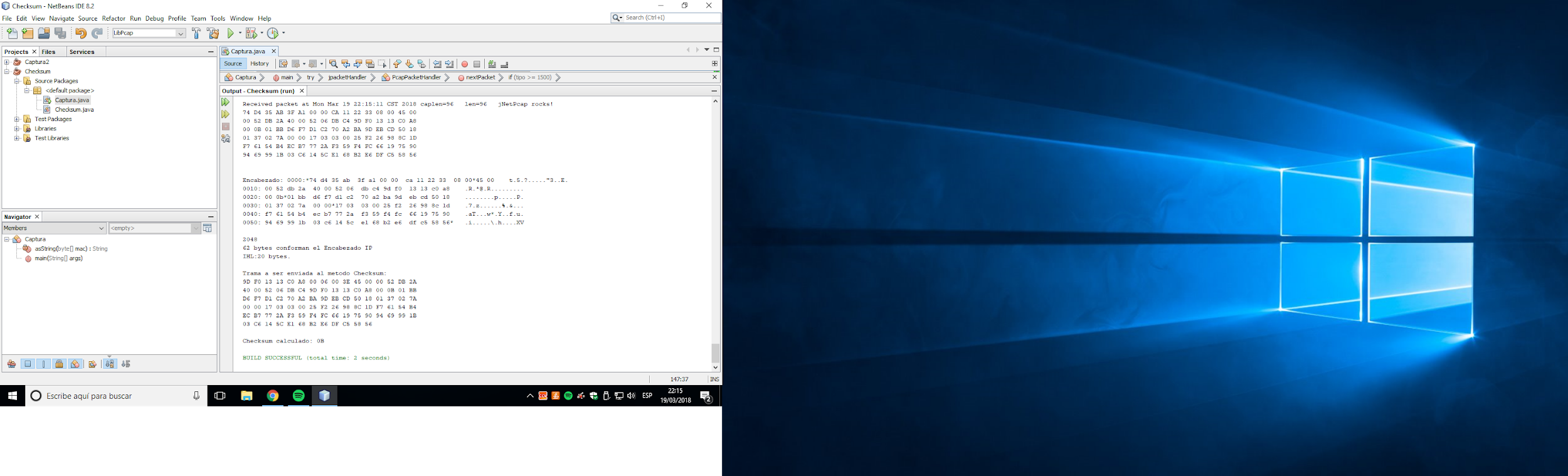


Para el psuedoEncabezado de TCP.



Para el pseudoEncabezado de UDP.



**PRUEBAS**

**POSIBLES MEJORAS**

Antes de mostrar esta práctica al profesor, se tenía un código totalmente distinto a este, el motivo fue que, de manera inicial intente desmadejar la trama byte a byte y guardarla en distintas variables para después en un arreglo de bytes ordenar los datos y mandar al método checksum la trama, por alguna razón al comparar la trama ordenada con la trama capturada me daba lo mismo y el checksum salía diferente, entonces opte por mandar al checksum, el arreglo de bytes desde un método de la propia librería JNetPcap, de ahí procedí a querer hacer lo mismo con los protocolos de la capa de transporte pero ante mis dudas, estuve investigando y fui encontrando en varias páginas de internet ideas o métodos específicos usando JNetPcap, entonces cambie mis ideas de codificación y se implementaron los métodos para unir los arreglos de bytes y el cálculo de pseudoEncabezados que facilitó la generación de estos y la comprensión a su vez.

**CONCLUSIONES**

* Rojas Alvarado Luis Enrique

Gracias a ésta práctica pudimos observar como mediante la programación se nota claramente cómo está la trama que se envían a través de la tarjeta de red. Me gustó esta práctica porque prácticamente lo que hicimos es programar el algoritmo para detección de error, de la misma manera que lo analizamos y checamos varios puntos del ChecSum y su comprobación.

* Miranda Sandoval Mario A.

En general esta práctica fue importante para el entendimiento de un algoritmo de errores tan importante como es el checksum, al principio se veía algo compleja, pero después de preguntarle al profe, me di cuenta que era fácil, como tal al inicio tenía un error ya que estaba mandando la trama dos veces, pero después de corregir eso y pasar por lo ya antes mencionado en las mejoras al código, la práctica fue realizada con éxito.

Con esta práctica se logra el entendimiento de las tramas IP, el cómo se generan y como tomar los bytes para encontrar el checksum y saber si se tienen errores en los envíos o no, esto en futuro nos servirá para aplicarlo en diversas prácticas de redes.

**BIBLIOGRAFÍA**

1. <http://148.204.58.221/axel/redesnp/sniffer/>
2. <https://www.techopedia.com/definition/1792/checksum>