

CS7344HW5

Haojing Zhai(hzhai@mail.smu.edu), Yayu Mo(yayum@mail.smu.edu), Liping Tang(lipingt@mail.smu.edu)

Theory questions

1. Why does UDP exist? Would it not have been enough to just let user processes send raw IP packets?

Using UDP ensures that a segment is correctly delivered to the intended application because UDP includes source and destination ports, unlike raw IP packets which lack these ports. Consequently, a segment cannot be delivered to a specific application if sent as a raw IP packet.

2. In both parts of Fig. 6-6 Links to an external site. in your textbook, there is a comment that the value of SERVER_PORT must be the same in both client and server. Why is this so important?

Since the server is listening on a specific port, the client must specify this port; otherwise, the server won't recognize that the client is attempting to connect.

3. For each of the following applications, tell whether it would be (1) possible and (2) better to use a PHP script or JavaScript, and why:

- **Displaying a calendar for any requested month since September 1752**
 - JavaScript is probably more suitable for this task because it can dynamically create and display the calendar on the client side, eliminating the need for server-side processing.
- **Displaying the schedule of flights from Amsterdam to New York.**
 - PHP is well-suited for server-side tasks such as accessing and manipulating database information. Given that flight schedules are typically stored in a database, PHP is appropriate for retrieving and displaying this data.
- **Graphing a polynomial from user-supplied coefficients.**
 - JavaScript is better suited for this task because it allows for instant feedback and dynamic content right in the user's browser. It can manage user input, process it, and update the graph instantly, all without needing extra server requests.

4. You request a Web page from a server. The server's reply includes an Expires header, whose value is set to one day in the future. After five minutes, you request the same page from the same server. Can the server send you a newer version of the page? Explain why or why not?

Yes, even if the "Expires" header is set to expire one day in the future, the server can still deliver a newer version of the page. The "Expires" header primarily governs client-side caching rather than dictating how content is delivered from the server. Depending on its configuration and the details of the client request, the server can supply an updated version of the page.

Lab Question

5. Implement a client-server communication system using socket programming using Java programming language. The client should send a text message to the server, which will respond accordingly. Additionally, integrate one of the following features into your application: error detection, error correction, or data encryption.

- Explanation
 - We integrate data encryption in our code. Specifically, we apply the DES/ECB/PKCS5Padding in Java Cipher package and Base64 package to implement data encryption in our code.
 - We are using the specific jdk-1.8 version due to the reason that it will be much easier to import the net and encryption packages.
- Server Code

```
/**
 * using jdk 1.8
 */
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Base64;
import java.util.Random;

public class Server {
    public static void main(String[] args) {
        try {
            // Create server and assigning port
            ServerSocket serverSocket = new ServerSocket(12321);

            System.out.println("Server Starting...");

            // monitoring request
            Socket clientSocket = serverSocket.accept();
```

```

        System.out.println("Connected");
        System.out.println("-----");

        // readstream and writestream obj
        BufferedReader input = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        PrintWriter output = new PrintWriter(clientSocket.getOutputStream(), true);

        String messageReceived;

        while (true) {
            // read input from client
            messageReceived = input.readLine();

            if (messageReceived.equalsIgnoreCase("exit")) {
                System.out.println("Disconnected");
                // if get "exit" break
                break;
            }

            // decrypt message
            String keyReceived = messageReceived.substring(0,8);
            String messageText = messageReceived.substring(8);
            String decryptedMessageReceived = decrypt(messageText, keyReceived);
            // print all info received
            System.out.println("Received Message from Client:");
            System.out.println("Key Received: " + keyReceived);
            System.out.println("Decrypted Message Received: " + decryptedMessageReceived);
            System.out.println("Total Dataframe Received: " + messageReceived);
            System.out.println("-----");

            // encrypt response to client
            // encrypt sent message
            String messageResponse = "Message Received: " + decryptedMessageReceived;
            String keyRes = generateRandomKey(8);
            String encryptedMessage = encrypt(messageResponse, keyRes);
            String dataframeResponse = keyRes + encryptedMessage;
            // send response to client
            output.println(dataframeResponse);
            // print all info response
            System.out.println("Message Response: ");
            System.out.println("Key Response: " + keyRes);
            System.out.println("Encrypted Message Response: " + encryptedMessage);
            System.out.println("Total Dataframe Response: " + dataframeResponse);
            System.out.println("-----");
        }

        // close connection
        clientSocket.close();
        serverSocket.close();

    } catch (IOException e){
        e.printStackTrace();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

// encryption
public static String encrypt(String plainText, String key) throws Exception {
    SecretKey secretKey = new SecretKeySpec(key.getBytes(), "DES");

    // initialize with trans in Encryption mode
    Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);

    byte[] encryptedData = cipher.doFinal(plainText.getBytes());

    return Base64.getEncoder().encodeToString(encryptedData);
}

// decryption
public static String decrypt(String encryptedText, String key) throws Exception {
    SecretKey secretKey = new SecretKeySpec(key.getBytes(), "DES");

    Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, secretKey);

    byte[] encryptedBytes = Base64.getDecoder().decode(encryptedText);

```

```

        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        return new String(decryptedBytes);
    }

    // generate random key
    public static String generateRandomKey(int length){
        char[] charset = "abcdefghijklmnopqrstuvwxyz0123456789".toCharArray();
        Random random = new Random();
        StringBuilder str = new StringBuilder(length);
        for (int i = 0; i < length; i++) {
            char c = charset[random.nextInt(charset.length)];
            str.append(c);
        }
        return str.toString();
    }
}

```

- Client Code

```

    ◦ import com.sun.media.sound.SF2Instrument;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Base64;
import java.util.Random;

public class Client {
    public static void main(String[] args) {
        try {
            // create socket instance
            Socket socket = new Socket("127.0.0.1", 12321);

            // readstream and writestream obj
            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

            // get user input
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            String messageSent;

            while (true) {
                System.out.println("Please input the message (input 'exit' to quit):");
                // read input from console
                messageSent = reader.readLine();

                if (messageSent.equalsIgnoreCase("exit")) {
                    // if get "exit" break
                    output.println("exit");
                    break;
                }
                System.out.println("-----");

                // encrypt sent message
                String keySent = generateRandomKey(8);
                String encryptedMessage = encrypt(messageSent, keySent);
                String dataframeSent = keySent + encryptedMessage;

                // send message to server
                output.println(dataframeSent);
                // print all info sent
                System.out.println("Message Sent: ");
                System.out.println("Key Sent: " + keySent);
                System.out.println("Encrypted Message Sent: " + encryptedMessage);
                System.out.println("Total Dataframe Sent: " + dataframeSent);
                System.out.println("-----");

                // received response
                String res = input.readLine();

                // decrypt response
                String keyRes = res.substring(0,8);
                String MessageRes = res.substring(8);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

        String decryptedMessageRes = decrypt(MessageRes, keyRes);
        // print response
        System.out.println("Server Response: ");
        System.out.println("Key Response: " + keyRes);
        System.out.println("Decrypted Message Response: " + decryptedMessageRes);
        System.out.println("Total Dataframe Response: " + res);
        System.out.println("-----");
    }

    // close connection
    socket.close();

} catch (IOException e){
    e.printStackTrace();
} catch (Exception e) {
    throw new RuntimeException(e);
}
}

// encryption
public static String encrypt(String plainText, String key) throws Exception {
    SecretKey secretKey = new SecretKeySpec(key.getBytes(), "DES");

    // initialize with trans in Encryption mode
    Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);

    byte[] encryptedData = cipher.doFinal(plainText.getBytes());

    return Base64.getEncoder().encodeToString(encryptedData);
}

// decryption
public static String decrypt(String encryptedText, String key) throws Exception {
    SecretKey secretKey = new SecretKeySpec(key.getBytes(), "DES");

    Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, secretKey);

    byte[] encryptedBytes = Base64.getDecoder().decode(encryptedText);

    byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
    return new String(decryptedBytes);
}

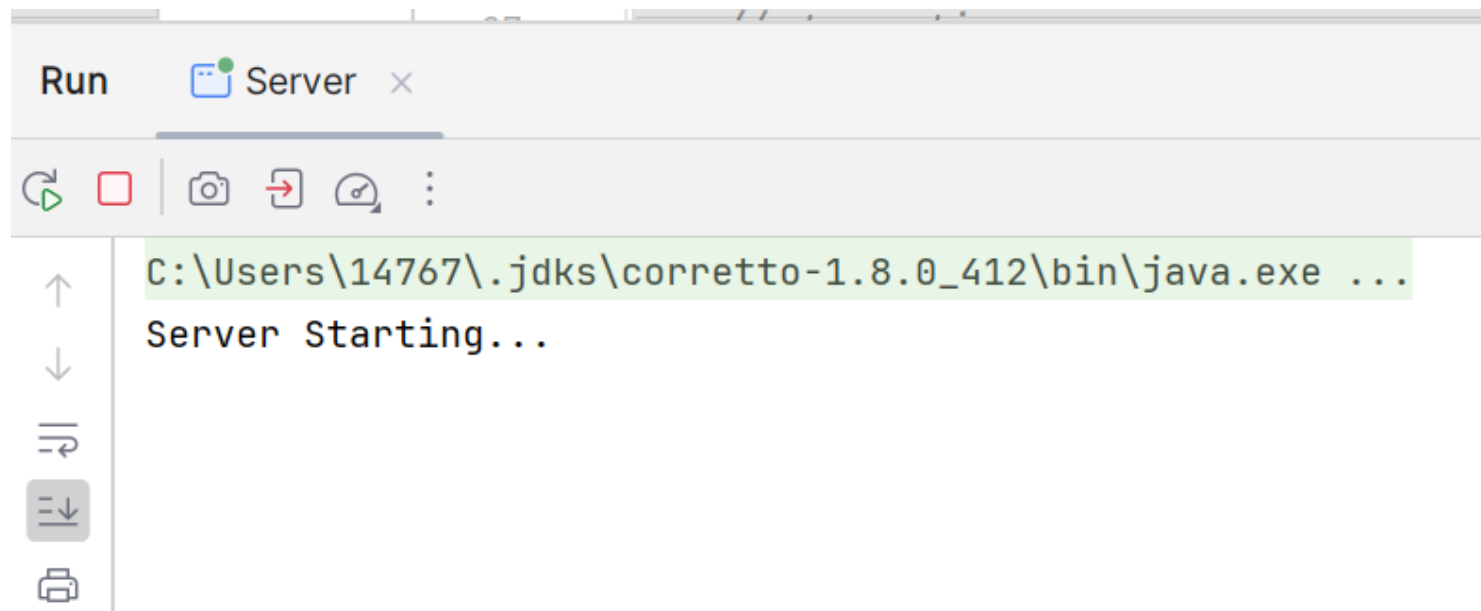
// generate random key
public static String generateRandomKey(int length){
    char[] charset = "abcdefghijklmnopqrstuvwxyz0123456789".toCharArray();
    Random random = new Random();
    StringBuilder str = new StringBuilder(length);
    for (int i = 0; i < length; i++) {
        char c = charset[random.nextInt(charset.length)];
        str.append(c);
    }
    return str.toString();
}
}

```

- Running Result


- For the first step, we start the server on 127.0.0.1:12321, the "Server Starting" text will be displayed.

◦



- Then we start the client on the same port, the server displays the connection result, it shows "Connected".

Server xClient x




C:\Users\14767\jdk\corretto-1.8.0_412\bin\java.exe ...
Server Starting...
Connected

|

- The client shows the required information clients have to input


Server xClient x



C:\Users\14767\jdk\corretto-1.8.0_412\bin\java.exe ...
Please input the message (input 'exit' to quit):
|

- We input "Hello Server!", the data will be encrypted with a randomly generated length 8 key and combined with the key, and then send the entire dataframe to server. After the server receives it, it will respond "Message Received: Hello Server!", and encrypt the same way as client does, then send back to the client.

Server xClient x



C:\Users\14767\jdk\corretto-1.8.0_412\bin\java.exe ...
Server Starting...
Connected

Received Message from Client:
Key Received: 8m91lub0
Decrypted Message Received: Hello Server!
Total Dataframe Received: 8m91lub04Zmbz0pFY+0/jKQIk2Y2/w==

Message Response:
Key Response: myq075tu
Encrypted Message Response: emmKQcm+MiYbmos3qeLbwqQ4L5VSwH5/qTFqdGnV4m4=
Total Dataframe Response: myq075tuemmKQcm+MiYbmos3qeLbwqQ4L5VSwH5/qTFqdGnV4m4=

```
Server x Client x
[Icons]
C:\Users\14767\jdk\corretto-1.8.0_412\bin\java.exe ...
Please input the message (input 'exit' to quit):
Hello Server!
-----
Message Sent:
Key Sent: 8m91lub0
Encrypted Message Sent: 4Zmbz0pFY+0/jKQIk2Y2/w==
Total Dataframe Sent: 8m91lub04Zmbz0pFY+0/jKQIk2Y2/w==
-----
Server Response:
Key Response: myq075tu
Decrypted Message Response: Message Received: Hello Server!
Total Dataframe Response: myq075tueemmKQcm+MiYbm0s3qeLbwqQ4L5VSwH5/qTFqdGnV4m4=
-----
Please input the message (input 'exit' to quit):
|
```

6. Demonstrate the functionality of your application in question 5 by exchanging multiple messages between client and server, showcasing the integrated advanced feature's impact on performance. Include a detailed performance evaluation report with findings and insights.

- The Data Encryption Standard (DES) algorithm is used for encryption and decryption on both the server and the client. The DES is a symmetric key block cipher algorithm used for encryption and decryption of electronic data. It operates on 64-bit blocks of plaintext, using a 56-bit key. DES applies a series of operations, including substitution, permutation, and XOR operations, repeated for a number of rounds, to transform plaintext into ciphertext and vice versa.
- At its core, DES comprises the following key components:
 1. Initial Permutation (IP): Rearranges the bits of the plaintext according to a predefined permutation table.
 2. Key Generation: Produces 16 subkeys, each derived from the original 56-bit key, through a combination of permutation and rotation operations.
 3. Feistel Network: Utilizes a Feistel structure, where the plaintext block is divided into two halves, and each half undergoes a series of operations involving subkey mixing, substitution (S-boxes), permutation (P-box), and XOR operations.
 4. Final Permutation (FP): Reorders the bits of the output from the last round to generate the ciphertext.
- Despite its widespread adoption in the past, DES is now considered insecure due to its small key size and vulnerability to brute-force attacks. As a result, it has been largely replaced by more secure encryption algorithms such as AES (Advanced Encryption Standard).
- In order to evaluate the performance of C-S model with DES encryption algorithm, we measure the time of encryption and decryption, and compare the time of unencrypted C-S model to evaluate.
- For the transmission of the message "Hello, world!", the unencrypted elapsed time is 4,128,125 ns, and the total time about encryption and decryption is 101,087,208 ns.

```
Please input the message (input 'exit' to quit):
Hello world!
-----
Server Response: Message Received: Hello world!
Elapsed Time (nanoseconds): 4128125

Please input the message (input 'exit' to quit):
Hello world!
-----
Message Sent:
Key Sent: mytd1m5k
Encrypted Message Sent: a7tE00SE1mKzkwb2o7VUTw==
Total Dataframe Sent: mytd1m5ka7tE00SE1mKzkwb2o7VUTw==
Encryption Time: 100656875 nanoseconds
-----
Server Response:
Key Response: vzmXilin
Decrypted Message Response: Message Received: Hello world!
Total Dataframe Response: vzmXilinTsLNDpWuqWIHGe7BEBKtrd4yNRe7QFsFmrqGV0mMBQ4=
Decryption Time: 430333 nanoseconds
```

- For the transmission of the message with special characters "@#encrypt@#", the unencrypted elapsed time is 505,041 ns, and the total time about encryption and decryption is 902,459 ns.

Please input the message (input 'exit' to quit):

@#encrypt@#

Server Response: Message Received: @#encrypt@#

Elapsed Time (nanoseconds): 505041

Please input the message (input 'exit' to quit):

@#encrypt@#

Message Sent:

Key Sent: wzz7qk90

Encrypted Message Sent: TVWhMjYuoK2KSskRyXVcqQ==

Total Dataframe Sent: wzz7qk90TVWhMjYuoK2KSskRyXVcqQ==

Encryption Time: 504500 nanoseconds

Server Response:

Key Response: qo4e3thi

Decrypted Message Response: Message Received: @#encrypt@#

Total Dataframe Response: qo4e3thixsXd+0nU05ubMdtXhJ4Xd6CXzE/V26MfQqV+IIUQvVc=

Decryption Time: 397959 nanoseconds

- For the transmission of the longer message "Nice to meet you, Alice!", the unencrypted elapsed time is 1,477,084 ns, and the total time about encryption and decryption is 913,792 ns.

Please input the message (input 'exit' to quit):

Nice to meet you, Alice!

Server Response: Message Received: Nice to meet you, Alice!

Elapsed Time (nanoseconds): 1477084

Please input the message (input 'exit' to quit):

Nice to meet you, Alice!

Message Sent:

Key Sent: 4fx4mahh

Encrypted Message Sent: 8bHIeNYwKdri9DgdfDImYPWNHr0NnfY7rKLwLyytsD4=

Total Dataframe Sent: 4fx4mahh8bHIeNYwKdri9DgdfDImYPWNHr0NnfY7rKLwLyytsD4=

Encryption Time: 419750 nanoseconds

Server Response:

Key Response: 1y56l7e0

Decrypted Message Response: Message Received: Nice to meet you, Alice!

Total Dataframe Response: 1y56l7e00obx80IbkUdf9K2bAGWmhHRw0WLZlS2kbMkup0ELSguGzdLtl6s+4pTBGzq+gF+K

Decryption Time: 494042 nanoseconds

- Obviously, if DES encryption is enabled, the elapsed time of the C-S model will at least double. This may affect performance.
- However, DES has several notable weaknesses:
 1. Key Length: DES has a relatively short key length of only 56 bits, making it susceptible to brute force attacks with modern computing power.
 2. Limited Security: Due to advances in cryptanalysis, DES is no longer considered secure against determined attackers. Its small key size and fixed block size make it vulnerable to various attacks, including exhaustive key search and differential cryptanalysis.
 3. Block Size: DES operates on 64-bit blocks of data, which can be inefficient for modern applications that handle large amounts of data. This limitation can lead to issues with performance and compatibility with certain modes of operation.