Universidad del Valle De Guatemala

Facultad de Ingeniería

Redes

Laboratorio 3: Algoritmos de enrutamiento

Jose Hernández 20053
Pablo González 20362
Javier Mombiela 20067

Guatemala, 10 de septiembre 2023

1. Descripción de la práctica.
   Para la segunda parte del laboratorio 3, se explora el funcionamiento de los algoritmos que implementamos en la parte uno, pero esta vez las simulaciones son reales, es decir, nada es hard-coded. En la primera parte se utilizaron los algoritmos Djikstra, Flooding y Distance Vector Routing, de los cuales se habían hecho simulaciones exitosas pero con ejemplos hard-coded.

   Para este segunda parte, las simulaciones fueron reales, utilizando el servidor XMPP utilizado para el proyecto con el dominio alumchat.xyz. Para esta práctica lo que era requerido era poder hacer que los algoritmos funcionaran de forma real. Cada usuario sería simulado como un nodo, en donde esté dependiendo del algoritmo, podría realizar varias opciones.

   El cliente tiene un listener el cual esta a la espera de cualquier mensaje que le pueda llegar al usuario. Dependiendo del tipo del mensaje names|topo|echo|info|message, el listener llama a una función la cual se encarga de procesar el mensaje que ha entrado y realizar una seria de pasos.

   Con names y topo, el programa se encarga de hacer el setup de que Id pertenece a cada usuario y cuáles son sus vecinos. Con echo, el programa logra enviar un mensaje con un timestamp el cual el vecino debe responder con otro timestamp y así poder encontrar la distancia o peso entre ambos vecinos. Con el info, el programa recibe la tabla de enrutamiento de sus vecinos, actualiza la del usuario y en caso de haber cambios, la vuelve a enviar a sus vecinos. Por último, si es un mensaje, el programa ve si es para el usuario, en caso de ser cierto, despliega el mensaje, en caso de ser falso, mira en su tabla a qué vecino enviárselo y hace un foreward a dicho vecino.

2. Descripción de los Algoritmos utilizados y su implementación.
   A. Dijkstra:
      Para el algoritmo de Djikstra, solo se simuló el envío de paquetes, ya que este algoritmo es estático, y se debe de configurar todo el grafo para que este funcione de manera apropiada.

   B. Flooding:
      Para el algoritmo Flooding, también se simula solamente el envío de paquetes, ya que este algoritmo funciona en enviar el paquete a todos sus vecinos (excluyendo el vecino del que acaba de recibir el paquete), sin importar el peso, y esperar a que el destinatario reciba dicho paquete.

   C. Distance Vector Routing:
      Para el Distance Vector Router, si hay más opciones ya que primero, el nodo puede enviar un mensaje tipo “echo” el cual lograra definir la distancia entre él y sus vecinos. Luego se envía un mensaje tipo “info” el cual envía la tabla de enrutamiento a todos sus vecinos para que estos puedan actualizar su propia

tabla. Por último, también se puede enviar un paquete a un destinatario especificado.

D. Implementaciones de las funciones más importantes:

listener: Para poder escuchar cada mensaje entrante y poder manejarlo

```
client.xmpp.on("stanza", async (stanza) => {
        // console.log(stanza.toString());
        if (stanza.is("message")) {
            // console.log(stanza.toString());
            const type = stanza.attrs.type;
            const from = stanza.attrs.from.split("/")[0];
            const body = stanza.getChildText("body");

            if (type === "chat" && body) {
                console.log(`\n${from}: ${body}`);
                try {
                    const messageData = JSON.parse(body);
                    if (messageData.type === "message" && messageData.headers && messageData.payload) {
                        receivePacket(messageData);
                    }
                    else if (messageData.type === "info" && messageData.headers && messageData.payload) {
                        updateRoutingTable(messageData);
                    }
                    else if (messageData.type === "echo" && messageData.headers && messageData.payload) {
                        updateNeighbors(messageData);
                    }

                } catch (error) {
                    console.error(`Error parsing JSON: ${error}`);
                }
            }
            else if (type === "headline" && body) {
                try {
                    const messageData = JSON.parse(body);
                    if (messageData.type === "names" && messageData.config) {
                        setUpId(messageData.config);
                    }
                    else if (messageData.type === "topo" && messageData.config) {
                        setUpIdNeighbors(messageData.config);
                    }
                } catch (error) {
                    console.error(`Error parsing JSON: ${error}`);
                }
            }
        }
    });
```

sendEcho: Para ver la distancia entre cada vecino

```
async function sendEcho() {

    const echoMessage = {
        type: "echo",
        headers: {
            from: `${client.username}@${client.domain}`,
            to: "",
        },
        payload: {
            timestamp1: "",
            timestamp2: "",
        },
    };

    const neighbors = client.router.neighbors;

    for (let neighbor of neighbors) {
        const neighborJid = client.names[neighbor];
        const echoMessageCopy = { ...echoMessage };
        echoMessageCopy.headers.to = neighborJid;

        const timestamp1 = Date.now();
        echoMessageCopy.payload.timestamp1 = timestamp1;
        try {
            client.directMessage(neighborJid, JSON.stringify(echoMessageCopy));
        }
        catch (err) {
            console.log(err.message);
        }
    }
}
```

sendRoutingTable: Para poder enviar la tabla de enrutamiento a cada vecino

```
function sendRoutingTable() {

    // Creamos la stanza que lleva el payload del mensaje
    const infoMessage = {
        type: "info",
        headers: {
            from: `${client.username}@${client.domain}`,
            to: "",
            hop_count: 0,
        },
        payload: client.router.routingTable,
    };


    for (let neighbor of client.router.neighbors) {
        const neighborJid = client.names[neighbor];
        const infoMessageCopy = { ...infoMessage };
        infoMessage.headers.to = neighborJid;
        client.directMessage(neighborJid, JSON.stringify(infoMessageCopy));

    }

}
```

sendPacket: Para enviar un paquete a un destinatario específico

```
function sendPacket() {
    console.log('\nSEND PACKET:')
    rl.question("Usuario: ", user => {

        const userJid = `${user}@${client.domain}`;
        const nodeId = Object.keys(client.names).find(key => client.names[key] === userJid);

        rl.question("Mensaje: ", message => {

            //agararr el nextHop
            const nextHop = client.router.getNextHop(nodeId);
            //agarrar el usuario del nextHop
            const nextHopRouter = client.names[nextHop];
            console.log(`Destination : ${userJid} NextHop: ${nextHopRouter }`)


            const messageData = {
                type: "message",
                headers: {
                    from: `${client.username}@${client.domain}`,
                    to: userJid,
                    hop_count: 0,
                },
                payload: message,
            };

            client.directMessage(nextHopRouter, JSON.stringify(messageData));
            console.log(`Mensaje enviado a ${nextHopRouter}`);
            submenuDV();

        });
    });
}
```

3. Resultados

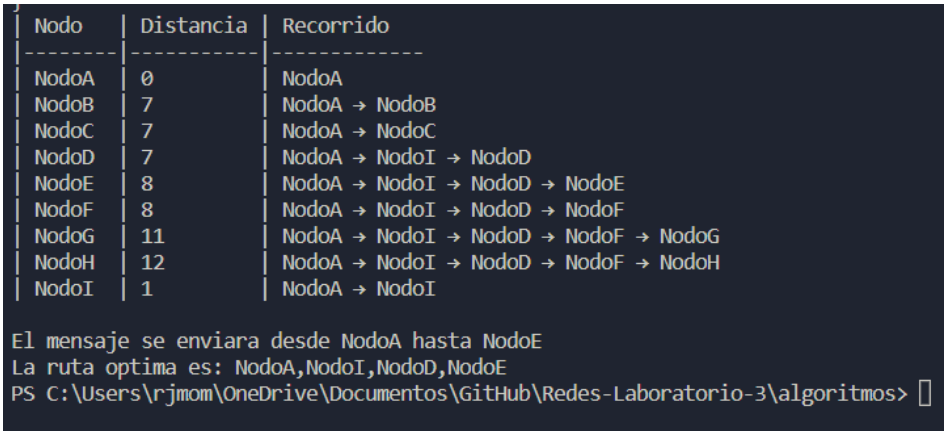
Se utilizó la siguiente topología para poder probar los algoritmos

```
[
    {"type":"names", "config":{"A": "messi10@alumchat.xyz", "B": "joehueco@alumchat.xyz", "C": "pablishi@alumchat.xyz"} },

    {"type":"topo", "config":{"A": ["B", "C"], "B": ["A", "C"], "C": ["A", "B"]} }
]
```

Djikstra

El nodo A verifica las rutas óptimas para cada nodo y en este caso ve que para llegar al nodo E tiene que mandar el paquete al nodo I primero.

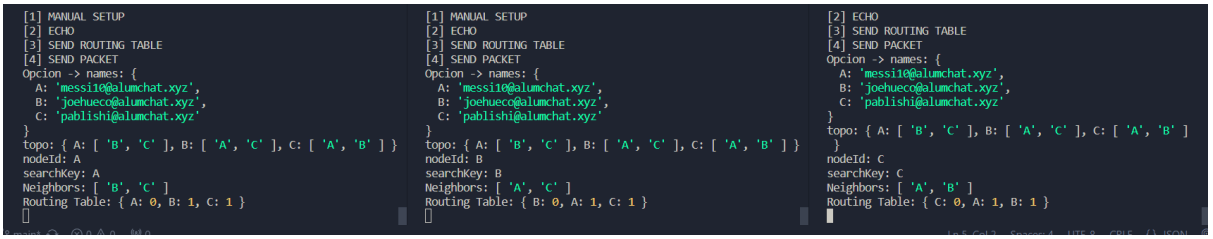


Nodo	Distancia	Recorrido
NodoA	0	NodoA
NodoB	7	NodoA → NodoB
NodoC	7	NodoA → NodoC
NodoD	7	NodoA → NodoI → NodoD
NodoE	8	NodoA → NodoI → NodoD → NodoE
NodoF	8	NodoA → NodoI → NodoD → NodoF
NodoG	11	NodoA → NodoI → NodoD → NodoF → NodoG
NodoH	12	NodoA → NodoI → NodoD → NodoF → NodoH
NodoI	1	NodoA → NodoI

```
El mensaje se enviara desde NodoA hasta NodoE
La ruta optima es: NodoA,NodoI,NodoD,NodoE
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos>
```

Flooding

Setup de los tres nodos: Se hace el setup manual en cada uno de los tres clientes diferentes y se logra ver como se pueden ver todos los vecinos.

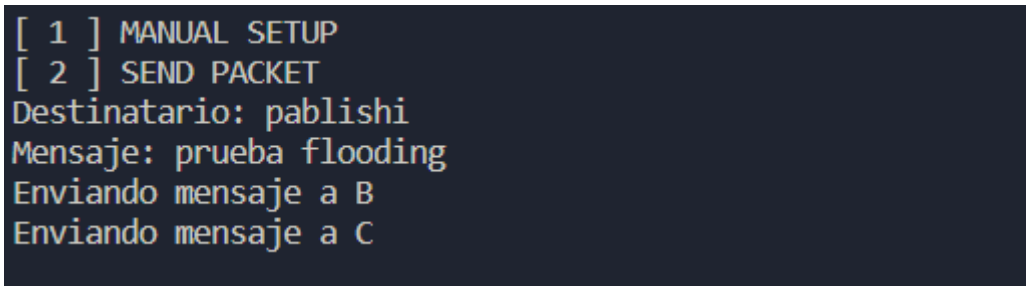


```
[1] MANUAL SETUP
[2] ECHO
[3] SEND ROUTING TABLE
[4] SEND PACKET
Opcion -> names: {
  A: 'messi10@alumchat.xyz',
  B: 'joehuecc@alumchat.xyz',
  C: 'pablishi@alumchat.xyz'
}
topo: { A: [ 'B', 'C' ], B: [ 'A', 'C' ], C: [ 'A', 'B' ] }
nodeId: A
searchKey: A
Neighbors: [ 'B', 'C' ]
Routing Table: { A: 0, B: 1, C: 1 }
```

```
[1] MANUAL SETUP
[2] ECHO
[3] SEND ROUTING TABLE
[4] SEND PACKET
Opcion -> names: {
  A: 'messi10@alumchat.xyz',
  B: 'joehuecc@alumchat.xyz',
  C: 'pablishi@alumchat.xyz'
}
topo: { A: [ 'B', 'C' ], B: [ 'A', 'C' ], C: [ 'A', 'B' ] }
nodeId: B
searchKey: B
Neighbors: [ 'A', 'C' ]
Routing Table: { B: 0, A: 1, C: 1 }
```

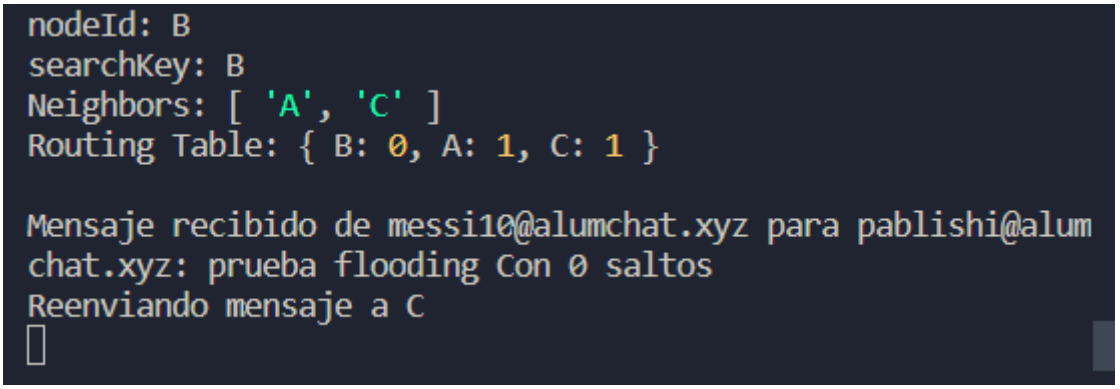
```
[2] ECHO
[3] SEND ROUTING TABLE
[4] SEND PACKET
Opcion -> names: {
  A: 'messi10@alumchat.xyz',
  B: 'joehuecc@alumchat.xyz',
  C: 'pablishi@alumchat.xyz'
}
topo: { A: [ 'B', 'C' ], B: [ 'A', 'C' ], C: [ 'A', 'B' ] }
nodeId: C
searchKey: C
Neighbors: [ 'A', 'B' ]
Routing Table: { C: 0, A: 1, B: 1 }
```

Envio de paquete: El nodo A envia el paquete con destino al nodo C, pero ya que es flooding, el programa envia el paquete a ambos vecinos.

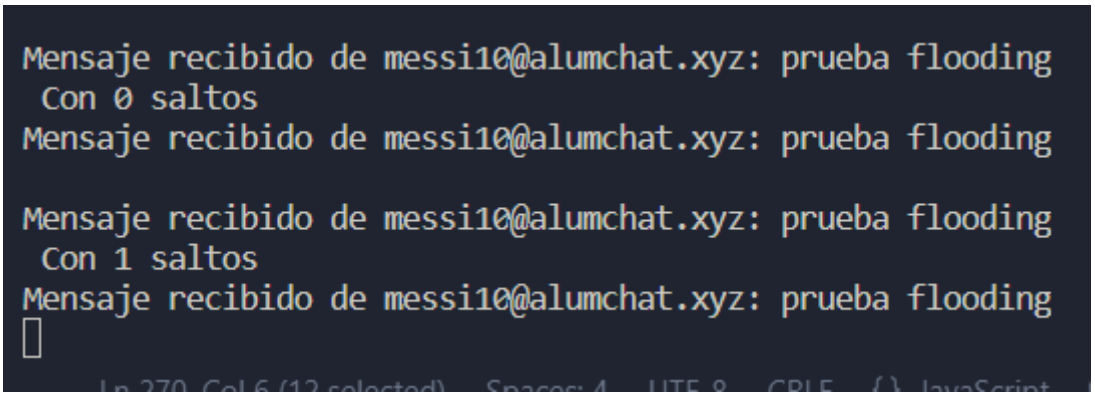


```
[ 1 ] MANUAL SETUP
[ 2 ] SEND PACKET
Destinatario: pablishi
Mensaje: prueba flooding
Enviando mensaje a B
Enviando mensaje a C
```

El nodo B recibe el mensaje, revisa que no es para él y lo reenvía el nodo C, y modifica el salto.

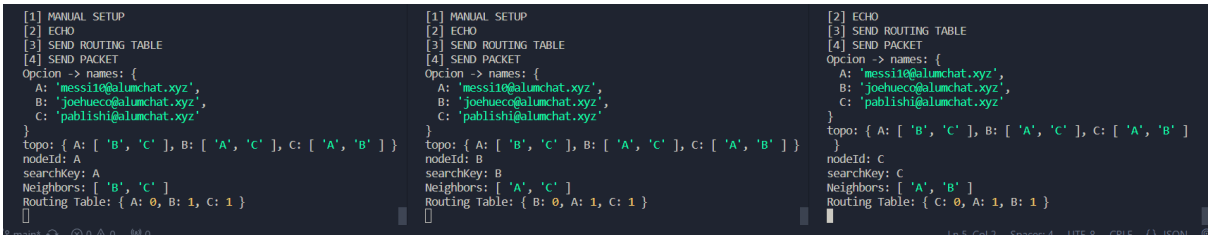


El nodo C ha recibido dos paquetes, uno directamente desde A y otro que paso por B, y este imprime el mensaje dos veces.

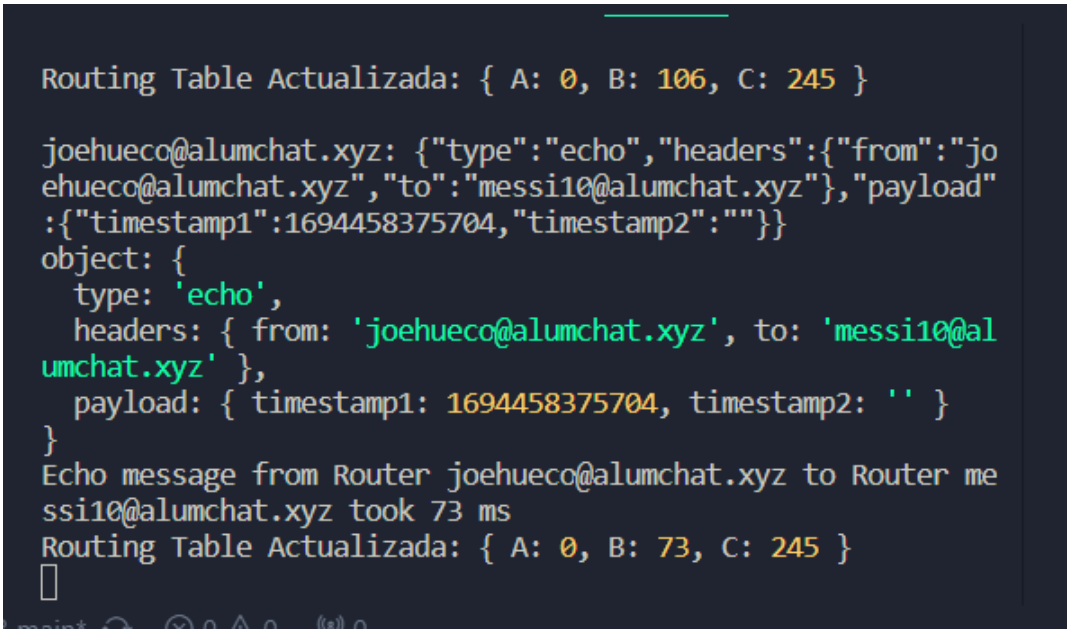


Distance Vector

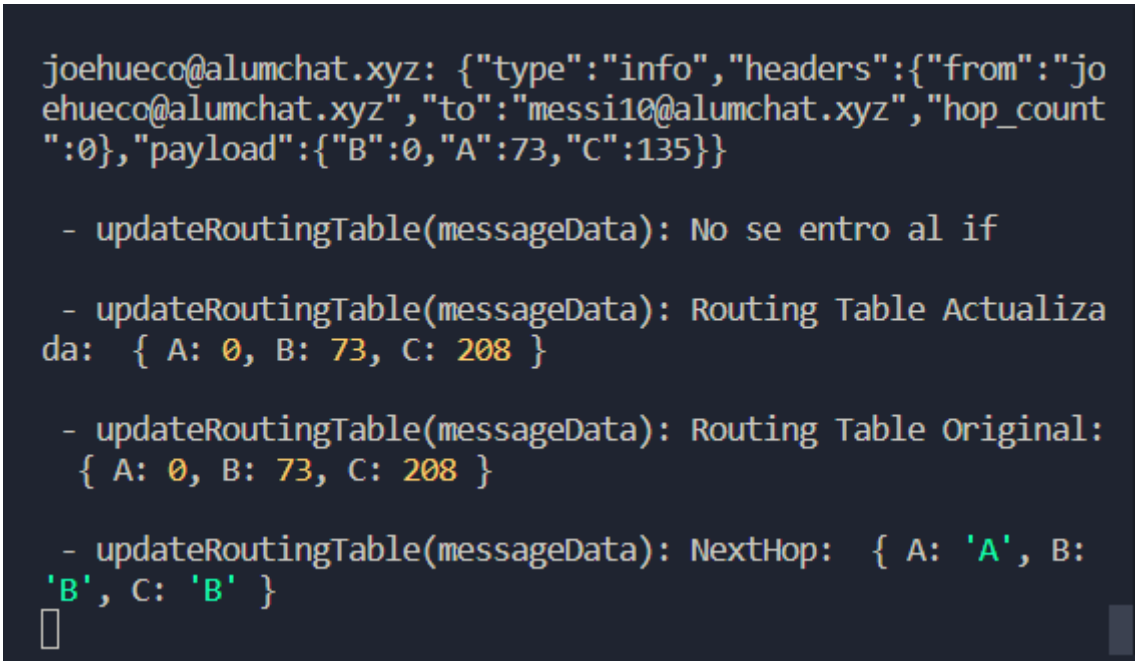
Setup de los tres nodos: Se hace el setup manual en cada uno de los tres clientes diferentes y se logra ver como se pueden ver todos los vecinos.



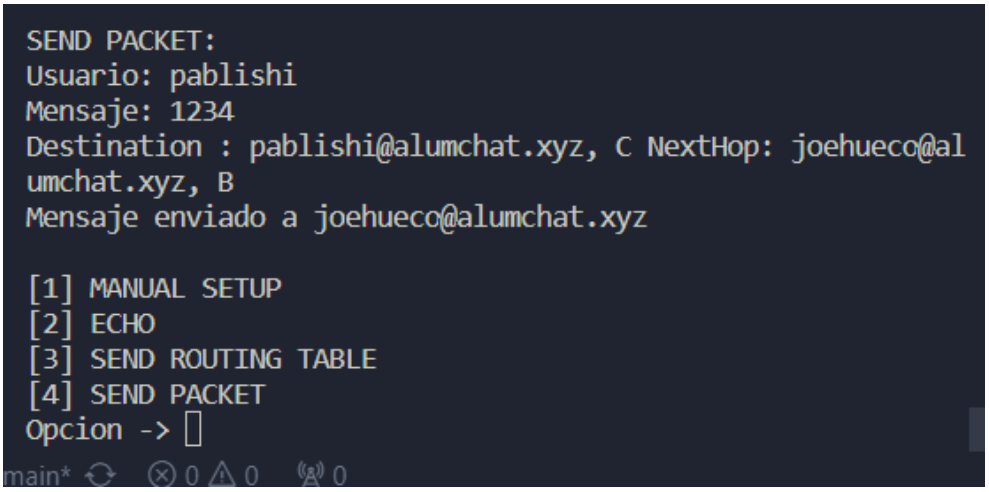
Routing Table Nodo A: Se envía el echo desde todos los clientes y podemos ver como se reciben los dos timestpams para poder definir el peso de cada vecino.



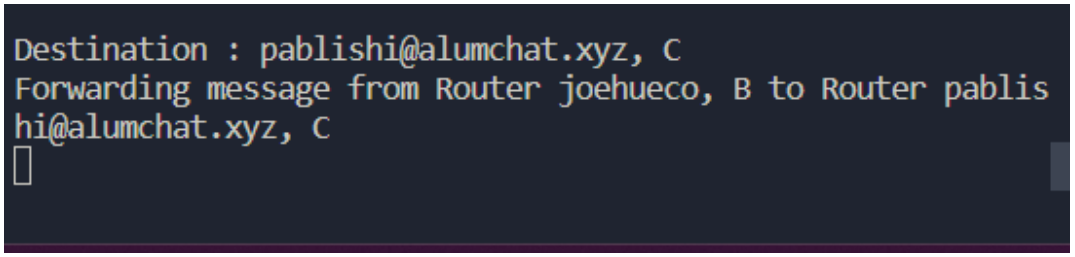
Routing Table Nodo A: Se envían las tablas de enrutamiento entre todos los vecinos y podemos ver como la tabla de enrutamiento del nodo A ha sido modificado y ahora el nextHop hacia C es B, y no C directamente.



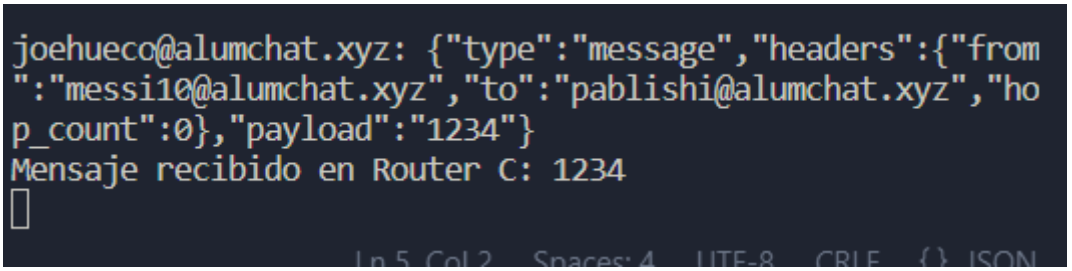
Proceso del envio de paquete: El router A manda mensaje al router C, pero sabe que es mejor ir por router B, por lo cual se lo envia a el.



El router B recibe el paquete, lo revisa y ve que no es para el, entonces revisa su tabla de enrutamiento y hace el forward.



El router C recibe el paquete, revisa el mensaje y como si es para el, imprime el mensaje final.



4. Discusión

   Los resultados de la práctica muestran que se logró implementar con éxito los tres algoritmos de enrutamiento: Dijkstra, Flooding y Distance Vector.

   Para Dijkstra, los resultados validan que el nodo A pudo identificar la ruta óptima para llegar al nodo E pasando por el nodo I. Esto demuestra que el algoritmo funcionó correctamente para encontrar la ruta más corta en esta topología estática.

   Con Flooding, los resultados confirman que los nodos pudieron hacer broadcast de los paquetes a todos sus vecinos, incluso cuando no eran el destino final. Esto permite

que los paquetes eventualmente lleguen al destino a través de múltiples caminos. Como se puede observar con los resultados, eventualmente el nodo C recibió el paquete dos veces y hasta los hops fueron modificados.

Finalmente, para Distance Vector, los resultados verifican que los nodos pudieron medir los pesos entre vecinos usando ECHO, compartir sus tablas de enrutamiento con INFO y reenviar paquetes MESSAGE por las mejores rutas disponibles. El nodo A pudo enviar un paquete al nodo C pasando por B al detectar que esa era la ruta más corta.

En conclusión, los resultados indican una implementación exitosa de los algoritmos que cumple con el comportamiento esperado en cuanto a descubrimiento de rutas, redundancia, y selección de caminos óptimos.

5. Conclusiones
    a. El algoritmo Djikstra no se puede simular de la misma manera que los otros dos algoritmos utilizados, ya que este es un algoritmo estático.
    b. Para el algoritmo Flooding no es necesario enviar mensajes tipo echo e info, ya que este algoritmo de igual manera enviara el paquete a todos sus vecinos sin importar los pesos de cada vecino y sin importar cuál es el camino más corto.
    c. Distance vector machine es un algoritmo dinámico y también más complejo, lo cual lo hace muy preciso y más eficiente que flooding, ya que flooding es robusto pero a costo de su eficiencia.

6. Comentario grupal
   La segunda parte del laboratorio tres fue algo que nos interesó a todo el grupo, ya que nos desafió a ir un paso más y no solo implementar los algoritmos, sino que hacerlos funcionar con el servidor del proyecto uno. Fue algo interesante ver como funcionaba el código cada vez más y a la hora de conectarse cada uno desde su casa e intentar el funcionamiento y ver que si funcionaba como debía, fue un sentimiento de satisfacción. Este laboratorio nos enseñó a ver como en realidad es que funcionan estos algoritmos en la vida real, lo cual fue muy entretenido y nos ayudó a aprender más a fondo sobre los tres algoritmos que implementamos.

7. Referencias

Pedraza, L. F., López, D., & Salcedo, O. (2011). Enrutamiento basado en el algoritmo de Dijkstra para una red de radio cognitiva. Tecnura, 15(30), 94–100. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-921X2011000300010#:~:text=El%20algoritmo%20de%20Dijkstra%20proporciona

¿Qué es una inundación? - definición de techopedia - Redes 2023. (2023). Icy Science. https://es.theastrologypage.com/flooding

Vector de distancias. (2020, September 9). Wikipedia. https://es.wikipedia.org/wiki/Vector_de_distancias