UNIVERSIDAD NACIONAL DEL SUR

TESIS DE GRADO

Incentived Mesh Networking: Integración de Althea y Libremesh

Autor:
Luciano BRUNA

Directores:
Dr. Claudio DELRIEUX
Ing. Eduardo PAOLINI

Consejeros:
Dr. Carlos MATRÁNGOLO
Ing. Lisandro N. PÉREZ MEYER

Departamento de Ingeniería Eléctrica y Computadoras

7 de mayo de 2019

«A remarkable glorious achievement is what a long series of unremarkable unglorious tasks looks like from far away.»

Mark Manson

UNIVERSIDAD NACIONAL DEL SUR

Resumen

Universidad Nacional del Sur
Departamento de Ingeniería Eléctrica y Computadoras

Ingeniero Electrónico

Incentived Mesh Networking: Integración de Althea y Libremesh

por Luciano BRUNA

En la última década, las redes *mesh* han pasado de ser un concepto teórico a convertirse en redes distribuidas, con links autónomos y autogenerativos. Dentro de este crecimiento sostenido hay iniciativas como la empresa Althea que proponen redes en las que los nodos se paguen entre ellos por cantidades determinadas de ancho de banda. De esta manera, en vez de haber un ISP en la cima de una jerarquía colectando pagos mensuales, muchas personas diferentes pueden ganar dinero al expandir y hacer la red más robusta. Esto implica abaratamientos de costos por el internet, reducción de cortes de conexión, etc.

El problema de estos proyectos es que al estar en sus fases iniciales de desarrollo, su software es compatible con un número muy reducido de routers en el mercado. En este trabajo, se realizó un estudio exhaustivo de Althea y un *firmware* candidato con el que se podría integrar, Libremesh. Luego se avanzó con la integración de su software hasta llegar a una versión estable que fue probada en una máquina virtual.

Agradecimientos

Este trabajo no podría haber sido realizado de no ser por el soporte sostenido por parte de los directores Claudio Delrieux y Eduardo Paolini. Ellos han sido una guía impresindible durante el desarrollo del proyecto, su documentación y publicación. El agradecimiento que se tiene por ellos es por lo tanto, enorme.

También es importante mencionar a los *advisors* Carlos Matrángolo y Lisandro Pérez Meyer, quienes a partir de su experiencia, fueron clave en la solución de algunos problemas que fueron surgiendo en el transcurso del trabajo. Además, han sido vitales las participaciones de Nicolás Pace y Justin Kilpatrick, desarrolladores de Libremesh y Althea respectivamente. Ellos han facilitado la información y han abierto el espacio necesario de trabajo para la integración entre sus iniciativas. La fundación Pampa Energía fue un soporte económico durante los estudios y es por ende también, un actor más en este trabajo.

Y finalmente, están mi familia y amigos, quienes siempre fueron un soporte incondicional y representan la base de cualquier logro que se alcance en el futuro.

Gracias, gracias, gracias.

Índice general

Dedicado a Diana Sanchez.

Capítulo 1

Introducción

Durante los últimos años, las *mesh networks* [1] han pasado de ser un concepto teórico a convertirse en dispositivos comercialmente disponibles que prometen crear redes distribuidas, capaces de crear links entre sí y mantenerlos autónomamente. El IEEE creó el grupo de trabajo 802.11s, que define cómo los dispositivos inalámbricos pueden interconectarse para crear una red mesh inalámbrica [1]. Esto tiene la intención de guiar a institutos de investigación y a la industria para desarrollar un estándar, con el objetivo de proveer interoperabilidad entre todos los dispositivos. Mientras el estándar 802.11 sigue bajo desarrollo, hay un número de protocolos de red que están actualmente disponibles.

De todas las aplicaciones de redes mesh, en este trabajo se destacan las redes comunitarias no cableadas, o mesh WCNs (*Mesh Wireless Comunity Networks*), cuya topología está representada en la figura (1.1). Éstas han sido desarrolladas por movimientos de entusiastas de redes no cabledas, quienes usando equipamiento de bajo costo para una interconexión gratuita, han creado redes completamente autónomas. Su aparición se debe principalmente al interés de estos grupos de lograr un servicio equivalente a los ofrecidos por redes 2.5G y 3G pero de manera gratuita[2].

Las razones para participar en una red mesh son muy variadas. Aquellas personas que creen que la conectividad de banda ancha debería ser libre y gratuita y que las barreras impuestas por los oligopolios de los ISPs deben ser eliminadas usualmente son los primeros en unirse a WCNs. Esto último se agrava al existir muchas zonas en las que solo hay un ISP disponible [3]. En el cuadro (1.1) se reportan algunas de las mesh WCNs más significativas a nivel mundial categorizadas según el año en que han sido diseñadas y la cantidad de nodos. Cabe destacar que todas ellas parten de una iniciativa comunitaria, es decir que son el resultado de esfuerzos colectivos de voluntarios individuales y funcionan sin fines de lucro.

En el estudio de este fenómeno se destacó la iniciativa de la empresa Althea, con la cual se desarrolló este trabajo en conjunto. Althea propone un sistema basado en el *incentivized mesh* [2] en donde se permite que los routers se paguen entre ellos por cantidades determinadas de ancho de banda. De esta manera, en vez de haber un ISP en la cima de una jerarquía colectando pagos mensuales, muchas personas diferentes pueden ganar dinero al expandir y hacer la red más robusta. De ahí surge el adjetivo *incentived*. Esto implica varias ventajas tales como el abaratamiento de los costos por el internet, la reducción de cortes de conexión, etc.

[1] Una red *mesh* es una topología de red local en la que la infraestructura de nodos (routers, Access Points y otros dispositivos) se conectan directamente, dinámicamente y sin jerarquía a tantos otros nodos como sea posible cooperando entre todos para hacer ruteo de la forma más eficiente posible.

[2] El *incentivized mesh* es un modelo económico que motiva a las comunidades a construir infraestructura robusta e independiente de ISPs utilizando cripto-monedas como medio de pago por ancho de banda.

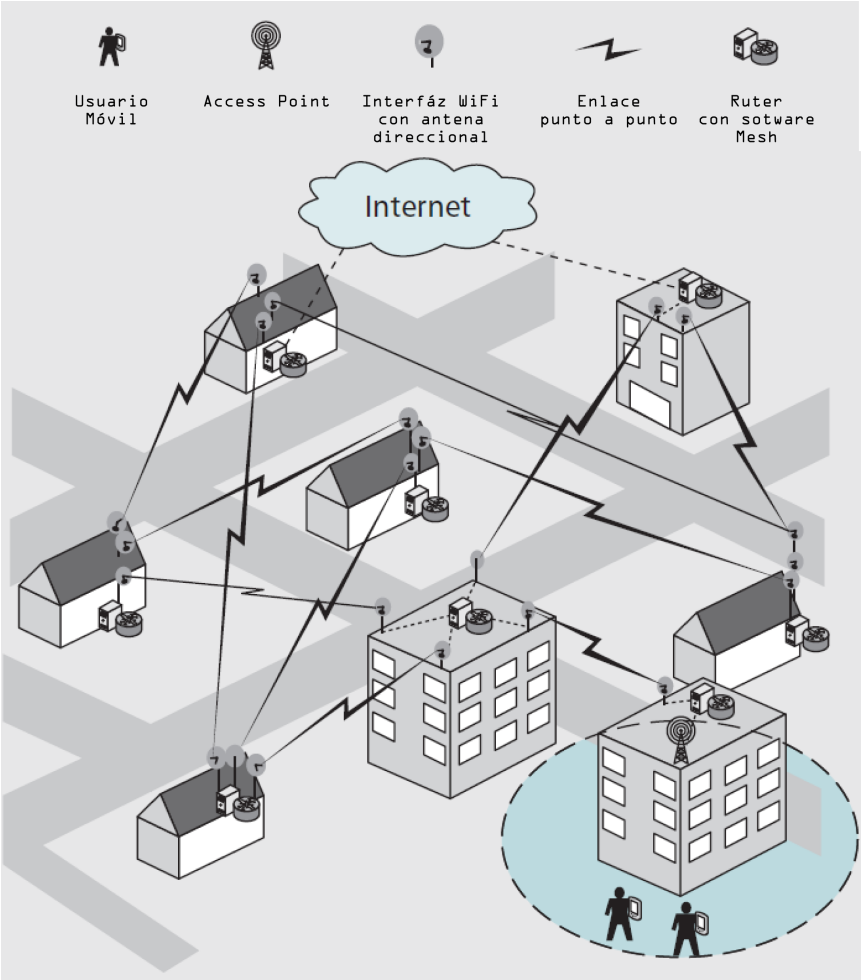

FIGURA 1.1: *Topología de redes mesh WCNs.*

Sin embargo, Althea presenta varias problemáticas inherentes a estar en sus primeras fases de desarrollo. Su software es compatible con un número muy reducido de routers en el mercado [3] lo que dificulta su ensayo y retrasa su escalabilidad. Por lo tanto, un paso muy importante para su crecimiento es y será la compatibilización con la mayor cantidad de firmwares posibles y por ende, de dispositivos. Entre los primeros candidatos, el firmware de *Libremesh* [4] ganó mucho interés recientemente ya que este último será compatible con *Librerouter*, un fabricante de routers que ya cuenta con más de 4500 precompras.

En este proyecto se realizó un estudio y análisis exhaustivo de los dos actores: Libremesh y Althea. Manteniéndose en constante contacto con las empresas detrás de ambas iniciativas, se avanzó con la integración de su software hasta llegar a una versión estable y se la probó para un dispositivo en particular. Todo esto implicó el uso de herramientas como *git*, *lua*, *makefile*, *bash*, etc., lenguajes como *javascript*, *rust* y *c*, configuraciones basadas en archivos de formatos *.toml* así como el estudio de una amplia gama de conceptos que se verán en el capítulo 2. El capítulo 3 explica

[3] El software de Althea consiste principalmente de binarios instalables como paquetes en firmwares específicos. En la fecha de escritura de este trabajo, esos firmwares solo funcionan de forma estable en menos de 10 modelos de routers.

[4] Libremesh es un *framework* utilizado para crear firmwares para dispositivos partícipes de una red WCNs. Está basado en *OpenWRT*, un firmware de routers basado en Linux utilizado en una amplia gama de routers.

Nombre de la Red	**Ubicación**	**Año de Fundación**	**Cantidad de nodos**
SeattleWireless	Seattle, WA, Estados Unidos	2000	80
AWMN	Athena, Grecia	2002	2473
CUWiN	Urbana, IL, Estados Unidos	2002	48
Berlin's Freifunk	Berlin, Alemania	2002	316
Wireless Leiden	Leiden, Países Bajos	2002	73
NetEquality Roofnet	Portland, OR, Estados Unidos	2007	126
NYCwireless	New York, Estados Unidos	2001	145

CUADRO 1.1: Comunidades de redes mesh WCNs en todo el mundo (basado en información del año 2009) [2].

con mucho mayor detalle en qué consiste la integración de ambas iniciativas. Finalmente, en el capítulo 4 se describen todos los procedimientos llevados a cabo con las dificultades que se fueron presentando y las decisiones tomadas para superarlas.

Capítulo 2

Marco teórico

Es este capítulo se presentan los conocimientos básicos necesarios para entender todos los conceptos involucrados en este proyecto. Primero, se incursiona en conceptos básicos de redes y en algoritmos de ruteo específicos. Éstos son claves para el entendimiento tanto de Libremesh como de Althea. Luego, se explican temas de criptografía que son claves para luego comprender el funcionamiento de *Blockchain*, la tecnología necesaria para las transacciones entre routers.

2.1. Definiciones Iniciales

2.1.1. Redes Ad Hoc y Redes Mesh

En gran parte de la bibliografía consultada, se han utilizado los términos redes *ad hoc* y redes mesh de manera indistinta. Si bien esto no representa ningún error en esos casos, puede generar confusiones si no se conoce la diferencia entre ellos. Por eso es importante destacarla.

Una *red ad* hoc implica el compromiso cooperativo de un conjunto de nodos wireless sin ningún tipo de intervención por parte de algún access point centralizado o de alguna infraestructura existente. Este tipo de red tiene características claves de ser auto-generativas y auto-curativas y no depender de ningún servicio centralizado de ningún nodo en particular. Una red *ad hoc* puede ser *wireless*, en cuyo caso sus dispositivos, tales como laptops o sensores, realizan una función de ruteo para enviarse datos creando una forma arbitraria de topología de red. Cuando todos los dispositivos son móviles, se habla de redes MANET [1], que pueden cambiar de topología muy rápidamente y de forma impredecible. Un buen ejemplo de redes MANET son las redes de sensores no cableados.

Una red *wireless mesh* se caracteriza por tener routers estáticos o cuasi-estáticos dedicados que tienen el trabajo de realizar la función de ruteo de paquetes a lo largo de la red. Además, están los dispositivos clientes, quienes no tienen funcionalidad de ruteo. Estos están conectados a los *routers* wireless.

Existen muchos protocolos de ruteo compatibles con estos tipos de redes. La gran mayoría de ellos parte de grupos académicos que buscan optimizar diferentes cuestiones como la velocidad de convergencia o evitar los lazos de ruteo. A estos se les prestará una mayor atención en la sección (2.2). Además, el grupo de trabajo 802.11s de la IEEE define un estándar para las redes mesh en donde se define, entre otras cosas, el método de ruteo para este tipo de redes.

[1] MANET: Abreviación de Mobile Ad-Hoc Network.

2.2. Protocolos de Ruteo

Los protocolos para redes mesh se clasifican en tres categorías: *reactivos*, *proactivos* e *híbridos* [4]. Los protocolos reactivos solo buscan un camino entre nodos cuando hay datos para ser enviados. Este método tiene la ventaja de no gastar ancho de banda de la red con mensajes de control cuando no se requiere la transmisión de datos. Los protocolos reactivos idealmente se utilizan en redes *ad hoc* con nodos móviles donde los caminos de datos podrían cambiar contínuamente.

Por otro lado, los protocolos proactivos establecen y mantienen activamente caminos de datos para nodos tanto si los datos necesitan ser enviados como si no. Esto permite una latencia menor a la hora de enviar datos a través de la red porque el camino óptimo ya es conocido. De todas maneras, esto tiene un costo computacional mayor y la necesidad de una administración de red mucho más compleja.

Los protocolos híbridos comparten las propiedades tanto de los protocolos proactivos como de los reactivos. En general, intentan usar características proactivas o reactivas según el escenario, lo cual explota sus fortalezas y es por eso que pueden lograr un mayor nivel de escalabilidad. En general, son más complejos en comportamiento, lo cual hace que sean más difíciles de implementar que los protocolos puramente proactivos o reactivos.

En la sección 2.2.1 se explicará el algoritmo *vector distancia* que es la base de varios protocolos de ruteo muy conocidos como RIP y OLSR. Se pondrá en discusión la característica principal y la problemática de este último en 2.2.2. Luego se detallarán protocolos más eficientes para redes mesh en las secciones 2.2.3, 2.2.5 y 2.2.6. Finalmente, en 2.2.7 se compararán todos los protocolos previamente descritos.

2.2.1. Algoritmo vector distancia

El algoritmo *vector distancia* se trata de uno de los más importantes junto con el *estado de enlace* [2]. Se basa en el algoritmo de Bellman-Ford para calcular las rutas. Requiere que un router informe a sus vecinos de los cambios en la topología periódicamente y en algunos casos cuando se detecta un cambio en la topología de la red.

Se basa en calcular la dirección y la distancia hasta cualquier enlace en la red. El costo de alcanzar un destino se lleva a cabo usando cálculos matemáticos como la métrica del camino. Una de las métricas más utilizadas es el número de *hops* [3].

Los cambios son detectados periódicamente ya que la tabla de ruteo de cada router se envía a todos los vecinos que usan el mismo protocolo. Una vez que el router tiene toda la información, actualiza su tabla e informa a sus vecinos de los mismos. Este proceso se conoce también como *enrutamiento por rumor* ya que los nodos utilizan la información de sus vecinos y no pueden comprobar a ciencia cierta si ésta es verdadera o no.

El algoritmo de Bellman-Ford se adapta perfectamente al modo de aprendizaje de los nodos que "nacen", es decir, cuando se conectan a la red. A medida que el algoritmo progresa, el nuevo nodo va adquiriendo más información sobre el resto de nodos de la red. Este algoritmo converge rápidamente cuando se conectan nuevos nodos. Por ello se suele decir que las buenas noticias viajan rápido por la red.

[2] Estado de enlace es un algoritmo de ruteo alternativo al de *vector distancia*. Fue implementado por primera vez en 1979 por ARPANET.

[3] En una red, se le dice *hop* a una porción de camino entre una fuente y su destino. La cuenta de *hops* se refiere al número de dispositivos intermediarios a través de los cuales deben pasar los datos.

Un problema que padece este algoritmo es el de la transmisión de malas noticias por la red tales como la ruptura de un enlace o la desaparición de un nodo. Este algoritmo converge lentamente en estos casos. Aunque el principal inconveniente de este algoritmo es el de la *cuenta a infinito*. Para ilustrarlo, se puede tomar como ejemplo el de la figura (2.1), en donde cada círculo representa un nodo, y las lineas que los unen los caminos con un costo igual a 1.

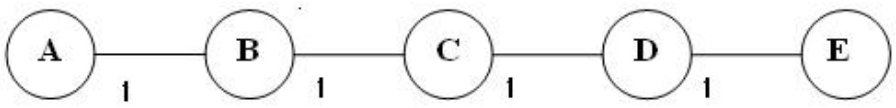

FIGURA 2.1: *Topología de red para ejemplificar la limitación del protocolo vector distancia.*

- Inicialmente A está desactivado. Cuando A se activa, B se entera de que A existe al recibir su vector distancia y actualizar su tabla indicando que A dista 1.
- El nodo C se entera de que A existe porque B le indica que tiene un enlace hacia A de coste 1. Entonces C actualiza su tabla registrando una trayectoria hacia A de coste 2.
- Si el nodo A se desconecta entonces B no recibe el vector distancia de A. Sin embargo el nodo C le dice que tiene una trayectoria hasta A de distancia 2. B no sabe que la trayectoria de C a A pasa por el mismo y por tanto cree que puede llegar a A a través de C por lo que actualiza su tabla registrando la distancia 2 + 1 = 3 hasta A.
- En el siguiente intercambio, el nodo C comprueba que sus vecinos B y D tienen una trayectoria hasta A de distancia 3. C calcula su propia distancia hasta A en 3 + 1 = 4. En los siguientes intercambios, los nodos elevan ilimitadamente su distancia a A (cuenta a infinito).

Mientras no se interrumpa la cuenta a infinito, el algoritmo no converge. Aunque se han propuesto diversas soluciones a este problema. El protocolo RIP establece el siguiente criterio:

$$\infty = 16.$$

Es decir, cuando la cantidad de incrementos de distancia supera los 15 en un período de tiempo relativamente corto, RIP considera que se está en un lazo infinito y corta las iteraciones. El protocolo OLSR utiliza un acercamiento más eficiente, como se verá en (2.2.2).

2.2.2. Protocolo OLSR

El protocolo de ruteo *Optimized Link State Routing* (o OLSR) emplea el algoritmo de *vector distancia* pero tiene muchos menos conteos a infinito que el protocolo RIP.

Para mantener informados a los nodos de la red de los cambios topológicos de la misma, utiliza el mecanismo *Multi-Point Relay* (o MPR) para hacer *flooding* [4]. A diferencia del flooding convencional, el método MPR evita que todos los nodos vecinos retransmitan los paquetes. Todos los nodos de la red seleccionan entre sus vecinos un conjunto de retransmisores (o *multi point relays*) encargados de retransmitir los mensajes que envía el nodo inicial. Los demás vecinos del nodo no pueden retransmitir, lo cual reduce significativamente el tráfico generado respecto al flooding tradicional.

Hay varios criterios para elegir los MPRs de un nodo. En general, se considera que el conjunto de MPRs de un nodo debe verificar que son capaces de alcanzar a todos los vecinos situados a 2 *hops* del nodo que los define como MPR.

El protocolo OLSR es un protocolo muy usado pero como fue desarrollado en los inicios del 2000 no está pensado para rutear información en redes mesh. De hecho, hay multiples fuentes que demuestran que OLSR sufre de muchos conteos a infinito en redes mesh [5] y que su desempeño con respecto a otros protocolos de ruteo como los que se presentarán en las secciones 2.2.3, 2.2.5 y 2.2.6 es mucho menor [4].

2.2.3. Protocolo BatMan

El protocolo BatMan (o Better Aproach to Mobile Ad-hoc Network) surge como la solución a todos los loops de ruteo que ocurren en una red mesh que utilice OLSR. Es un protocolo pro-activo ya que hace que periódicamente todos los nodos de la red envíen *hello packets* [5] a sus vecinos. Cada uno de estos mensajes tienen tres partes:

- Dirección de recepción
- Dirección de remitente
- Secuencia única de números

Cada vecino que recibe el paquete sustituye la dirección de remitente por su propia dirección. Luego realiza un *re-broadcasting* [6] del mensaje. Cada vez que un nodo recibe un mensaje cuya secuencia única de números fue generada por si mismo, verifica que ese mensaje fue emitido por él anteriormente. Esto le permite saber si la conexión con otros nodos es o no bi-direccional.

Para explicar el algoritmo con el que trabaja BatMan se modela a la red como $G = (N, E)$, donde N representa un conjunto de nodos y E representa un conjunto de links entre pares de nodos. Para cada nodo $i \epsilon N$, existe un conjunto K de vecinos a 1-hop de distancia. El mensaje desde la fuente $s \epsilon N$ a un destino d es transmitido sobre el link $(s, d) \epsilon E$ si d es también un elemento de K. Si no fuera el caso, se transmitiría sobre una ruta de múltiples hops hecha por un link (s, i) y una ruta $[i, d]$, donde i es un nodo dentro de K y (s, i) es un link en E. La ruta $[i, d]$ representa una ruta del nodo i al nodo d a través de la *subnet* $S = (N - \{s\}, A - \{(s, i) : i \epsilon K\})$.

Se puede separar al algoritmo en cuatro pasos:

1. Considerar el mensaje de ruteo m desde el nodo s al nodo d en la red G. Eliminar todos los links (s, i) $\forall i \neq K$ para reducir el grafo.

[4] Se le dice *flooding* a la acción de enviar un paquete de actualización de métricas a *todos* los nodos de la red. Cuando un nodo desea hacer *flooding* primero le envía el paquete deseado a todos sus nodos vecinos. Luego, estos retransmiten el mensaje hasta que éste llegue a todos los nodos de la red.

[5] Se le dice *hello packet* a una corta cadena de texto que un nodo de una red le envía a otro para informarle de su existencia.

[6] En una red mesh, *broadcasting* es la transmisión de un cierto mensaje a todos los nodos vecinos.

2. Asociar cada link con el peso w_{si} donde w_{si} es el número de *hello packets* recibidos desde el destino a través del nodo i durante un período de tiempo reciente.

3. Encontrar el link con el peso w_{si} más grande en el sub-grafo y enviar m a través del link (s, i).

4. Si $i \neq d$ entonces repetir los pasos de 1 a 4 para rutear el mensaje desde i hasta d en el sub-grafo S.

Las figuras (2.2), (2.3) y (2.4) ilustran el funcionamiento del algoritmo en el siguiente escenario:

- El nodo A quiere enviar un mensaje al nodo F. Solo considera este conjunto de links $\{(A, B), (A, C), (A, D)\}$ a sus vecinos $\{B, C, D\}$. Esto se aprecia en la figura (2.3).
- Determinar el mejor link como el link con el mayor número de *hello packets* recibidos desde el nodo F.
- Suponiendo que (A, B) es el mejor link entonces enviar el mensaje a través de dicho link.
- Como el nodo B no es el destino, reducir el grafo N al grafo S y repetir los pasos del algoritmo de 1 a 4. Esto se aprecia en la figura (2.4).
- El nodo B solo considera el conjunto de links $\{(B, C), (B, E)\}$ hacia sus vecinos $\{C, E\}$.
- Determinar el mejor link como el link con el mayor número de *hello packets* recibidos desde el nodo F.
- Suponiendo que (B, E) es el mejor link entonces enviar el mensaje a través de dicho link.
- Como el nodo E no es el destino, reducir el grafo N al grafo S y repetir los pasos del algoritmo de 1 a 4.
- El nodo E solo considera este conjunto de links $\{(E, F), (E, C)\}$ hacia sus vecinos $\{F, C\}$.
- Determinar el mejor link como el link con el mayor número de *hello packets* recibidos desde el nodo F.
- Suponiendo que (E, F) es el mejor link entonces enviar el mensaje a través de dicho link.
- El nodo F es el destino.

Como se puede ver, usando el protocolo BatMan no se guarda la ruta completa a ningún destino. Cada nodo sobre una ruta solo mantiene información sobre el siguiente link a través del cual se puede encontrar la mejor ruta. Resulta curiosa la similitud que tiene esta filosofía con un comportamiento particular de las termitas llamado *estigmergia* [7]. Las termitas dejan caminos de feromonas que otras termitas

[7] Estigmergia significa colaboración a través del medio físico. Fue introducido por Pierre-Paul Grassé, un estudioso de las hormigas, para explicar cómo se lograban realizar las tareas en insectos sociales sin necesidad de planificación ni de un poder central. Se ha usado como inspiración en algoritmos de ruteo como *AntHocNet* [6]. BatMan también comparte muchas similitudes con su filosofía.

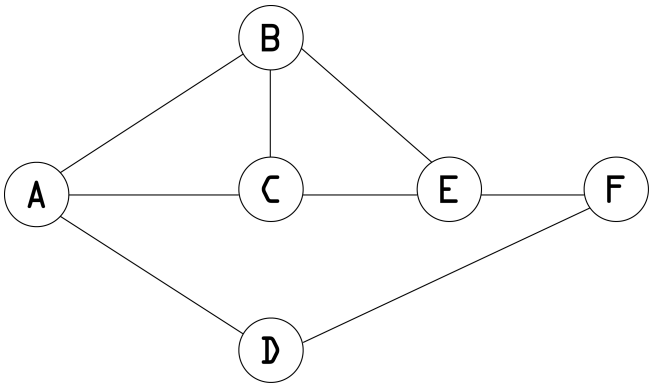

FIGURA 2.2: *Grafo inicial G.*

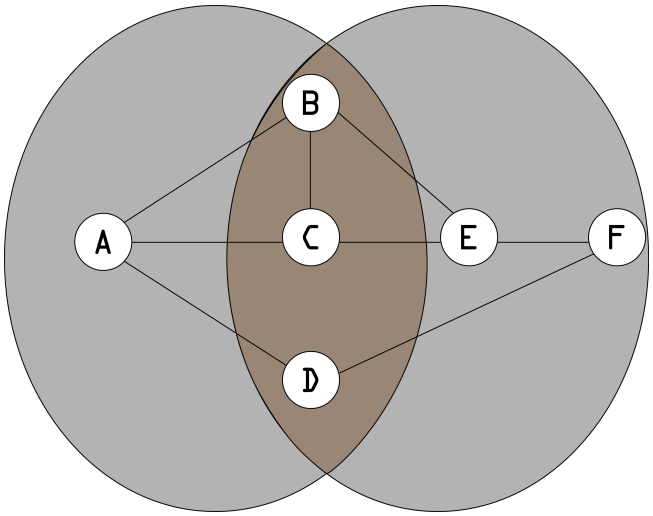

FIGURA 2.3: *Subconjunto de nodos formados por el algoritmo de BatMan luego de la primera iteración. Muestra la relación entre los tres subconjuntos que están referidos en el algoritmo.*

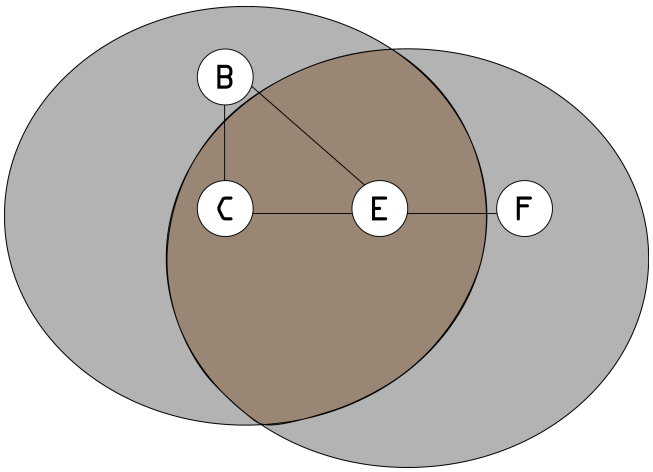

FIGURA 2.4: *Subconjunto de nodos formados por el algoritmo de BatMan luego de la segunda iteración.*

pueden detectar para permitirles saber en qué dirección se puede encontrar la mejor ruta hacia la comida o hacia el nido. Pero las termitas nunca saben el camino completo hacia ninguno de ellos.

BatMan es uno de los protocolos más utilizados en redes mesh y ha sido un punto de partida para diversas investigaciones, no necesariamente limitadas a aplicaciones comunitaras [7].

2.2.4. Protocolo BMX

BMX (o BatMan-eXperimental) es un protocolo de ruteo para redes mesh hecho para sistemas operativos basados en linux. Comenzó siendo una rama del protocolo BatMan diseñada para implementar y probar nuevas características y conceptos y para superar ciertas limitaciones en el algoritmo de ruteo. Pero con el tiempo fue tomando una dirección totalmente diferente de manera que la re-integración se hizo imposible. Hoy en día, es un proyecto independiente que al día de escritura de este trabajo se encuentra en su séptima versión de desarrollo (BMX7).

2.2.5. Protocolo Babel

El protocolo Babel fue creado para reducir al mínimo la cantidad de loops infinitos que se generan en las redes mesh. Conceptualmente, es como el protocolo RIP pero incluye una serie de refinamientos que previenen la formación de loops o al menos hacen que un loop generado desaparezca rápido y no se forme de nuevo [8].

La filosofía de este protocolo radica en saber que en una red mesh siempre van a existir constantes desconexiones y reconexiones. Entonces no tiene sentido hacer un protocolo que converja muy rápido ya que es esto último lo que genera loops. Babel solo acepta actualizaciones provenientes de nodos que sean *factibles*. El concepto de *factibilidad* se ha utilizado en muchos protocolos, teniendo cada uno de ellos su propio método para verificarla.

En el caso de Babel, se considera *factible* a un nodo cuya ruta enviada tenga una métrica [8] estrictamente menor a las que siempre se tuvieron almacenadas. Cada vez que un nodo envía una actualización a otro se guarda el valor de la métrica y se mantiene un registro actualizado de la métrica más pequeña que jamás se haya enviado para esa dirección. Entonces, solo se acepta una ruta nueva si y solo si tiene una métrica mejor que el valor más chico que jamás se haya enviado. Con esto se evitan los loops fácilmente.

En la figura (2.5) se ilustra una topología de red similar a la analizada en 2.2.1 en la que en caso de apagarse el nodo A, podría generarse un loop infinito si se usase el algoritmo vector-distancia sin refinar. En cambio, al usar Babel, en el momento en que A se apaga y C le informa a B de una nueva métrica hacia el nodo A, B *no le cree* ya que esa nueva métrica es mayor a la anterior. Y de esa manera, nunca se origina ningún loop.

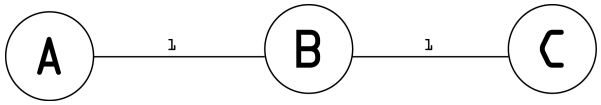

FIGURA 2.5: *Red mesh sencilla que podría generar loops infinitos con RIP pero no con Babel.*

[8] La métrica de una red es el valor que se utiliza para medir la calidad de las rutas. En general, se utiliza el número de hops entre el nodo origen y nodo destino.

Sin embargo, Babel enfrenta un problema llamado *starvation* [9]. Considérese una topología de red como la ilustrada en la figura (2.6) donde todas las métricas son iguales a 1. Supóngase que la conexión entre los nodos A y S se rompe, quedando una nueva topología de red ilustrada en (2.7). A partir de ese momento, B enviará una actualización a A informando una nueva métrica hacia S de valor 2. Como esta nueva métrica es mayor a la anterior, no será aceptada por A, evitando así un loop. Pero de esta manera A y S quedarán incomunicados indefinidamente.

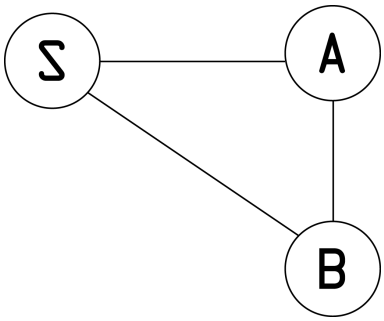

FIGURA 2.6: *Caso inicial de red mesh que utiliza Babel.*

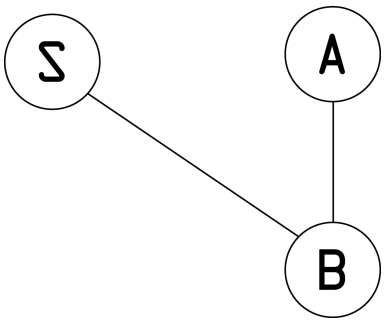

FIGURA 2.7: *Caso final de una red mesh que utiliza Babel. Se puede apreciar la starvation.*

Para evitar la *starvation* se pueden implementar diversas soluciones entre las que se destacan tres:

1. **Apagar toda la red y prenderla desde cero, reiniciando todas las memorias.**

   Esto es inviable en una red mesh ya que las desconexiones y reconexiones ocurren constantemente.

2. **Hacer una sincronización global y olvidar todos los valores previos de *factibilidad* con el nodo cuya conexión de mejor métrica se haya roto.**

   El problema de esta implementación es que el software necesario para sincronizar a los nodos suele ser muy complejo. Además, si bien puede funcionar muy bien en redes cableadas, no es el caso en redes *wireless*.

3. **Usar rutas secuenciadas.**

   Esto consiste en enviar un dato extra además de la métrica en las actualizaciones de los nodos. Esto es lo que implementa Babel y es lo que le da el mejor tiempo de reparación de ruta frente a otros protocolos [4].

[9] El término *starvation* o hambruna surge de la analogía entre una red mesh y un festín. Cuando un nodo tiene información sobre otro nodo pero no puede aceptarla por restricciones del protocolo, se dice que es como estar frente a una mesa llena de comida y no poder probarla.

Normalmente, las actualizaciones que envían los nodos de una red contienen solo la métrica. Usar rutas secuenciadas implica enviar un dato extra en cada *announcement* como sigue:

$$(s, d(B)),$$

donde s es un número de secuencia y $d(B)$ es la métrica. s es un número arbitrario generado únicamente por el nodo que envía el *announcement*. Él es el único que lo puede cambiar mientras el resto solo propaga el *announcement* manteniendo s fijo. De esta manera, se lo puede pensar como un reloj ya que solo es modificado por la fuente de la actualización y además, cada vez que cambia lo hace en forma creciente.

A partir de esto, la función de Babel que define si un nodo es o no *factible* se ve a continuación:

$$\begin{gathered}(s1, d1) \text{ es una ruta mejor que } (s2, d2) \\ \text{si y solo si} \\ s1 > s2 \\ \text{ó} \\ s1 = s2 \text{ y } d1 \leq d2.\end{gathered}$$

Entonces, con esta definición se evita el *starving* en la mayoría de los casos. Para el ejemplo de la figura (2.7) se puede ver que cuando el link entre S y A se rompe, solo hay que esperar a que S incremente su número de secuencia. De esa manera, A aceptará una actualización proveniente de B por más que la nueva métrica sea mayor que la anterior. Para este sencillo ejemplo, esto tomará una o dos iteraciones.

Pero en el caso de redes mucho más grandes en términos de cantidad de nodos, existe una *starvation* temporal ya que el número de secuencia puede tardar muchas iteraciones en llegar. Algunos protocolos como DSDV incrementan s en períodos muy cortos y se dice que eso no suele funcionar muy bien [9].

Babel incorpora la capacidad de enviar un *request*. Cada vez que un nodo sufre de *starvation*, puede enviarle a un vecino un *request* en el que indica su estado. De esta manera, el resto de los nodos sabe que debe incrementar su número s para reparar una ruta rota. Así se decrementa considerablemente el tiempo de reparación de ruta.

Babel es un protocolo "100 % loop-free" en el caso de que haya solo un *gateway*. En el caso de haber más de uno, puede tener problemas en casos muy patológicos como que haya un *crash* en varios de ellos al mismo tiempo. Aún así, el mecanismo de factibilidad lo resuelve relativamente rápido.

2.2.6. Protocolo SSR

Scalable Source Routing (SSR) surge como una solución al problema de escalabilidad que enfrentan otros protocolos [10]. Es un protocolo híbrido, lo cual le hace consumir mucho menos ancho de banda cuando se lleva a cabo el ruteo. Hay dos consideraciones que definen a SSR:

1. Cada nodo mantiene un *cache* que guarda rutas fuentes usadas [10] recientemente. El *cache* se guarda de forma binaria con lo que ocupa poco espacio.

2. Cuando un nodo necesita enviar un paquete a un destino que no tiene guardado, esa ruta se construye iterativamente.

[10] El término "source routing" o ruteo desde la fuente es un sistema en el cual el paquete sale desde el emisor con la "hoja de ruta"que le indica por qué nodos debe pasar para llegar al destino. El primer problema es que como cada una elige su camino, puede ocurrir que muchos seleccionen los mismos tramos y haya sobrecarga o congestionamiento en algunos links y otros permancezcan ociosos.

2.2.7. Comparaciones

En el cuadro (2.1) se comparan los protocolos de ruteo más populares en redes *mesh*. Hay otros protocolos muy utilizados tales como SEAD [11] y SEMTOR [12] pero se decidió no incluirlos en este trabajo ya que no son implementados por ninguna de las iniciativas con las que se trabajó.

	Escalabilidad Logarítmica	**Calidad de Ruta**	**Seguridad**	**Velocidad**
BatMan	No	Provee la mejor ruta	Baja	Alta
BMX7	No	Intermedia	Alta	Alta
Babel	No	Provee la mejor ruta	Intermedia	Alta
SSR	Si	Baja	Alta	Baja

CUADRO 2.1: Comparaciones entre los protocolos más utilizados en redes mesh.

2.3. Criptografía

La criptografía se define como el ámbito de la criptología que se ocupa de las técnicas de cifrado o codificado destinadas a alterar las representaciones lingüísticas de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados. La aparición de la informática y el uso masivo de las comunicaciones digitales han producido un número creciente de problemas de seguridad. Las transacciones que se realizan a través de la red pueden ser interceptadas, y por tanto, la seguridad de esta información debe garantizarse.

Esto le ha dado a la criptografía una importancia muy grande en las últimas décadas, siendo una de las protagonistas principales en el ámbito del blockchain y las criptomonedas. Es por esto último que tiene un apartado propio en este trabajo.

Desde el año 1882 se habla de un método criptográfico *invulnerable* conocido como "libreta de un solo uso" [11]. Conceptualmente, es muy sencilla y se puede implementar con el uso de compuertas XOR. Considérese el caso de dos individuos: El remitente Alice y el receptor Bob, como se muestra en la figura (2.8). Alice desea enviar el mensaje m hacia Bob, donde m es el *string* "CS" que transformado al dominio binario es el número "1010011 1000011". Para encriptar el mensaje este método requiere de una clave k previamente compartida entre todos los participantes de la comunicación que debe ser tan arbitraria como sea posible. Para este ejemplo se supone $k = 11001000100110$. Entonces el mensaje encriptado c [12] es la operación XOR entre m y k como se ve a continuación:

$$c = m \bigoplus k = 10100111000011 \bigoplus 11001000100110 = 01101111100101.$$

Finalmente para que Bob decripte el mensaje solo debe hacer la operación XOR entre c y k. Se verifica entonces que

$$m = c \bigoplus k = 10100111000011.$$

Se ha demostrado que si k es suficientemente arbitrario, entonces este método es *invulnerable*. En la historia, se ha usado el concepto de Libreta de un solo uso

[11] Hay muchos términos que aluden al mismo concepto que "libreta de un solo uso". Entre ellos están "cuaderno", "cifrado de Vernam" y su versión en inglés: "one-time pad".

[12] En criptografía se suele usar la letra c para referirse a un mensaje cifrado.

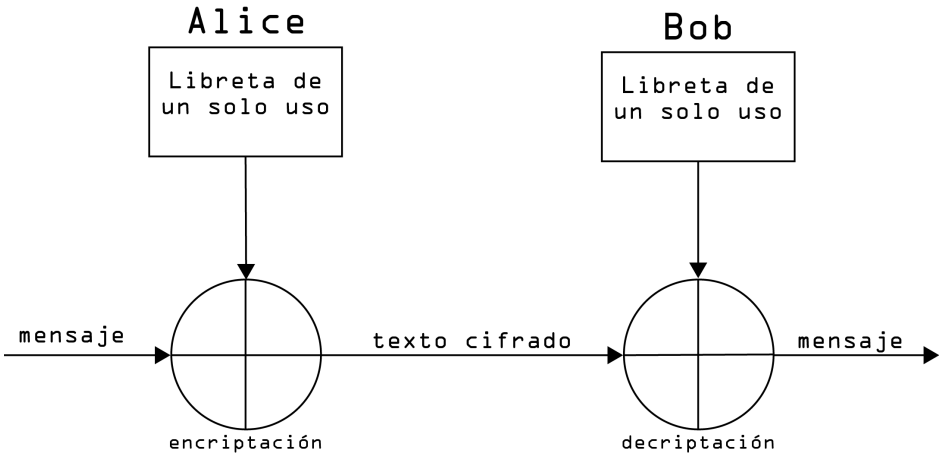

FIGURA 2.8: *Ejemplo de encriptación con el método Libreta de un solo uso.*

en múltiples contextos, como el de la figura (2.9) que se corresponde con la máquina Enigma que usaban los nazis para encriptar sus mensajes en la Segunda Guerra Mundial [13]. El problema de este tipo de encriptación es que generar un k con suficiente aleatoriedad es complejo. Además, por definición k siempre tiene el mismo tamaño (o un tamaño mayor) que el mensaje. Esto es una limitación cuando se desea enviar mensajes relativamente largos.

FIGURA 2.9: *Máquina de encriptación de uso militar "Enigma". Fue usada a finales de los 1930 y durante la Segunda Guerra Mundial.*

Hoy en día, la criptografía enfrenta el desafío de lograr una buena seguridad en la comunicación manteniendo un tamaño de claves relativamente bajo. En las subsecciones 2.3.1 y 2.3.2 se discutirá sobre las criptografías simétricas y no simétricas respectivamente.

[13] La máquina Enigma tenía la desventaja de poseer un k con baja aleatoriedad y por ende, muy predecible. Sin embargo, hizo falta que los aliados capturen tablas de claves y *hardware* para poder descifrar las comunicaciones.

2.3.1. Criptografía Simétrica

La criptografía simétrica se basa en algoritmos que usan la misma clave tanto para encriptar como para desencriptar un mensaje [14] [15]. Las claves pueden ser literalmente idénticas o tener una simple transformación matemática entre ellas. Un sistema que utilice este tipo de encriptación requiere que todas las partes tengan acceso a la clave antes de comenzar la comunicación.

Hay dos tipos de encriptación simétrica:

1. Cifrado por Streaming, con el que se encriptan los dígitos (típicamente bytes) uno a la vez.

2. Cifrado por Bloque, con el que se toman un conjunto de bits y se los encripta como una única unidad.

Independientemente del tipo, es necesario que tanto el remitente como el receptor tengan la misma clave. Esto es un problema ya que es inseguro enviar la clave por un medio virtual pero puede resultar costoso enviarla por un medio físico. Además, la generación de claves se suele hacer con algoritmos de pseudoaletoriedad que a veces otorgan claves predecibles. La criptografía No Simétrica aparece como la solución a estos problemas, como se verá en la sección 2.3.2.

2.3.2. Criptografía No Simétrica

También llamada criptografía de Clave Pública, la criptografía No Simétrica utiliza un par de claves: una clave pública que puede ser difundida ampliamente y una clave privada que es conocida solo por su dueño. Estas claves están relacionadas matemáticamente, con la suficiente complejidad para que no se pueda deducir una a partir de la otra.

En un sistema con este tipo de encriptación, cualquier individuo puede encriptar un mensaje utilizando la clave pública del receptor. Luego, el mensaje solo puede ser desencriptado utilizando la clave privada. Esto se ilustra en la figura (2.10) en la que el individuo B le envía un mensaje al individuo A. Con esto se soluciona el problema que tiene la criptografía Simétrica de tener que recurrir a un medio físico para compartir la clave para garantizar seguridad. Además, la sustentabilidad de este sistema radica en el alto costo computacional requerido para encontrar una clave privada a partir de su clave pública.

Normalmente, estos sistemas se apoyan en algoritmos basados en problemas matemáticos que actualmente no admiten una solución eficiente. Entre ellos se destacan los inherentes a la factorización de enteros, el logaritmo discreto y curvas elípticas. Debido a esta complejidad, las comunicaciones son mucho más lentas que en la criptografía Simétrica. Es por esto, que se suele utilizar solo para pequeños bloques de información como cadenas de texto o para transmitir seguramente una clave de sesión [16].

En un sistema de firma de clave pública, un individuo puede combinar un mensaje con una clave privada para crear una corta firma digital en el mensaje. Cualquier

[14] La "libreta de un solo uso" es de hecho, un tipo de criptografía simétrica que usa como método de encriptación y desencriptación la operación XOR.

[15] Entre los estándares de criptografía simétrica que se utilizan hoy en dia, AES (*Advanced Encryption Standard*) es el más usado debido a su alta seguridad. Incluso se implementa en la transmisión de información clasificada [13].

[16] Una clave de sesión es una clave simétrica de un solo uso utilizada para encriptar todos los mensajes en una sesión de comunicación.

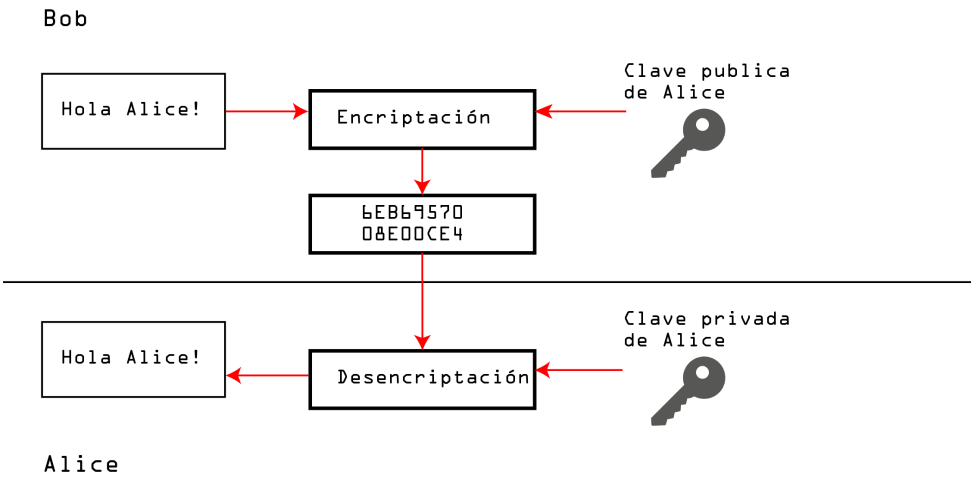

FIGURA 2.10: *Esquema que representa el envío de un mensaje desde el individuo Bob al individuo Alice utilizando un sistema de criptografía No Simétrica.*

otro individuo que posea la correspondiente clave pública puede verificar que efectivamente fue firmado por el dueño de la clave privada. Esto se ilustra en la figura (2.11) en la que un individuo B firma un mensaje y un individuo A lo verifica. Cambiar el mensaje, incluso reemplazar solo un bit, causaría un fallo en la verificación. Esto logra un nivel de autenticidad que es clave en sistemas distribuídos como *blockchain* como se verá en la sección 2.4.

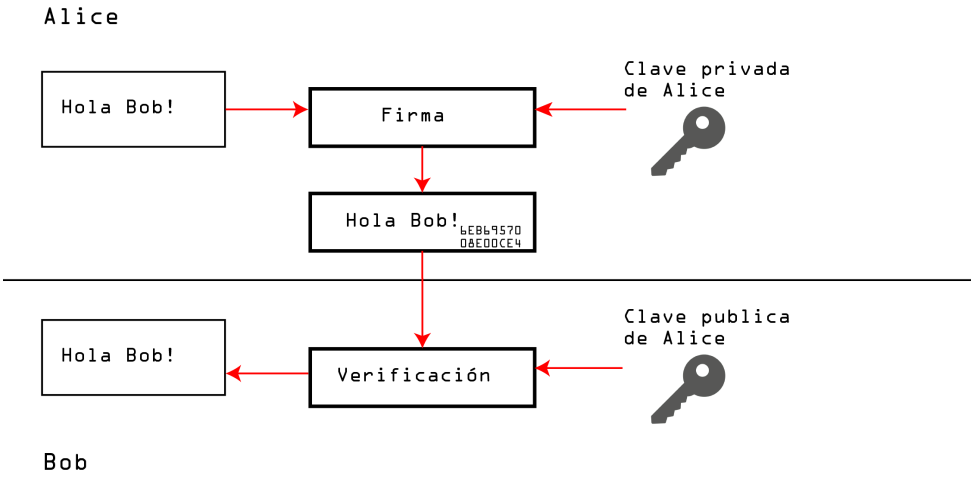

FIGURA 2.11: *Esquema que representa la firma de un mensaje del individuo Bob y su respectiva verificación por parte del individuo Alice utilizando un sistema de firma de clave pública.*

El problema central que enfrenta este tipo de criptografía radica en la dificultad de probar si una clave pública es o no auténtica y si pertenece al individuo que la anuncia y no a una tercera parte maliciosa. En general, esto se soluciona por medio de una infraestructura de clave pública en la que una o varias terceras partes, conocidas como *autoridades certificadoras*, certifican la propiedad de un par de claves.

2.3.3. Funciones Hash

Una función *Hash* es una función que dada una clave genera una dirección (o índice) en una tabla. En computación se ha utilizado para reducir la complejidad en la búsqueda de elementos en arreglos desordenados. Se puede definir como sigue:

$$i = h(o),$$

donde h es la función *Hash* y o es el elemento que se está buscando e i es el índice en la tabla en donde se encuentra ese elemento. Dependiendo del tamaño del arreglo, estas funciones pueden llegar a ser muy complejas de realizar [17] y en ciertas situaciones conviene implementarlas en forma distribuída [15].

Es interesante denotar que para ciertos algoritmos de *Hashing* un pequeño cambio en o puede alterar drásticamente el valor de i. En las figuras (2.12) y (2.13) se muestra el resultado de ejecutar el algoritmo SHA256 a dos *strings* que solo difieren en un punto al final de la oración. Sin embargo, se denota fácilmente que las salidas del algoritmo son muy diferentes. Esto es muy útil ya que dificulta enormemente la capacidad de obtener o a partir de i. Por eso se suele decir que las funciones *Hash* son en un solo sentido.

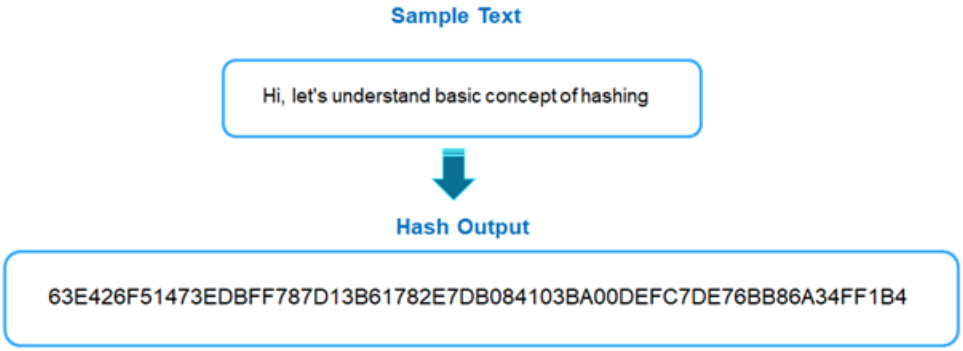

FIGURA 2.12: *Cómputo del algoritmo SHA256 sobre una cadena de caracteres.*

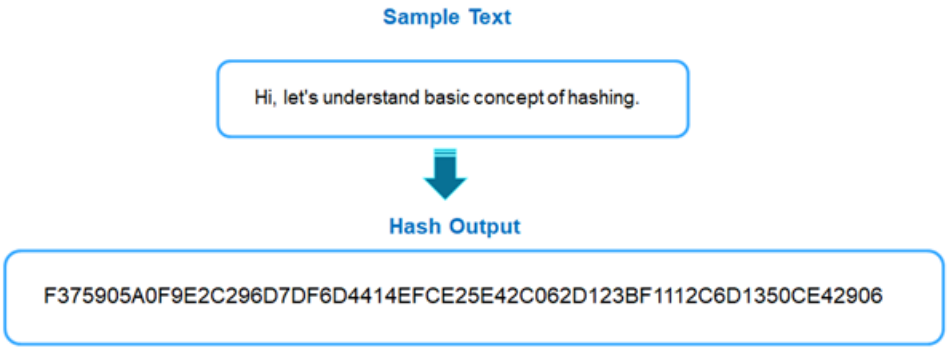

FIGURA 2.13: *Cómputo del algoritmo SHA256 sobre una cadena de caracteres.*

[17] Las funciones *Hash* se basan típicamente en una función que transforma un *string* en un número entero. En ciertos casos puede que dos *strings* distintos devuelvan el mismo índice, lo cual implica una colisión. A lo largo de la historia se han implementado distintas soluciones a ese problema siendo la más común es hacer que la función *Hash* devuelva una lista de índices en vez de un solo índice [14].

Actualmente, los algoritmos de *Hashing* cumplen un rol imprescindible en las redes *Blockchain* [18]. Éstas se basan en la implementación de una lista indexada, como se ve en la figura (2.14). Pero en vez de usar punteros básicos, se usan punteros de tipo *Hash*. Es decir, cada puntero es básicamente una representación *Hash* del bloque de datos anterior.

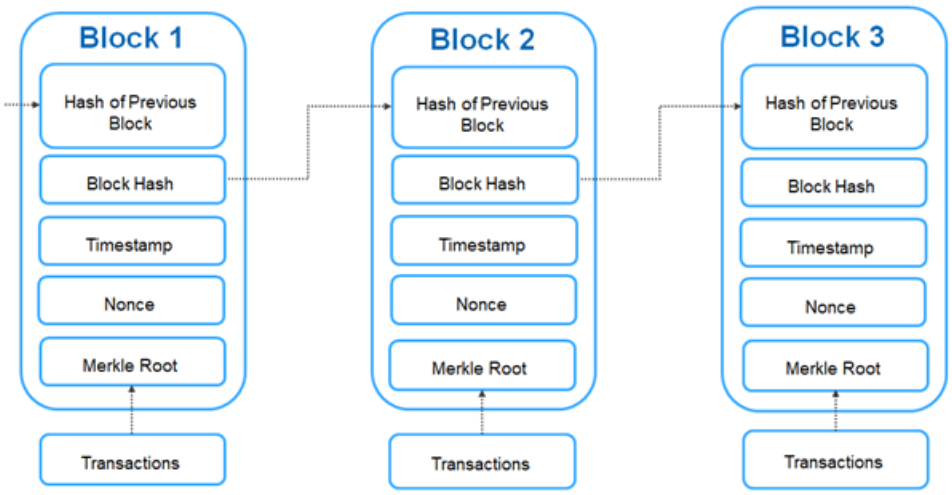

FIGURA 2.14: *Representación esquemática de una Blockchain.*

Como se vio en las figuras (2.12) y (2.13), un pequeño cambio en los datos genera un *Hash* completamente nuevo. Es decir que si los datos del bloque anterior son manipulados entonces se modificará el *Hash* de ese bloque completamente lo cual lleva a *romper* la conexión con el siguiente bloque y por ende con la cadena entera. Este es el punto que hace que las redes *Blockchain* sean tan seguras [17].

2.3.4. RSA

RSA (Riverst-Shamir-Adleman) es uno de los primeros sistemas criptográficos no simétricos y es ampliamente usado para transmisión segura de datos. Como se mencionó en 2.3.1, en este sistema la clave de encriptación es pública y la clave de desencriptación es privada y la sustentabilidad del sistema radica en el alto costo computacional requerido para descifrar una clave a partir de la otra. El principio para lograr esto último se basa en la expresión (2.1).

$$(m^e)^d \equiv m \ (mod \ n) \tag{2.1}$$

Es práctico encontrar tres enteros positivos muy grandes (e, d y n) que cumplan con dicha expresión para cualquier entero m tal que $0 \leq m < n$, donde *mod* es la exponenciación modular [19]. Y luego es extremadamente difícil hallar d a partir de los otros valores, incluso conociendo a todos ellos [20]. La clave pública se representa

[18] Dos de los algoritmos más conocidos son *SHA256* y *Keccak256*, los cuales se utilizan en las *Blockchains* de Bitcoin y Ethereum respectivamente. Ambos entregan una i de un tamaño fijo de 256 bits. El algoritmo *MD5* es otro de los más utilizados, siendo más veloz que los anteriores pero entregando una i de 128 bits [16].

[19] Una exponenciación modular calcula el residuo cuando un número entero positivo b se eleva a la enésima potencia b^e y es dividido por el entero positivo m, llamado módulo.

[20] Se puede demostrar que ningún algoritmo de hoy en día podría usarse para violar la seguridad de RSA en un período de tiempo razonable [18].

con los enteros n y e. La clave privada con el entero d y m representa el mensaje (previamente preparado con alguna técnica en particular).

En la operación de RSA hay cuatro etapas principales. La primera es la generación de claves, la cual respeta el siguiente procedimiento:

- Se eligen dos números primos p y q [21].
- Se computa $n = p * q$, el *módulo* tanto la clave pública como de la privada. Su tamaño en bits es el tamaño de las claves.
- Se calcula $\lambda(n) = Lcm(p-1, q-1)$ donde λ es la función de Carmichael. Esta valor se mantiene guardado de forma privada.
- Se elige un entero e tal que $1 < e < \lambda(n)$ y tal que e y $\lambda(n)$ sean coprimos.
- Se calcula d como $d \equiv e^{-1} \ (mod \ \lambda(n))$.

La segunda etapa consiste en la distribución de clave. Tal y como se ejemplificó en (2.3.2), si Bob quiere enviar información a Alice y deciden usar RSA, Bob debe saber la clave pública de Alice para encriptar el mensaje y Alice debe usar su clave privada para desencriptarlo. Es decir que Alice debe enviarle n y e a Bob mientras que el valor d queda oculto.

Una vez que Bob tiene la clave pública de Alice, puede encriptar un mensaje M y enviárselo. Primero, debe convertir M en un entero m de manera que $0 \leq m \leq n$ usando algún protocolo pre-determinado [22]. Luego computa el texto cifrado c, usando el valor de e de Alice.

$$c \equiv m^e \ (mod \ n). \tag{2.2}$$

Esto puede computarse razonablemente rápido, incluso para números de 500 bits, usando exponenciación modular. Finalmente Bob puede transmitir c a Alice y ésta desencriptarlo. Para ello solo hace falta computar una expresión.

$$m \equiv c^d \ (mod \ n). \tag{2.3}$$

Una vez que tiene m, Alice puede recuperar el mensaje original M aplicando la inversa al protocolo pre-determinado en la comunicación.

RSA puede usarse también para firmar mensajes, es decir para asegurarle al destinatario que el mensaje fue realmente enviado por un determinado remitente. Para ello, y suponiendo que Alice le desea enviar un mensaje firmado a Bob, debe crear una firma siguiendo la expresión:

$$Firma : h(m)^d \ (mod \ n). \tag{2.4}$$

y enviarla junto con el mensaje ,donde h(m) es un *Hash* del mensaje. Al recibir el mensaje, Bob puede obtener el *Hash* del mensaje generado por Alice. Luego, puede computar él mismo el *Hash* del mensaje y verificar que sea igual al que obtuvo. En caso afirmativo, queda verificado que el mensaje fue efectivamente enviado por Alice.

$$h(m) = Firma^e \ (mod \ n). \tag{2.5}$$

[21] Por seguridad p y q deben ser aleatorios, similares en magnitud pero diferir en algunos dígitos para hacer que la factorización sea más difícil.

[22] En general se usa un protocolo conocido como *padding scheme* con el que básicamente se añade información irrelevante al mensaje con el fin de hacerlo más difícil de acceder maliciosamente.

2.3.5. Criptografía de Curvas Elípticas

La criptografía de Curvas Elípticas (ECC) es un tipo de criptografía no simétrica que se basa en la estructura algebraica de curvas finitas sobre campos finitos. Sus etapas de operación son análogas a las presentadas en (2.3.4) pero con diferente fundamentación matemática. Con un tamaño de claves mucho mas pequeño, ECC otorga la misma seguridad que otros mecanismos [23]. Esto se debe a que es mucho más difícil obtener el valor de la clave privada a partir de la pública con respecto a otros enfoques de la criptografía no simétrica [24].

Este mecanismo, además de más seguro, requiere un consumo de CPU mucho menor con respecto a RSA. Sin embargo, en la mayoría de los sistemas criptográficos se implementa RSA en vez de ECC. Esto se debe principalmente a que RSA existe desde mucho antes y está mucho más establecido. También es más fácil de entender y por ende de implementar. Además, algunas operaciones como la verificación de firma, son más veloces que con ECC.

2.4. Blockchain

Hasta hace poco menos de diez años, el comercio en internet había llegado a contar únicamente con instituciones financieras sirviendo como terceras partes de confianza en el proceso de las transacciones electrónicas. Y si bien este sistema funciona satisfactoriamente en la mayoría de los pagos, tiene las desventajas inherentes a un modelo basado en la confianza. Las transacciones completamente irreversibles no son realmente posibles dado que las instituciones financieras no pueden evitar mediar disputas. El costo de la mediación incrementa los costos de la transacción, evitando la posibilidad de que sea rentable hacer transacciones casuales de pequeño valor. Además, con la posibilidad de revertir un pago, los vendedores necesitan más datos de sus clientes por lo que les piden más información de la que realmente necesitarían. Finalmente, en este modelo existe un porcentaje de fraude no omitible que en ciertos casos deja al dinero físico como único medio confiable.

La tecnología Blockchain surgió en el año 2008 con el trabajo de la red Bitcoin como una solución a estos problemas [19]. El primer punto a destacar respecto a esta tecnología es que la cantidad de Bitcoins que uno posee no se almacenan como la cantidad de dinero en una cuenta de débito convencional. Los balances se computan en función de las transacciones, las cuales están encadenadas entre ellas. Si un individuo 1 envía dinero a los individuos 2 y 3, ambas transacciones estarán en la misma cadena y luego, cuánto posea 1 quedará definido por las transacciones en las que se le envió monedas a 1 y no fueron gastadas. Una billetera digital solo coloca esos números ordenadamente para que sea más fácil entender el balance.

Para participar de una red Blockchain es necesario contar con un par de claves pública-privada [25] y con una dirección, la cual se genera luego de aplicar varias funciones Hash a la clave pública. En la figura (2.15) se representa una cadena de bloques de Bitcoin que contiene dos transacciones. Las transacciones pueden tener mas

[23] La criptografía de Curvas Elípticas logra con 256 bits de ancho de clave la misma seguridad que RSA logra con 3072 bits.

[24] Básicamente, para violar la seguridad de ECC hay que computar el logaritmo discreto de una curva elíptica aleatoria para lo cual no existe ningún algoritmo eficiente aún. Además, es una operación más lenta que el logaritmo modular (necesario para hacer ingeniería inversa en RSA).

[25] En la red Bitcoin, el par de claves pública y privada son generadas con un estándard basado en Curvas Elípticas.

de una entrada y mas de una salida pero por simplicidad se tomo de una entrada y una salida solamente.

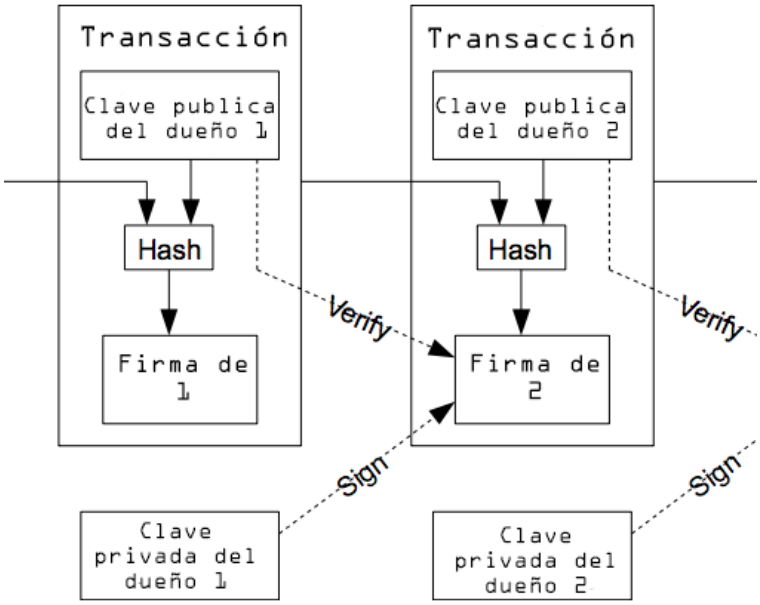

FIGURA 2.15: *Cadena de bloques de dos transaccciones.*

El contenido de cada Tx (es decir, transacción) representa cómo ese Tx está firmado. Supóngase que el dueño 1 quiere generar la transacción de la derecha, es decir hacer un envío de Bitcoins al dueño 2. Utiliza entonces la clave pública del dueño 2 y la Tx previa (la de la izquierda) para generar un *Hash* y firmarlo con su propia clave privada (la clave privada del dueño 1). La razón por la que se genera un *Hash* es porque a medida que crece la cadena, el tiempo que se tarda en firmar es cada vez mayor. Entonces, al hacer un *Hash* se genera una representación de tamaño fijo (y por ende más rápida de computar) de toda la cadena pasada.

La razón por la cual dueño 1 puede gastar las monedas es porque previamente un 'dueño 0' firmó una declaración incluyendo la clave pública de dueño 1 de la misma manera que dueño 1 hizo con dueño 2.

Añadir las direcciones de Bitcoin encima de las claves públicas añade otra capa de seguridad. Si por algún motivo alguien descubre un método para hallar el valor de una clave privada a partir de la clave pública eficientemente y las monedas no fueron transferidas aún, el dinero seguiría estando seguro porque la gente solo ve la dirección a donde el dinero iba a ir, no la clave pública correspondiente.

Además de seguro, Bitcoin evita que se produzca *Double Spending* [26] por medio de una marca de tiempo, como la representada en la figura (2.16). Con ella se verifica cuando un evento ocurrió a partir de una secuencia de caracteres. Cada vez que se realiza una transacción, se genera un *Hash* combinando dicha transacción con todas las transacciones pasadas. Eso hace que sea prácticamente imposible pretender que los eventos ocurrieron en otro orden maliciosamente.

[26]Se le llama *Double Spending* al hecho de añadir dos transacciones muy seguidas una de la otra en tiempo, que partan de la misma salida. Es decir, que una cantidad de dinero sea gastada dos veces antes de que se actualice el balance.

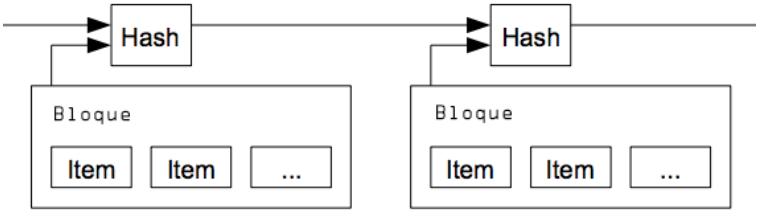

FIGURA 2.16: *Verificación de estados en la Blockchain de Bitcoin.*

En la red de Bitcoin, este proceso se realiza de forma descentralizada [27]. Esto se logra generando un sistema de votaciones entre los participantes de la red para determinar cual es el estado actual y el orden de las transacciones a través de un mecanismo de tipo *PoW* [28]. A aquellos que resuelvan el mecanismo *PoW* en primer lugar, se los recompensa con Bitcoins. Ese incentivo logra por un lado que sea más rentable ser un nodo honesto y ganar Bitcoins ayudando al sistema que intentar hacer actos maliciosos [29].

Bitcoin ha implicado una revolución en los últimos años, tanto tecnológica como económica. Y si bien tiene problemas de escalabilidad que le impiden hoy en día competir con tarjetas de crédito [20], es considerada por muchos expertos la mejor criptomoneda y la que posiblemente sea la más estable cuando se dejen de poner nuevas monedas en circulación. En lo que respecta a este trabajo, comprender los conceptos detrás de Bitcoin y sus ventajas es un punto de partida para entender otra tecnología que se basa en Blockchain.

2.4.1. Ethereum

Ethereum es una plataforma descentralizada en la que se ejecutan *contratos inteligentes* corriendo encima de una red Blockchain [30] desarrollada únicamente con ese propósito. Esto le permite a los desarrolladores crear mercados, mover fondos de acuerdo a instrucciones dadas en el pasado, entre otras cosas. Y todo esto sin necesidad de intermediarios y con la seguridad que la tecnología Blockchain otorga.

Este proyecto se inició en el año 2014 y desde entonces ha sido la base de una inmensa cantidad de iniciativas, muchas de ellas muy exitosas. Hoy en día, cuenta con una comunidad muy extensa y con su propio lenguaje de programación para desarrollar *contratos inteligentes* descentralizados con facilidad. Ethereum es la primer plataforma que introdujo el imprescindible concepto de *Token*.

[27] La descentralización de los tiempos y las verificaciones de las transacciones es lo que hace a Bitcoin tan seguro y tan valuable como moneda.

[28] Un sistema de prueba de trabajo (*Pow*) requiere que el cliente de un servicio realice algún tipo de trabajo que tenga cierto coste y que sea verificado fácilmente en la parte del servidor. Este mecanismo se hizo popular para evitar que se generen cadenas *spam* de mails. En las redes *Blockchain* de Bitcoin, el trabajo a realizar consiste en aplicar una función *Hash* a distintas variantes del header del bloque a analizar hasta que la salida de dicha función entregue una mínima cantidad de 0s [19]. Esto requiere mucho tiempo y mientras la cantidad de nodos honestos de la red sean mayores a la de nodos maliciosos, el sistema es prácticamente *invulnerable*. Además, la dificultad de realizar actos maliciosos crece exponencialmente con cada transacción.

[29] Al proceso de dedicar capacidad de cómputo de un CPU propio para participar en la verificación de estados se lo conoce como *mining* por su similitud con los buscadores de oro en las minas.

[30] Un contrato inteligente es un programa informático que facilita, asegura, hace cumplir y ejecuta acuerdos registrados entre dos o más partes. Este programa vive en un sistema no controlado por ninguna de las partes. Cuando se dispara una condición pre-programada, no sujeta a ningún tipo de valoración humana, el contrato inteligente ejecuta la cláusula contractual correspondiente.

2.4.2. Tokens

Un *Token* es una unidad de valor que permite a su dueño acceder a diferentes servicios que ofrece una plataforma basada en *contratos inteligentes* distribuídos. Existen varios tipos pero las destacadas en este trabajo son las usadas en un modelo conocido como *network medium of exchange token*. En este modelo, los desarrolladores construyen una red capaz de proveer a sus usuarios alguna utilidad novedosa. Esta red es un sistema de economía compartida: consiste puramente de un conjunto de vendedores que proveen servicios con algún protocolo y compradores que los compran donde tanto compradores y vendedores son parte de la comunidad. La única restricción que existe es que los pagos pueden hacerse únicamente con una *Token* determinada lo que le hace a ésta ganar valor. En la primer instancia de este sistema, los desarrolladores ponen las primeras *Tokens* en circulación vendiéndoselas a los primeros clientes interesados. Es así como se suelen financiar estos proyectos.

Para que estos sistemas funcionen en el tiempo y sean escalables, las *Tokens* deben estar en constante circulación entre compradores y vendedores [21]. Además, en algunos casos puede que sea necesario añadir *sinks* [31] para evitar la inflación.

[31] Se le llama *sink* al mecanismo por el cual se quitan monedas de circulación.

Capítulo 3

Objetivos del Proyecto

Este proyecto tiene como objetivo lograr la integración tecnológica entre las iniciativas Libremesh y Althea. Esto implica obtener una versión de firmware de Libremesh qué esté instalada y que funcione de manera estable en al menos un *target* de router [1]. Además, debe contener los paquetes de Althea correspondientes para que pueda ser fácilmente incorporado a una red Althea en donde se efectúen las transacciones por ancho de banda.

En las secciones (3.1) y (3.2) se explican conceptos claves para enteder a las dos tecnologías por separado. Y luego, en la sección (3.3) se explican los pasos trascendentales para llevar a cabo la integración.

3.1. Libremesh

Libremesh es un *framework* modular para crear firmwares basados en OpenWrt y LEDE para nodos wireless mesh. La iniciativa fue iniciada en 2013 por un conjunto de activistas de diferentes culturas y proyectos de diferentes partes del mundo.

La arquitectura de red se basa en dos capas: Capa 2 (en la que se utiliza el protocolo de ruteo batman) y Capa 3 (en la que se utiliza el protocolo BMX7). Por defecto, todos los nodos están corriendo ambos protocolos pero en diferente VLAN [2]. Por lo tanto el ruteo es aislado por la capa MAC [3]. Entonces la red de BMX7 será única para toda la red mesh pero la red de BatMan puede estar dividida en varias nubes. En la figura (3.1) se ilustra una posible topología de conexión de red.

Con esta configuración se pueden aislar las nubes de capa 2. Por ejemplo un edificio de una compañía, un conjunto de departamentos o habitaciones o simples *hotsposts* al nivel de la calle pueden elegir aislar su LAN del resto de la red. Pero al mismo tiempo, pueden conectarse al resto de los nodos usando la red de capa 3. El *roaming* está disponible dentro de la nube, con lo que cualquier sesión TCP, streaming de video o cualquier operación que requiera alto ancho de banda, puede realizarse en movimiento sin problemas. En la figura (3.2) se muestra una topología de conexión en donde se detaca el uso de cada protocolo en cada capa.

3.2. Althea

Althea es una iniciativa de incentived mesh, es decir que busca crear redes mesh hechas por y para la comunidad, en las que los routers se paguen entre ellos por

[1] Se le dice *target device* a un dispositivo que recibe y envía señales, en este caso un modelo de router en particular con un firmware en particular.

[2] VLAN es la abreviación de *Virtual isolated LAN*.

[3] La VLAN de BMX7 es siempre la misma de modo que todos los nodos se ven entre si, mientras que la VLAN de BatMan depende de un identificador de nube.

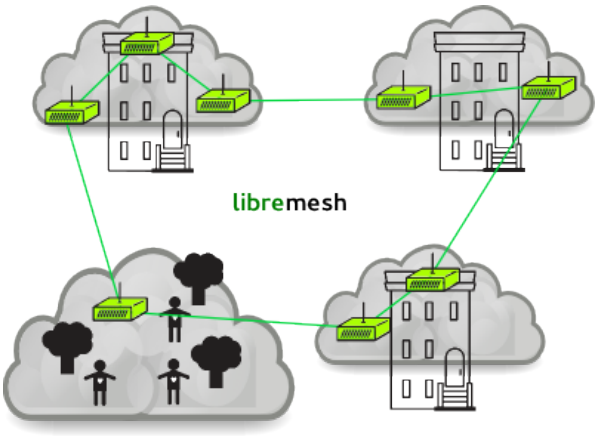

FIGURA 3.1: *Topología de red con Libremesh que ejemplifica las aislaciones entre capa 2 y capa 3.*

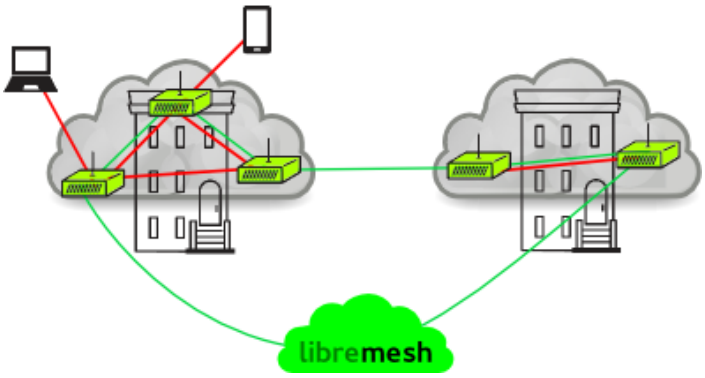

FIGURA 3.2: *Topología de red con Libremesh. En color rojo se muestran las conexiones que usan a BatMan como protocolo de ruteo y en color verde las que usan BMX7.*

el ancho de banda. Esto le permite a la gente tener un ISP descentralizado que sea fácil de administrar y hacer crecer. Cada uno de los nodos de la red cumple un determinado rol (aunque puede haber nodos que cumplan más de uno) y los pagos entre ellos se hacen con criptomonedas en una red blockchain. En la figura (3.3) se esquematiza una red que funciona con Althea.

3.2.1. Red

El protocolo de ruteo que utiliza Althea es Babel, al cual se le agregan dos métricas extras. La primera es una métrica de verificación de calidad de conexión y la segunda es una métrica de precios. Con esta última se comunican los precios de los nodos en la red. Entonces, la elección del router a conectarse dependerá de la relación costo-beneficio entre esas dos métricas.

La arquitectura de red de Althea está compuesta por los siguientes roles lógicos:

- **Nodos de usuario** que son instalados por personas que quieren comprar internet a través de Althea. Se los puede pensar como equivalentes a un router o módem instalado por un ISP tradicional, con la diferencia de que es independiente de cualquier ISP. Estos nodos conectan el tráfico de dispositivos no pertenecientes a Althea a la red de Althea por medio de *hotspots* Wi-fi y/o puertos LAN.

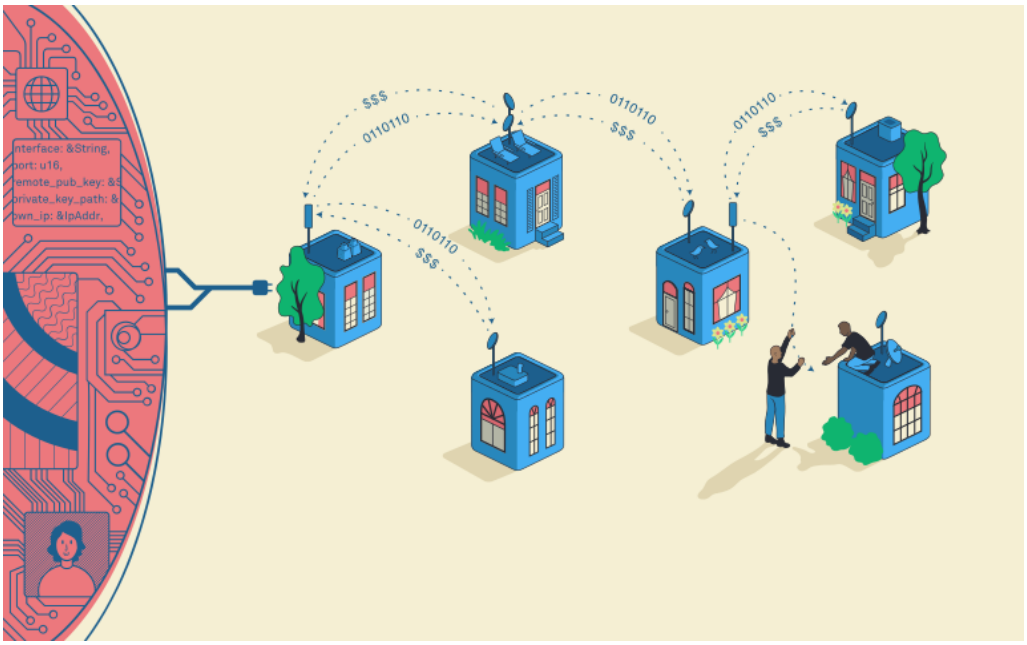

FIGURA 3.3: *Esquema de red Althea.*

- **Nodos intermediarios** que son instalados por gente que quiere ganar dinero al reenviar paquetes. En general son dispositivos más potentes y están ubicados en lugares estratégicos. Cabe destacar que la mayoría de los nodos actúan tanto como nodos de usuario como intermediarios.
- **Nodos de Puerta de Enlace** que son nodos intermediarios pero también están conectados a una fuente de internet barata. Actúan como una conexión de la capa física de Althea a la Internet. Aunque están escudados y no tienen que tomar ningún tipo de responsabilidad legal por el tráfico.
- **Nodos de Salida** que no son necesariamente parte de una red física local sino que pueden estar ubicados en un *datacenter*, por ejemplo. Estos nodos se encargan del rol legal de un ISP con la traducción de direcciones, problemas con copyright, etc. Están conectados con los nodos de puerta de enlace por medio de túneles VPS con lo que les permiten a estos últimos actuar puramente como proveedores de ancho de banda sin tener que lidiar con cuestiones legales.

En la figura (3.4), se esquematizan los roles principales de cada uno de los nodos.

3.2.2. Pagos

Los pagos en una red Althea se efectúan entre los nodos a cambio del reenvío de paquetes. En los inicios de su desarrollo, Althea pretendía utilizar *Tokens* sobre una red Ethereum como medio de pago pero recientemente se ha decidido efectuar los pagos sobre una red blockchain llamada *Cosmos*. Ésta tiene la característica de poder conectar múltiples redes blockchain independientes entre sí, tales como Bitcoin y Ethereum. Los pagos se realizan utilizando la criptomoneda *Dai*, la cual mantendrá su valor relativo al dolar estadounidense, generando de esa manera, un sistema de red económico bastante estable.

Como se utiliza una red blockchain, no hace falta una tercera parte que administre y/o verifique las transacciones. Entonces, los pagos pueden hacerse en porciones

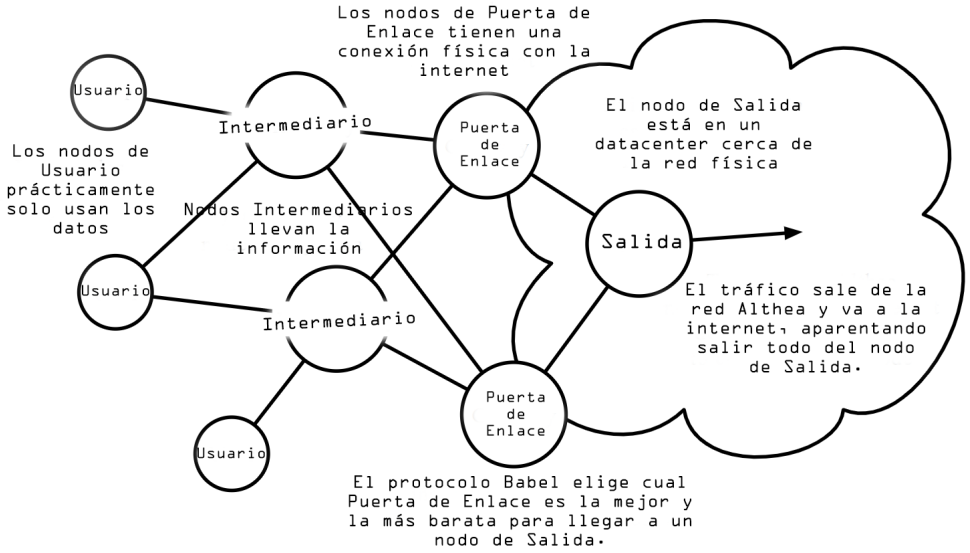

FIGURA 3.4: *Esquema de nodos de red Althea.*

tan pequeñas como sea posible e ir incrementándolas a medida que incrementa la confianza entre los nodos [4].

3.2.3. Software

Althea tiene componentes distribuídos en múltiples repositorios de Github, la mayoría de ellos programados con *Rust*. El de mayor jerarquía es identificado como *althea_rs* y es el encargado de generar al binario más importante, llamado *Rita*. Rita a su vez, produce dos binarios: *client* y *exit*. El primero es el encargado de los nodos mesh generales y el segundo para el server vpn de salida utilizado para proteger al tráfico que sale de la mesh. Estos binarios podrían ser comprimidos e instalados en cualquier firmware como paquetes [5].

3.3. Segmentación de la integración

Lograr la integración tecnológica entre Althea y Libremesh implica una serie de pasos que deben ser segmentados para lograr una organización eficiente en el proceso, identificar los fallos con mayor velocidad y encontrar soluciones correspondientes rápidamente. Los pasos principales son los siguientes:

1. Estudiar y entender el funcionamiento de tanto Althea como Libremesh.
2. Instalar el firmware de Libremesh en un router y verificar su correcto funcionamiento. Si es posible, utilizar más de uno y configurar una red mesh.
3. Instalar los paquetes de Althea en el/los routers.
4. Configurar a los paquetes para que sus servicios correspondientes estén activos y estables todo el tiempo, desde el encendido del router.

[4] De hecho, la red Althea está hecha para que se pague por paquetes, que son la unidad *indivisible* de la información.

[5] Althea ya cuenta con paquetes que contienen al binario *Rita* y a otros de igual importancia listos para instalarse en firmwares basados en *OpenWRT* tales como *Libremesh*.

5. Verificar el funcionamiento correcto de los servicios de Althea.

6. Automatizar la configuración para que en el futuro cualquier usuario pueda instalar y hacer funcionar a los paquetes de Althea en Libremesh de forma rápida y sencilla.

7. Documentar el proceso para facilitar el avance a futuros desarrolladores.

La intención del proyecto es centrarse únicamente en el rol de **Nodos de usuario** de Althea y no en el resto de los roles. Además, de todos los paquetes que ofrece Althea solo es necesaria la instalación de dos de ellos: *Rust_Binaries* y *Babeld*. El paquete *Rust_Binaries* es el que contiene a *Rita* que como se mencionó en (3.2.3), se encarga del funcionamiento de la lógica general de un nodo en una red Althea. Mientras que *Babeld* es el que opera al protocolo de ruteo Babel con las modificaciones pertinentes mencionadas en (3.2.1). Los otros paquetes aportan servicios y procesos que Libremesh ya tiene por defecto, como por ejemplo la capacidad de actualizar automáticamente los paquetes instalados en el firmware. Por lo tanto, no hace falta instalarlos.

Capítulo 4

Trabajo Realizado

4.1. Instalación de Libremesh en routers

Al inicio del proyecto se adquirieron tres routers: Dos modelos *TPLink TL WDR 3600* y uno *TPLink TL WDR 3500*. Son routers que no están más en fabricación pero son muy populares y están disponibles en muchos hogares. Por ende, hay muchos desarrolladores que los usan lo cual hace que haya mucha documentación y foros en internet. Es por esto último que se los eligió para este trabajo. En el cuadro (4.1) se muestran las especificaciones más importantes de estos routers.

Arquitectura	MIPS 74Kc
Vendedor	Qualcomm Atheos
Memoria Flash	8Mb

CUADRO 4.1: Especificaciones compartidas por los modelos TPLink-TL-WDR3500 y TPLink-TL-WDR3600.

La instalación del firmware de Libremesh se efectúa a partir de realizar un *flashing* [1] a partir de una imagen que debe estar construida para el *target* de router al que se vaya a instalar. Libremesh ya cuenta con imágenes compatibles para los dos modelos de routers elegidos en este proyecto.

El proceso de *flashing* fue muy rápido y en cuestión de pocas horas se logró tener a los tres routers con libremesh instalado y funcionando sin problemas. Para verificar el correcto funcionamiento se configuró una red mesh en las instalaciones de la Universidad del Sur. Dicha red tenía una topología representada en la figura (4.1). Los routers habían sido configurados para estar en diferente VLAN y por ende, se veían entre sí por medio del protocolo BMX7.

FIGURA 4.1: *Topología de primer red mesh realizada.*

[1] Se le dice *flashing* al proceso por el cual se instala un nuevo firmware en un dispositivo embebido. Generalmente se realiza conectando dicho dispositivo a otro que es el encargado de proveerle los archivos necesarios, configuraciones, etc.

4.2. Instalación de Paquetes de Althea

En los firmwares basados en OpenWRT, como Libremesh, la instalación de paquetes se suele hacer con un instalador de paquetes llamado opkg. Éste, a partir de unos simples comandos que se le pasan al router por medio de SSH [2] puede instalar cualquier paquete con solo contar con la página web de descarga del paquete. Esta página se puede pasar o bien por medio de comandos o bien modificando archivos de configuración.

En el directorio /etc/opkg del router hay un archivo de configuración llamado customfeeds.conf. Una vez abierto con comandos SSH, se le agregó la siguiente linea de código:

src/gz openwrt_althea
https://updates.altheamesh.com/rc/packages/mips_24kc/althea

Con eso, se le indicó al instalador de paquetes opkg la dirección en la cual hay uno o varios paquetes que podrían ser instalados y que deben agruparse dentro de un grupo llamado openwrt_althea. Nótese que en dicha dirección hay un directorio llamado *mips_24kc*. Althea tiene paquetes compatibles con varios *targets* y es estrictamente necesario que el directorio indicado a opkg sea el adecuado. Luego se ejecutaron los siguientes comandos:

- **opkg update** para actualizar la lista de paquetes instalables [3].
- **opkg install -list** para ver la lista de la lista de paquetes instalables y verificar que los paquetes deseados estuvieran disponibles.
- **opkg install babeld** y **opkg install althea-rust-binaries** para instalar los paquetes deseados [4]. Como se mencionó en el capítulo 3, Althea ofrece más paquetes pero estos son los únicos necesarios.

Cuando se intentó ejecutar los últimos dos comandos, se encontró con un inconveniente. Los routers utilizados tienen solo 8Mb de memoria flash, tal y como se mostró en el cuadro (4.1), 4 de los cuales son empleados por el firmware de Libremesh. Mientras tanto, los paquetes de Althea pesan más de 5 Mb entre los dos. Es decir, hay menos espacio del requerido para instalarlos y por lo tanto, no se pudieron instalar por opkg. Para solucionar este problema de almacenamiento, se tomaron diferentes medidas como se verá en las secciones 4.2.1, 4.2.2 y 4.2.3.

4.2.1. Generación de Firmware propio

Es posible hacer que un conjunto de paquetes esté instalado por defecto en un sistema operativo basado en Linux y que de esa manera, ocupen menos espacio que si se instalaran luego. Para ello es necesario construir el sistema operativo desde

[2] SSH o Secure Shell, es un protocolo de administración remota que permite a los usuarios controlar y modificar sus servidores remotos a través de Internet. Cualquier usuario de Linux puede hacer SSH en su servidor remoto directamente desde la ventana del terminal y luego ejecutar comandos shell de la misma manera que lo haría si estuviera operando físicamente el equipo remoto.

[3] Por defecto, el comando **opkg update** solo actualiza la lista para paquetes que tienen cierta verificación oficial, lo cual no es el caso de los paquetes de Althea. Por lo tanto, hizo falta modificar el archivo etc/opkg.conf borrando la linea *option check_signature 1*. Con eso, se desactivó la verificación por defecto.

[4] Previo a eso, hizo falta instalar paquetes adicionales necesarios para el funcionamiento de Althea (*ca-certificates*, *ca-bundle* y *libustream-openssl*).

cero, configurándolo para que en su construcción se adquieran los paquetes que se deseen. Hay múltiples constructores de sistemas operativos siendo *makefile* [5] la herramienta con la cual se construye Libremesh.

Se configuró una imagen virtual de Ubuntu ya que *makefile* requiere trabajar con Linux y se descargó el *buildroot* [6] necesario para la construcción de Libremesh desde el Github de Librerouter. Se instalaron los paquetes necesarios para el proceso: *gcc*, *binutils*, *bzip2*, *flex*, *python*, *perl*, *make*, *find*, *grep*, *diff*, *unzip*, *gawk*, *getopt*, *subversion*, *libz-dev* y *libc*. Y finalmente, se modificó el archivo feeds.conf.default añadiendo la siguiente línea:

src-git althea https://github.com/althea-mesh/althea-packages.git;master

Con eso, se le indica a *makefile* que en la dirección indicada hay paquetes que podrían ser descargados e instalados en la generación del sistema operativo. Se puede notar que la dirección es diferente a la utilizada anteriormente en opkg. Eso es porque opkg necesita que los paquetes estén comprimidos y listos para instalarse en el *target* mientras que *makefile* necesita solo el código fuente del paquete. A partir de este punto, se continuó con la ejecución de los siguientes comandos en la consola de Linux:

- **./scripts/feeds update -a** para actualizar la lista de paquetes instalables.
- **./scripts/feeds install -a** para descargar los paquetes disponibles y dejarlos listos para incorporarlos a la imagen del sistema operativo.
- **make menuconfig** que inicia una interfaz gráfica para configurar el proceso. Aquí se eligió el *target* correspondiente al router (*atheros ar7xxx/ar9xxx*), se seleccionó qué paquetes se incluirían en el firmware final, etc.
- **make -j1 V=s** para iniciar la herramienta y generar el firmware. Este proceso puede tardar varias horas y es por eso muy importante el uso del argumento *V=s* que permitió ver en tiempo real qué instrucciones estaba ejecutando *makefile* durante la construcción. De esa manera, se pudieron identificar fallos y errores eficientemente.

A partir de este punto, se obtuvieron varios errores en la generación de la imagen. La mayoría de ellos debido a configuraciones incorrectas elegidas con **make menuconfig**, propias de la falta de experiencia con la herramienta *makefile*. El error más complejo de solucionar fue uno relacionado a la auto-configuración de *cargo* [7], la cual no es compatible con los paquetes de Althea. Éstos últimos requieren que haya una variable de entorno definida para que *cargo* pueda efectuar las instrucciones necesarias. Esa variable debe llamarse *RUST_TRIPLE* y debe valer el *Target Triplet* [8] del router. Entonces, se la definió de la siguiente manera:

RUST_TRIPLE = "mips-openwrt-linux-musl"

[5] Make es una herramienta de gestión de dependencias que existen entre los archivos que componen el código fuente de un programa, para dirigir su recompilación o "generación" automáticamente.

[6] Se le dice *buildroot* a la carpeta que contiene los archivos necesarios para realizar la construcción de un sistema operativo con *makefile*.

[7] El paquete *cargo* es el manejador de paquetes de *rust*, el lenguaje de programación con el que está programado Rita.

[8] El *Target Triplet* es una cadena de caracteres que describe la plataforma en la cual corre un determinado código. Está formado por al menos tres partes: el nombre de la familia de CPU del dispositivo, el vendedor y el sistema operativo base.

Cabe destacar que por defecto Linux no tiene disponible ese valor de *Target Triplet*, por lo que hubo que instalarlo con el comando **rustup target add mips-unknown-linux-musl** en la consola de comandos de Linux.

Finalmente, la construcción de la imagen de Libremesh con los paquetes de Althea instalados dentro pudo ser completada satisfactoriamente, pesando al final 7.9 Mb con lo que entrarían perfectamente en la memoria flash de los dispositivos disponibles. Entonces, se prosiguió a hacer *flashing* de los routers aunque desafortunadamente y debido a que el *buildroot* de Libremesh está en fase de desarrollo, no se los pudo hacer funcionar ni comunicarse a ellos de ninguna manera. Así que hubo que recuperarlos con comandos TFTP y probar otra solución al problema del espacio de almacenamiento como se verá en la sección 4.2.2.

4.2.2. Uso de disco externo USB

La instalación de un paquete implica descomprimirlo, luego instalarlo, configurarlo y finalmente borrar los archivos basura que quedaron en el proceso. Es posible utilizar un disco USB que se encargue de la descompresión e instalar el paquete desde allí y así usar menos espacio del router. También es posible utilizar un disco USB como si fuese parte del router e instalar los paquetes directamente allí. Se probaron ambas metodologías.

Para el primer caso, se instalaron los paquetes necesarios para el manejo de discos USB: *block-mount*, *kmod-fs-ext4*, *kmod-usb-storage*, *e2fsprogs*, *kmod-usb-ohci*, *kmod-usb-uhci*, *fdisk*. Y luego, se corrieron los siguientes comandos por SSH en el router:

- **mkfs.ext4 /dev/sda** donde *sda* es el nombre del dispositivo USB. Para ver los nombres de todos los dispositivos USB se puede usar el comando **ls -al /dev/sd*** y así identificar cual es el que se quiere utilizar.
- **block detect | uci import fstab**
- **uci set fstab.mount[0].enabled='1' && uci set fstab.global[0].check_fs='1' && uci commit** con lo que se logra hacer que el dispositivo sea reconocido desde el inicio del router.
- **/sbin/block mount && service fstab enable**

Y luego de reiniciar el router se prosiguió a la instalación de los paquetes por opkg. El disco externo, detectado por el router sin problemas se encargó de la descompresión de los paquetes pero aún así no pudo concretarse la instalación por falta de espacio en el router. La instalación de paquetes necesarios para el manejo de dispositivos USB había ocupado prácticamente toda la memoria flash que quedaba. Con lo cual, la única alternativa fue intentar la segunda metodología.

La segunda forma de usar un disco USB no es muy eficiente ya que requiere dejar conectado el disco de manera indefinida pero se la consideró igualmente dado que las otras alternativas no habían funcionado. Para llevarla a cabo, se corrieron los siguientes comandos:

- **mkfs.ext4 /dev/sda** donde *sda* es el nombre del dispositivo USB.
- **block detect >/etc/config/fstab**
- **block mount**
- **mount /dev/sda /overlay**

- **mount**

Con eso se logró que uno de los routers tuviera 4Gb de espacio en el disco y se le pudieran instalar los paquetes por opkg. El problema es que una vez instalados, los paquetes deben configurarse y los archivos de configuración están por defecto dentro de la memoria flash del router y no dentro del disco externo. La memoria flash no tenía espacio suficiente para que se modifiquen los archivos de configuración y por ende, no se pudo finalizar la instalación de ninguno de los paquetes.

La única forma de hacer que los archivos de configuración estén alojados en la memoria externa, es necesario hacer *flashing* al router con una imagen nueva que ya incluya el manejo de discos USB por defecto. Y dados los problemas encontrados con el *builroot*, mencionados en (4.2.1), se decidió dejar de lado el uso de discos USB y se intentó proseguir con la integración de Althea y Libremesh por medio de un router virtual, como se explica en la siguiente sección.

4.2.3. Implementación con máquinas virtuales

Según los desarrolladores de Althea, sus paquetes deberían funcionar perfectamente en dispositivos que tengan al menos 16Mb de memoria flash [9]. Afortunadamente, es posible dedicar tanto espacio como sea necesario cuando se trabaja con imágenes virtuales. Con lo cual, no habría ninguna limitación.

Se decidió utilizar el software *VirtualBox* para la generación de un router virtual con 512 Mb de espacio de almacenamiento, para instalar los paquetes y proseguir con la integración. Esta herramienta crea sus imágenes a partir de archivos de extensión *.vdi* mientras que las imágenes de libremesh para el nuevo *target* correspondiente [10] están en formato *.img*. Se convirtió las imágenes de un formato al otro por medio del siguiente comando desde la consola de Linux:

VBoxManage convertdd A.img B.vdi,

con lo que se convirtió de la imagen A (previamente descargada desde la página de Libremesh) a la imagen B. Posteriormente, se montó la imagen en *VirtualBox* y se configuró una topología de red sencilla como la que se muestra en la figura (4.2).

Se accedió entonces al router por medio de SSH y con opkg se instalaron los paquetes de Althea satisfactoriamente, esta vez no hubo ningún problema de almacenamiento. Cabe mencionar que fue necesario actualizar la lista de *feeds* y correr varios comandos previos a la instalación tal y como se mencionó en (4.2). Luego de la instalación, se pudo proseguir con la configuración necesaria para el funcionamiento.

4.3. Configuración de los paquetes

Para tener el servicio *Rita* corriendo de forma estable, es necesario tener primero al servicio *Babeld* corriendo y luego modificar adecuadamente el archivo de configuración *Rita.toml*. Para lograr lo primero, basta con correr el siguiente comando por SSH:

[9] Se llegó a esta consideración recién luego de meses de desarrollo. Es decir, al comienzo de este proyecto se desconocía sobre la limitación de espacio y por eso se optó por adquirir los modelos mencionados.

[10] Como se trabajó con *VirtualBox*, la arquitectura debió pasar de ser *atheros ar7xxx/ar9xxx* a ser *x86_64*.

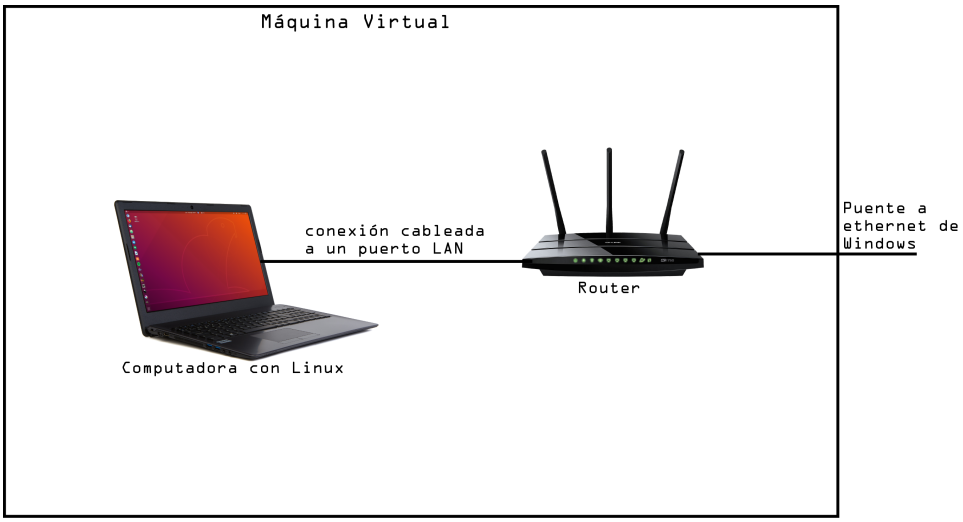

FIGURA 4.2: *Topología de red virtual con la que se trabajó.*

service babeld start

Normalmente, es necesario abrir un puerto en el firewall del router para que *babeld* opere a través de él. Sin embargo, en la fecha de escritura la última versión de Libremesh tiene todos los puertos abiertos por defecto. Con lo cual, no hizo falta hacer ningún cambio en el firewall. Con el comando **netstat -tunapl** se puede verificar qué servicios están activos y qué puertos se están utilizando. Se pudo verificar así que *Babeld* usaba el puerto 53 sin problemas, incluso luego de reiniciar el router. Es posible cambiar el puerto a utilizar por medio de **babeld -p P**, donde P es el puerto deseado. Pero como con el puerto 53 no había problemas, se decidió dejarlo así.

Entonces, el siguiente paso fue modificar el archivo *Rita.toml* adecuadamente. Hay que destacar que dicho archivo no se instaló automáticamente con la descarga del paquete, debido a problemas internos del desarrollo de Althea. Así que fue provisto por mail, por parte de Justin Kilpatrick, desarrollador de hardware de Althea [11]. El archivo cuenta con una serie de variables, cada una con sus valores que determinan los tiempos de ejecución del servicio *Rita*, los puertos por los que opera, si debe iniciarse en el arranque del dispositivo o no, etc. El cambio más importante para el funcionamiento fue asignarle el valor 53 a la variable *babeld port* que por defecto estaba en 6872. Además, se definieron las *peer_interfaces* correspondientes a Libremesh como se ve a continuación:

```
babel_port = 53
rita_hello_port = 4876
rita_contact_port = 4874
rita_dashboard_port = 4877
peer_interfaces = ["eth1", "br-lan", "anygw", "dummy0"]
manual_peers = ["test.althea.org", "exit.altheamesh.com"]
eth_address = "0x0101010101010101010101010101010101010101"
```

[11] Entre otras cosas, el archivo *Rita.toml* contiene la clave privada utilizada para las transacciones monetarias entre los routers. Y en la fecha de escritura de este trabajo, los mecanismos de actualización de los paquetes de Althea se basan en la sobre-escritura de todos los archivos. Es decir, que se podría perder todo el dinero almacenado al actualizar los paquetes. Es por eso que momentáneamente se quitó el archivo de la instalación y tuvo que ser provisto por mail.

Finalmente, se ejecutó el comando **service rita start** y se reinició el router. Se pudo comprobar rápidamente con **top** y **netstat -tunapl** que *Rita* estaba activo y buscando conectarse con otros routers por medio del puerto 53, con el protocolo *Babel*.

El siguiente paso en la integración, fue automatizar el proceso para que futuros desarrolladores y usuarios puedan obtener los mismos resultados obtenidos en mucho menor tiempo.

4.4. Automatización

En los repositorios de Althea, hay un archivo llamado *Rita.toml.j2* que se encarga de construir al archivo *Rita.toml* en el momento de la instalación del paquete. Básicamente esta compuesto de secuencias de control que verifican en qué *target* se está instalando Rita y por ende, qué forma debe obtener el archivo de configuración. En este proyecto, se propuso la modificación de dicho archivo agregando la siguiente estructura:

```
{% if rust_target = x86_64-unknown-linux-musl %}
babel_port = 53    #Puerto asignado por defecto en Libremesh
{% else %}
babel_port = 6872 #Puerto por defecto en otros dispositivos
```

4.5. Documentación

En este trabajo no se llegó a generar una red Althea con un número significativo de nodos con lo cual se decidió postergar la documentación de los códigos propuestos hasta tener eso resuelto. Además, en la última video-conferencia realizada con uno de los desarrolladores de Althea se concluyó que el puerto con el que trabaje *Babel* debería ser del 1000 en adelante. Eso es porque en general, los puertos de menor valor se usan para más de un servicio a la vez y por ende, usar el puerto 53 podría causar interferencias en las transacciones.

Entonces se intentó cambiar el puerto de *Babel* con el comando **babeld -p P** con varios valores de P mayores a 1000 y en todos los casos Libremesh terminaba forzando a *Babel* a operar por el puerto 53. Por una cuestión de tiempos académicos, se decidió dejar pendiente la solución de este problema para el futuro.

Capítulo 5

Conclusiones

5.1. Con respecto a la integración de Althea y Libremesh

Actualmente, las dos iniciativas con las que se trabajó transitan etapas de desarrollo y crecimiento, siendo Althea la que está más en sus inicios. Este proyecto marca la dirección a seguir para lograr la integración completa entre ellos. Si se continúa trabajando de forma sostenida, puede estar lista para antes de que Althea haga disponible su servicio para todo el mundo, lo cual sería muy conveniente considerando la cantidad de dispositivos con los que Libremesh es compatible.

Los pasos a futuro son los siguientes:

- **Configurar al protocolo babel en un puerto mayor al 1000.** Como se mencionó en el capítulo 4, el puerto 53 podría estar compartido con varios servicios a la vez. Eso podría resultar catastrófico en ciertos escenarios en los que se interrumpa una transacción, por ejemplo. Por lo tanto este paso es obligatorio.
- **Testear la integración en una topología virtual que tenga al menos 4 nodos.** Esto no llego a implementarse por cuestión de tiempos académicos, pero es igual de importante que el paso anterior.
- **Construir una imágen funcional de Libremesh con los paquetes de Althea dentro.** De esta manera, se lograría la integración con dispositivos de 8Mb de tamaño de memoria flash. Si bien no es el objetivo principal de Althea hacer su software compatible con routers de menos de 16Mb de memoria flash, con esto podría llegar a muchos más usuarios y no requerir que éstos deban hacer cambios de equipos.
- **Modificar la interfáz gráfica de Libremesh de modo que Althea pueda ser controlado por una UI.** Con esto sería mucho más simple configurar e interpretar el funcionamiento de la red Althea sin necesidad de usar comunicación SSH y por ende, de contar con conocimientos de redes.

5.2. Con respecto al futuro de Incentived Mesh

Contar con una red wi-fi global a la que usuarios de todo el mundo puedan conectarse es algo cada vez mas necesario y buscado por muchas empresas. Hay muchas iniciativas populares tales como *Google Loon* y *Starlink* en las que se busca contar con *Access Points* en zonas estratégicas por medio de globos aeroestáticos y satélites de órbita baja respectivamente. El problema con estos proyectos es el alto costo de inversión y la dificultad de generar ganancias a largo plazo a partir de ellos. Esto genera un gran panorama para el *incentived mesh*, que aparece como una solución totalmente accesible que prácticamente no requiere inversión.

Sin embargo, también se ve como una amezana para los ISPs, que podrían tomar medidas de todo tipo con tal de evitar o retrasar el cambio. Considerando que en general los ISPs son monopolios en las zonas que cubren, es muy probable que tengan éxito, al menos durante los próximos años. Otro factor en contra del *incentived mesh* es que basa su sistema de transacciones en Blockchain, que a pesar de ser una tecnología maravillosa es percibida con mucho esceptisismo. Esto se debe al incontable número de criptomonedas maliciosas creadas en el 2017 y 2018 con las que se estafó a miles de personas. Por lo tanto va a llevar un tiempo ganar la confianza de la gente, la cual es crucial para que estas iniciativas prosperen.

De todas maneras, en el transcurso de la historia han existido muchos avances tecnológicos que implicaron cambios de paradigmas y el *incentived mesh* podría ser uno de ellos. Es seguro que no ocurrirá de forma abrupta, sino que llevará un tiempo considerable y probablemente se logre sistemáticamente con pequeños avances. En las ciudades más pobladas del mundo por ejemplo, es cada día más usual el acceso a redes por medio de ISPs descentralizados por un costo mensual bajo. Ésto podría ser uno de esos pasos intermedios.

5.3. Con respecto al aprendizaje de este proyecto

Este proyecto implicó el estudio de múltiples tecnologías y conceptos no estudiados previamente durante la carrera universitaria. Entre ellos se destacan los protocolos de redes mesh, técnicas de encriptación y herramientas como *git*, *makefile*, *bash*, etc. Sin embargo, el mayor aprendizaje provino de trabajar en conjunto con dos iniciativas del mundo real, con todo lo que ello implica: comunicación, organización, etc. Esto último implicó un crecimiento como estudiante y futuro profesional muy importante.

Bibliografía

[1] IEEE. «IEEE 802.11s». En: *IEEE Standars Association* (sep. de 2011). URL: https://es.wikipedia.org/wiki/IEEE_802.11s.

[2] Pantelis A. Frangoudis y George C. Polyzos. «Wireless Community Networks: An Alternative Approach for Nomadic Broadband Network Access». En: *IEEE* (mayo de 2011). URL: https://www.cs.columbia.edu/~vpk/papers/wcn.commag11.pdf.

[3] Federal Communications Comission. «Internet Access Services». En: *Industry Analysis and Technology Division Wireline Competition Bureau* (jun. de 2016). URL: https://www.fcc.gov/general/iatd-data-statistical-reports.

[4] J. C.-P. Wang M. Abolhasan B. Hagelstein. «Real-World Performance of Current Proactive Multi-hop Mesh Protocols». En: *IEEE APCC* (oct. de 2009).

[5] Corinna Aichele David Johnson Ntsibane Ntlatlapa. «A simple pragmatic approach to mesh routing using BATMAN». En: *2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries* (oct. de 2008).

[6] F. Ducatelle y L.M. Gambardella G. Di Caro. «AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks». En: *Proceedings of Parallel Problem Solving from Nature (PPSN VIII)* 3242 (jul. de 2004), págs. 461-470.

[7] Matthew Britton y Andrew Coyle. «Performance analysis of the B.A.T.M.A.N. wireless ad-hoc network routing protocol with mobility and directional antennas». En: *IEEE* (nov. de 2011). URL: https://ieeexplore.ieee.org/document/6470393.

[8] J. Chroboczek. «The Babel Routing Protocol». En: *Independent Submission* (abr. de 2011). URL: https://tools.ietf.org/html/rfc6126.

[9] Juliusz Chroboczek. «Babel Doesn't Care». En: *Conferencia BattleMesh* 8 (ago. de 2015). URL: https://youtu.be/1zMDLVln3XM.

[10] Thomas Fuhrmann y col. «Pushing Chord into the Underlay: Scalable Routing for Hybrid MANETs». En: *Fakultät für Informatik, Universität Karlsruhe* (jun. de 2006). URL: http://i30www.ira.uka.de/research/publications/p2p/.

[11] Yih-Chun Hu, David B. Johnson y Adrian Perrig. «SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks». En: *Ad Hoc Networks* 1 (jun. de 2003). URL: http://computerscienceweb.com.

[12] Axel Neumann y col. «Securely-Entrusted Multi-Topology Routing for Community Networks». En: *Universitat Politècnica de Catalunya, Barcelona, España* (jun. de 2016).

[13] Lily Chen. «Cryptographic Standards and Guidelines». En: *National Institute of Standards and Technology* (dic. de 2016). URL: https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development.

[14] Victor S.Adamchik. «Concept of Hashing». En: *CMU* (oct. de 2018). URL: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Hashing/hashing.html.

[15] Petar Maymounkov y David Mazières. «Kademlia: A Peer-to-peer Information System Based on the XOR Metric». En: *Universidad de New York* (mar. de 2002). URL: http://kademlia.scs.cs.nyu.edu.

[16] R. Rivest. «The MD5 Message-Digest Algorithm». En: *MIT Laboratory for Computer Science* (abr. de 1992).

[17] Susmit Sil. «Concept of Hashing». En: *Medium Corporation* (jun. de 2018). URL: https://medium.com/@susmitsil/concept-of-hashing-a8fd97954569.

[18] Punita Meelu y Sitender Malik. «RSA and its Correctness through Modular Arithmetic». En: *AIP Conference Proceedings* (dic. de 2010). URL: https://aip.scitation.org/doi/pdf/10.1063/1.3526259.

[19] Satoshi Nakamoto. «Bitcoin: A Peer-to-Peer Electronic Cash System». En: *Bitcoin Webpage* (nov. de 2008). URL: www.bitcoin.org.

[20] Joseph Poon y Thaddeus Dryja. «The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments». En: *Lightning Network* (ene. de 2016). URL: https://lightning.network/lightning-network-paper.pdf.

[21] Vitalik Buterin. «On Medium-of-Exchange Token Valuations». En: *Vitalik Buterin's Website* (oct. de 2018).