GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

ANÁLISIS DE NETWORK CODING Y ROUTING EN COMUNICACIÓN DE DATOS: CASUÍSTICAS EN MÚLTIPLES ESCENARIOS

Autor: Fernando Siljestrom Berenguer

Director: Kevin A. Tang

Madrid

Julio 2024

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Análisis de Network Coding y Routing en Comunicación de Datos: Casuísticas en Múltiples Escenarios

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2023/2024 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

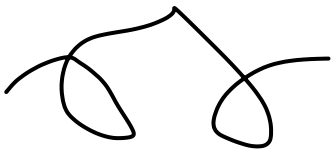
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Fernando Siljestrom Berenguer Fecha: 02/07/2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Kevin A. Tang Fecha: 03/07/2024



GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

ANÁLISIS DE NETWORK CODING Y ROUTING EN COMUNICACIÓN DE DATOS: CASUÍSTICAS EN MÚLTIPLES ESCENARIOS

Autor: Fernando Siljestrom Berenguer

Director: Kevin A. Tang

Madrid

Junio 2024

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a Cornell University por brindarme la oportunidad de ser parte de su comunidad académica. Desde el personal académico hasta mis compañeros, la calidad del trato que recibí durante mi estancia fue excepcional. En particular, me gustaría agradecer al profesor Kevin A. Tang por su valiosa ayuda en el desarrollo de este estudio. Su orientación y asesoramiento profesional fueron fundamentales para alcanzar los objetivos propuestos.

Esta experiencia en Cornell University ha sido verdaderamente única y ha enriquecido mi trayectoria académica y personal de una manera que nunca hubiera imaginado. También quisiera agradecer a la universidad por brindarme un ambiente estimulante y enriquecedor que fomentó mi crecimiento profesional y personal. El compromiso de la institución con la excelencia académica y de investigación es inspirador y me siento afortunado de haber tenido la oportunidad de ser parte de esta comunidad.

Mi reconocimiento especial va para el profesor Kevin A. Tang. Su dedicación y generosidad contribuyeron enormemente a mi desarrollo académico. Su orientación y apoyo continuo han sido un pilar de este proyecto y estoy profundamente agradecido por el impacto positivo que ha tenido en mi experiencia en Cornell University.

ANÁLISIS DE NETWORK CODING Y ROUTING EN COMUNICACIÓN DE DATOS: CASUÍSTICAS EN MÚLTIPLES ESCENARIOS

Autor: Siljestrom Berenguer, Fernando.
Director: A. Tang, Kevin.
Entidad Colaboradora: Cornell University.

RESUMEN DEL PROYECTO

La capacidad de transmisión de información dentro de una red de datos está limitada por la topología de la red y la capacidad de conexión. Las técnicas tradicionales para mejorar el rendimiento de la transmisión se centran en controlar estratégicamente el flujo de información a lo largo de un gran ancho de banda o múltiples rutas desde el origen al destino.

Hoy en día, sabemos que una estrategia de enrutamiento de este tipo por sí sola puede no ser suficiente. Más bien, se debe considerar la codificación/decodificación de datos en los nodos de la red para lograr un rendimiento óptimo. A esto se llama codificación de red o *Network Coding.* Se ha demostrado que la codificación de red para redes de reenvío único de sesión única y redes de transmisión de sesión única tiene el mismo rendimiento que el enrutamiento tradicional [9]. Para redes multicast de sesión única no dirigidas, el beneficio que brinda la codificación de red en términos de velocidad de transmisión está limitada a 2, pero para redes dirigidas puede ser infinita.

Sin embargo, para sesiones múltiples de envío único no dirigido donde se permite el enrutamiento fraccionado, todavía tenemos que encontrar un caso en el que la codificación de red pueda proporcionar beneficio. En este proyecto, revisamos la documentación y la literatura sobre codificación de redes y originamos cálculos y programas propuestos en investigaciones anteriores, que pueden dar respuesta a las cuestiones anteriormente mencionadas.

Este proyecto tiene dos objetivos:

- En primer lugar, probar la conjetura aún válida de que el cifrado de red no tiene ninguna ventaja sobre el enrutamiento multisesión estándar de sesión única en redes no dirigidas.
- En segundo lugar, elaborar un algoritmo de aproximación partiendo de un algoritmo de cifrado de red (cFlow), aproximado a un algoritmo de routing para evitar complejidad (Steiner Tree Packing).

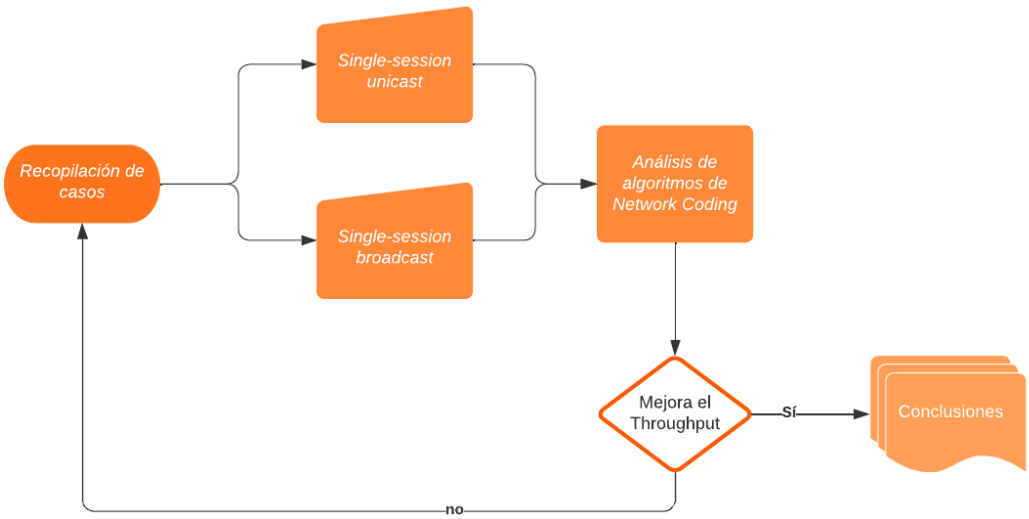


Ilustración 1: Esquema del modelo para el primer problema. Fuente: Elaboración propia

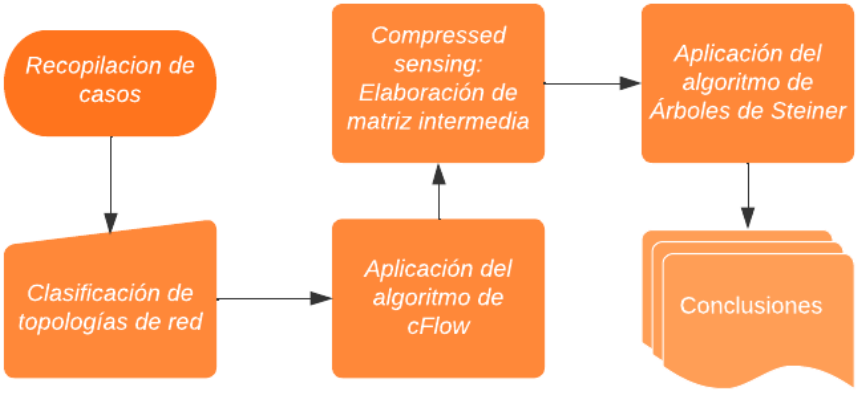


Ilustración 2: Esquema del modelo para el segundo problema. Fuente: Elaboración propia

Para intentar cumplir con los dos objetivos propuestos, realizamos una revisión exhaustiva de los algoritmos modernos para la comunicación de datos, centrándonos tanto en el enrutamiento como en la codificación de red. En cuanto al enrutamiento, se prestó especial atención no solo al algoritmo del árbol de Steiner, sino también al enrutamiento fraccional y al enrutamiento fraccional arbitrario. Estos algoritmos son fundamentales para optimizar la eficiencia de la transmisión de datos en las redes, ya que son capaces de encontrar la ruta

más eficiente para la transmisión de información, teniendo en cuenta aspectos como el ancho de banda disponible y la capacidad de conexión. Al mismo tiempo, se han estudiado en detalle los algoritmos de codificación de redes, entre los que destacan cFlow, oFlow y mFlow. Estos algoritmos desempeñan un papel importante en la mejora del rendimiento de la red al permitir la codificación y decodificación eficiente de datos en los nodos de la red. El análisis de estos algoritmos proporciona una comprensión profunda de cómo la codificación de red puede aumentar la capacidad de transmisión y mejorar la tolerancia a fallas en diferentes configuraciones de red.

La razón principal para el desarrollo del algoritmo descrito en la Ilustración 2, es la complejidad inherente de los algoritmos existentes en la comunicación de datos. Por un lado, los algoritmos de árbol de Steiner tienen complejidad NP-Hard, lo que significa que encontrar soluciones óptimas a tales problemas es computacionalmente costoso y, en muchos casos, poco práctico en un período de tiempo razonable. Por otro lado, los algoritmos de codificación de red tienen una complejidad NP-complete, lo que muestra que es inherentemente difícil encontrar una solución óptima y verificar que la solución propuesta, sin emabargo, es factible en tiempo polinomial. En otras palabras, ambos algoritmos plantean desafíos importantes en términos de eficiencia computacional y escalabilidad, pero en términos de tiempo, el Network Coding proporciona ventajas significantes.

La estrategia propuesta consiste en explotar las ventajas de la codificación de red y aplicarla a un entorno de enrutamiento utilizando matrices de sensores comprimidas. Al combinar estos métodos con el algoritmo de árbol de Steiner, se introduce un algoritmo de enrutamiento que hereda las características de eficiencia de la codificación de red, pero mantiene una complejidad computacional más manejable y está más cerca de la complejidad computacional del método del algoritmo de codificación de red.

En cuanto a las conclusiones obtenidas, se puede observar que el enfoque de algoritmo combinado ha dado resultados prometedores, a pesar de la falta de ventajas claras en términos de rendimiento en casos específicos como la multidifusión de sesión única y la transmisión de sesión única. En resultados actuales 0, si se aplica un algoritmo basado en codificación de red al algoritmo de enrutamiento, el rendimiento mejora en 1,55 en comparación con el algoritmo de enrutamiento estándar. Por otro lado, es de impresionante valor que el algoritmo desarrollado en este proyecto logró una mejora de rendimiento de 1,428 respecto al rendimiento del algoritmo del árbol de Steiner. Estos resultados confirman

la viabilidad y el potencial del método propuesto para mejorar el rendimiento de la comunicación de datos en redes complejas. En la siguiente ilustración se muestra la tabla de ejemplos con la mejora de rendimiento en cuanto a los resultados que proporciona el algoritmo respecto al rendimiento que ofrece el algoritmo del árbol de Steiner.

Network	$\|V\|$	$\|M\|$	$\|E\|$	$\chi(N)$	$\psi(N)$	$\pi(N)$	$\pi(N)/\psi(N)$	**# of trees**
C(3,2)	7	4	9	2	1.5	1.8	1.2	26
C(4,3)	9	5	16	3	2	2.667	1.333	1,113
C(4,2)	11	7	16	2	1.333	1.778	1.333	1,128
C(5,4)	11	6	25	4	2.5	3.571	1.428	75,524
C(5,2)	16	11	25	2	1.25	1.786	1.428	119,104

Ilustración 3: Resultados obtenidos tras la aplicación del algoritmo. Fuente: Elaboración propia.

El algoritmo empleado ha demostrado ofrecer prácticamente el mismo nivel de beneficio que el algoritmo reconocido mundialmente hasta el momento. Esta equiparación en términos de rendimiento subraya la eficacia del algoritmo desarrollado en este proyecto. Es importante destacar que este logro se ha alcanzado en un período de tiempo relativamente corto, lo que resalta la eficiencia del proceso de elaboración y desarrollo del algoritmo.

La capacidad de obtener resultados comparables a los de un algoritmo reconocido mundialmente en un lapso de tiempo más reducido refleja tanto la meticulosa investigación realizada como la habilidad para implementar soluciones innovadoras de manera eficaz. Este hecho pone de relieve la relevancia y la viabilidad del enfoque adoptado en este proyecto, así como la capacidad para desarrollar soluciones efectivas en un tiempo limitado.

NETWORK CODING AND ROUTING ANALYSIS IN DATA COMMUNICATION: CASE STUDIES IN MULTIPLE SCENARIOS

Author: Siljestrom Berenguer, Fernando.
Supervisor: A. Tang, Kevin.
Collaborating Entity: Cornell University

ABSTRACT

The information transmission capacity within a data network is limited by the network topology and connection capacity. Traditional techniques for improving transmission performance focus on strategically controlling the flow of information over a large bandwidth or multiple paths from source to destination.

Today, we know that such a routing strategy alone may not be sufficient. Rather, data encoding/decoding must be considered at the network nodes to achieve optimal performance. This is called Network Coding. It has been shown that network coding for single-session single forwarding and single session broadcast networks has the same performance as traditional routing [9] . For undirected single-session multicast networks, the benefit provided by network coding in terms of transmission rate is limited to 2, but for directed networks it can be infinite.

However, for undirected single-send multicast sessions where fractional routing is allowed, we have yet to find a case where network coding can provide benefit. In this project, we review the documentation and literature on network coding and originate computations and programs proposed in previous research, which can provide answers to the above questions.

This project has two objectives:

- First, to test the still valid conjecture that network encryption has no advantage over standard single-session multisession routing in undirected networks.

- Secondly, to elaborate an approximation algorithm starting from a network encryption algorithm (cFlow), approximated to a routing algorithm to avoid complexity (Steiner Tree Packing).

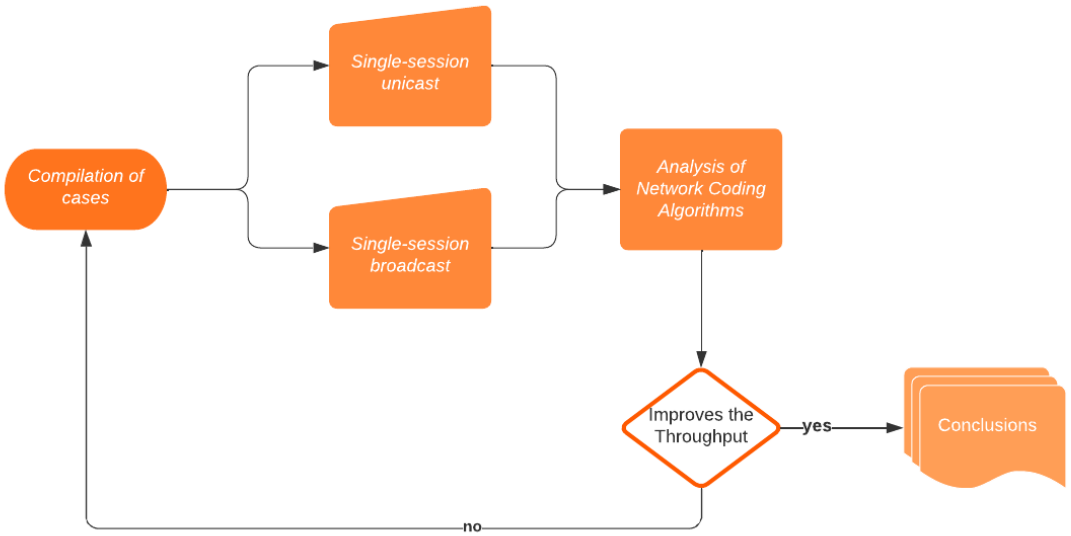


Ilustración 4: Scheme of the model for the first problem. Source: Own elaboration

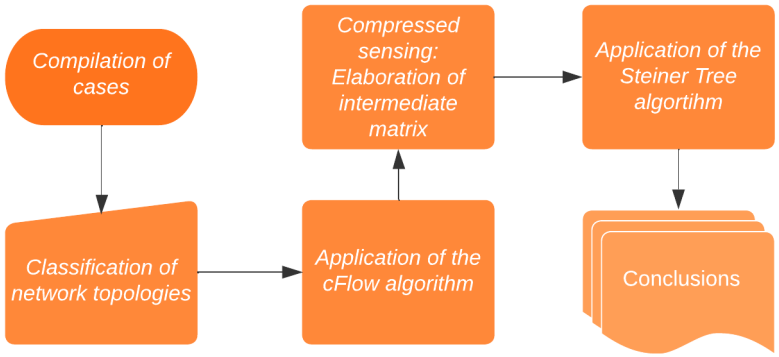


Ilustración 5: Scheme of the model for the second problem. Source: Own elaboration.

In order to try to meet the two proposed objectives, we perform a comprehensive review of modern algorithms for data communication, focusing on both routing and network encryption. Regarding routing, special attention was paid not only to the Steiner tree algorithm, but also to fractional routing and arbitrary fractional routing. These algorithms are fundamental for optimizing the efficiency of data transmission in networks, since they are capable of finding the most efficient route for information transmission, taking into

account aspects such as available bandwidth and connection capacity. At the same time, network coding algorithms have been studied in detail, including cFlow, oFlow and mFlow. These algorithms play an important role in improving network performance by enabling efficient data encoding and decoding at network nodes. Analysis of these algorithms provides an in-depth understanding of how network coding can increase transmission capacity and improve fault tolerance in different network configurations.

The main reason for the development of the algorithm described in Illustration 2 is the inherent complexity of existing algorithms in data communication. On the one hand, Steiner tree algorithms have NP-Hard complexity, which means that finding optimal solutions to such problems is computationally expensive and, in many cases, impractical in a reasonable period of time. On the other hand, network coding algorithms have NP-complete complexity, which shows that it is inherently difficult to find an optimal solution and verify that the proposed solution, however, is feasible in polynomial time. In other words, both algorithms pose important challenges in terms of computational efficiency and scalability, but in terms of time, Network Coding provides significant advantages.

The proposed strategy is to exploit the advantages of network coding and apply it to a routing environment using compressed sensor arrays. By combining these methods with the Steiner tree algorithm, a routing algorithm is introduced that inherits the efficiency characteristics of network coding but maintains a more manageable computational complexity and is closer to the computational complexity of the network coding algorithm approach.

In terms of the conclusions drawn, it can be seen that the combined algorithm approach has yielded promising results, despite the lack of clear advantages in terms of performance in specific cases such as single-session multicast and single-session broadcast. In current results 0, if a network coding-based algorithm is applied to the routing algorithm, the performance improves by 1.55 compared to the standard routing algorithm. On the other hand, it is of impressive value that the algorithm developed in this project achieved a performance improvement of 1.428 over the performance of the Steiner tree algorithm. These results confirm the feasibility and potential of the proposed method to improve the performance of data communication in complex networks. The following illustration shows the example table with the performance improvement in terms of the results provided by the algorithm with respect to the performance provided by the Steiner tree algorithm.

Network	$\|V\|$	$\|M\|$	$\|E\|$	$\chi(N)$	$\psi(N)$	$\pi(N)$	$\pi(N)/\psi(N)$	**# of trees**
C(3,2)	7	4	9	2	1.5	1.8	1.2	26
C(4,3)	9	5	16	3	2	2.667	1.333	1,113
C(4,2)	11	7	16	2	1.333	1.778	1.333	1,128
C(5,4)	11	6	25	4	2.5	3.571	1.428	75,524
C(5,2)	16	11	25	2	1.25	1.786	1.428	119,104

Ilustración 6: Results obtained after applying the algorithm. Source: Own elaboration.

The algorithm used has been shown to offer practically the same level of benefit as the algorithm recognized worldwide so far. This comparison in terms of performance underlines the effectiveness of the algorithm developed in this project. Importantly, this achievement has been accomplished in a relatively short period of time, highlighting the efficiency of the algorithm design and development process.

The ability to obtain results comparable to those of a world-renowned algorithm in a shorter time frame reflects both the meticulous research performed and the ability to implement innovative solutions in an efficient manner. This fact highlights the relevance and feasibility of the approach taken in this project, as well as the ability to develop effective solutions in a limited amount of time.

Índice de la memoria

Índice de figuras

Capítulo 1. INTRODUCCIÓN Y PLANTEAMIENTO DEL PROYECTO

En la era digital actual, donde la conectividad y el flujo de información son esenciales en casi todos los aspectos de nuestra sociedad, las redes de comunicación desempeñan un papel vital. Desde pequeñas redes de área local hasta grandes infraestructuras globales, estas complejas redes son responsables de la transmisión eficiente y en tiempo real de cantidades masivas de datos, asegurando que los datos lleguen a su destino de manera confiable y a tiempo.

Sin embargo, a medida que la demanda de ancho de banda y la complejidad de la red continúan aumentando, los métodos tradicionales de enrutamiento y entrega de datos enfrentan desafíos cada vez mayores. Las limitaciones asociadas con estos enfoques, como la congestión de la red, el desperdicio de recursos y la falta de escalabilidad, amenazan con reducir la eficiencia y eficacia de las comunicaciones en el futuro cercano.

Al abordar estos desafíos, dos áreas de investigación han surgido como soluciones prometedoras: codificación de red y algoritmos de enrutamiento avanzados. La codificación de red, un concepto revolucionario introducido por Ahlswede, Cain, Li y Yeung a principios de la década de 2000, promete mejorar significativamente la eficiencia de la transmisión de datos mediante la combinación y codificación de paquetes de datos en los nodos de la red. Al permitir que los nodos procesen y mezclen datos entrantes en lugar de simplemente reenviarlos, la codificación de red puede lograr velocidades de transmisión más altas, reducir la congestión y optimizar el uso de los recursos de la red.

Sin embargo, investigaciones recientes han demostrado que el uso de codificación de red en ciertos contextos, como multidifusiones de sesión única y transmisiones de sesión única, no proporciona una ventaja de velocidad sobre los algoritmos de enrutamiento tradicionales. Este hallazgo plantea dudas sobre la efectividad y aplicabilidad de la codificación de red en

ciertos escenarios de comunicación y resalta la importancia de investigar nuevas estrategias y enfoques que puedan superar las limitaciones identificadas.

Por otro lado, los algoritmos de enrutamiento avanzados, como el clásico problema del árbol de Steiner, han jugado un papel importante en la optimización del flujo de datos en las redes de comunicación. El propósito de estos algoritmos es determinar rutas óptimas para la transmisión de datos, minimizar el uso de recursos y maximizar la eficiencia. Sin embargo, la complejidad computacional asociada con estos algoritmos clasificados NP-Hard resalta la necesidad de desarrollar soluciones más eficientes y escalables para problemas de enrutamiento en redes complejas y en constante evolución.

Ante estos desafíos, existe la necesidad de un enfoque innovador que combine principios de codificación de red con técnicas de enrutamiento avanzadas, explotando las fortalezas de ambos enfoques y mitigando sus limitaciones. La integración de estas dos áreas de investigación abre la posibilidad de lograr un equilibrio óptimo entre rendimiento y complejidad, que mejore la eficiencia y escalabilidad de la transmisión de datos en las redes de comunicación.

Este proyecto se centra en explorar la intersección de la codificación y el enrutamiento en esta red con el objetivo de desarrollar una solución integral que aborde los desafíos identificados. Utilizando un enfoque multidisciplinario que incluye teoría de la información, optimización, teoría de grafos y simulación de redes, buscamos comprender a fondo las fortalezas y debilidades de cada enfoque y diseñar un marco unificado que utilice lo mejor de ambos mundos.

Este avance está motivado por la urgente necesidad de responder a desafíos críticos en las redes de telecomunicaciones y desarrollar soluciones innovadoras que mejoren el rendimiento y la eficiencia de la transmisión de datos. Además, la comprensión de que la codificación de red no ofrece ventajas significativas en ciertos contextos de comunicación y la complejidad computacional asociada con los algoritmos de enrutamiento resalta la importancia de explorar enfoques integrados que combinen lo mejor de ambas técnicas.

Con este proyecto, se pretende crear la base para una nueva generación de redes de comunicación que sean más eficientes, escalables y capaces de soportar las crecientes demandas de tráfico. Aprovechando los avances en codificación y enrutamiento de redes, nos esforzamos por desarrollar soluciones que puedan aplicarse a una amplia variedad de entornos, desde redes inalámbricas y redes de sensores hasta redes troncales de Internet y redes celulares.

Además, los resultados de este estudio pueden tener un impacto significativo en diversos sectores como las telecomunicaciones, los servicios en la nube, el Internet de las cosas (IoT) y todas las industrias que dependen de una transmisión de datos eficiente. Al abordar los desafíos actuales y futuros en este campo, este proyecto contribuirá a mantener la viabilidad y la capacidad de las redes de telecomunicaciones y garantizará que puedan seguir siendo las piedras angulares de nuestra sociedad digital en constante evolución.

Se propuso una estrategia de investigación que consta de dos fases principales para abordar los desafíos identificados. La primera fase consistió en un análisis exhaustivo de los algoritmos de enrutamiento y codificación de red existentes, con el objetivo de identificar sus fortalezas, debilidades y limitaciones en diversos escenarios de transferencia de datos. Este análisis se realizó a través de una revisión sistemática de la literatura académica relevante, incluidos artículos de investigación, actas de congresos y libros especializados.

La segunda fase desarrolló un nuevo enfoque que combina principios de codificación de red con técnicas de enrutamiento avanzadas. El principal objetivo de este enfoque integrado fue aprovechar las ventajas de ambas estrategias y mejorar las limitaciones identificadas en el paso anterior. Para lograr este objetivo, se realizaron una serie de experimentos y simulaciones utilizando diferentes entornos de red y patrones de tráfico.

El planteamiento utilizado en este proyecto se dividió en varios pasos interrelacionados:

1. **Revisión de la literatura:** se realizó una revisión extensa de la literatura académica relevante para obtener una comprensión profunda de los algoritmos de enrutamiento

y codificación de red existentes. En esta etapa se mapearon y analizaron artículos científicos, conferencias y libros especiales en el campo de las redes de telecomunicaciones.

2. **Análisis comparativo:** Después de la revisión de la literatura, se realizó un análisis comparativo de diferentes métodos de codificación en ruta y de red. Este análisis evalúa las fortalezas, debilidades y limitaciones de cada enfoque en escenarios de comunicación específicos, como multidifusión de sesión única, transmisión de sesión única y redes complejas.
3. **Enfoque integrado:** basándose en los resultados de las pruebas comparativas, se desarrolló un nuevo enfoque que combina principios de codificación de red con técnicas de enrutamiento avanzadas. Este enfoque integrado se desarrolló con el objetivo de explotar las ventajas de ambas estrategias y abordar las limitaciones identificadas.
4. **Simulaciones y experimentos:** para evaluar la eficacia del enfoque integrado propuesto, se realizaron una serie de simulaciones y experimentos utilizando diferentes entornos de red y patrones de tráfico. Estas simulaciones y experimentos han sido cuidadosamente diseñados para representar escenarios realistas y desafiantes en el campo de las redes de telecomunicaciones.
5. **Evaluación y refinamiento:** los resultados obtenidos de simulaciones y experimentos se analizaron en detalle para evaluar la efectividad del enfoque integrado propuesto. Este análisis nos permitió identificar áreas de mejora y perfeccionar el enfoque cuando fuera necesario.
6. **Validación y pruebas finales:** después del refinamiento, se realizaron pruebas y validaciones finales para garantizar la solidez y eficacia del enfoque integrado propuesto en múltiples escenarios de comunicación.

Para organizar y visualizar el progreso del proyecto, se ha elaborado un diagrama de Gantt que muestra de manera clara y concisa las diferentes etapas y actividades planificadas. El diagrama de Gantt es una herramienta ampliamente utilizada en la gestión de proyectos que permite establecer un cronograma detallado y gestionar eficazmente los recursos y el tiempo.

En la siguiente figura se presenta el diagrama de Gantt correspondiente al proyecto, donde se detallan las tareas planificadas, su duración estimada y las fechas de inicio y finalización previstas. Este diagrama proporciona una visión general del cronograma del proyecto, lo que facilita la identificación de posibles solapamientos, retrasos o problemas de planificación, permitiendo así tomar medidas correctivas de manera oportuna.

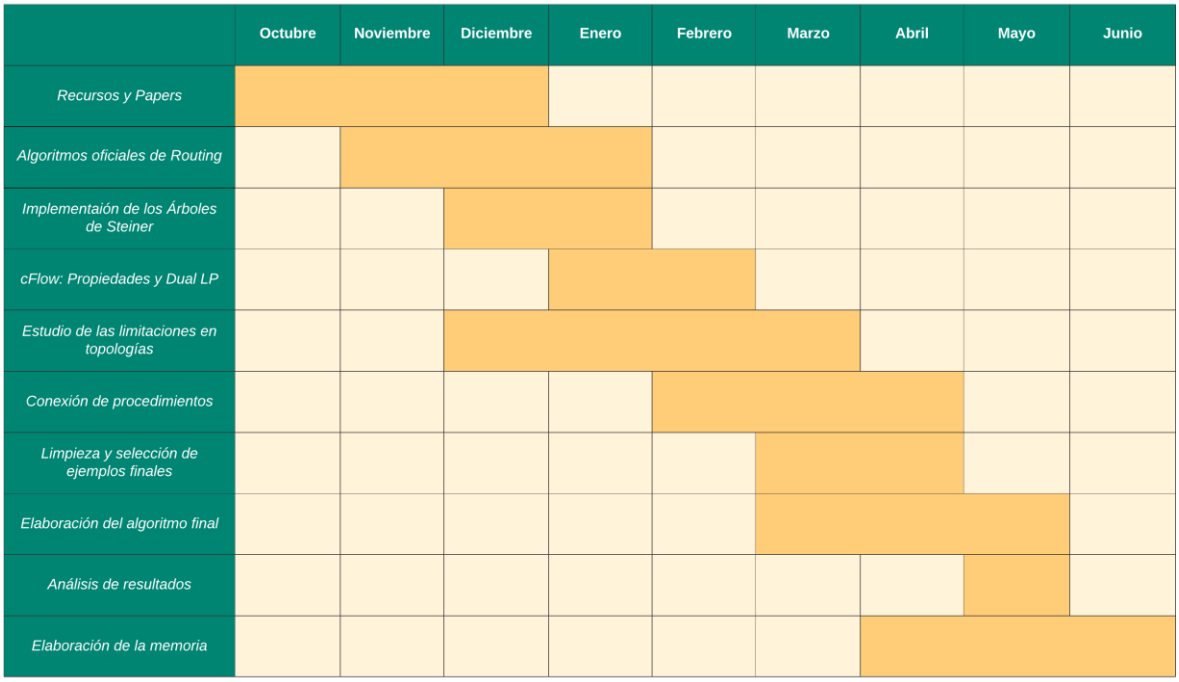


Figura 1. Diagrama de Gantt del proyecto. Fuente: Elaboración propia.

Capítulo 2. ESTADO DE LA CUESTIÓN

En las siguientes páginas se van a explicar los principales algoritmso utilizados para la elaboración del modelo y las pruebas. En este apartado simplemente se van a describir de forma que se entiendan las bases de su funcionamiento. En el próximo capítulo, se demostrará el funcionamiento de estos algoritmos aplicados a casos de uso reales.

1. Algoritmo cFlow:

Concepto y definición

El algoritmo cFlow es un método utilizado para codificar flujos en redes de datos. Este enfoque se basa en la idea de permitir la transmisión simultánea de múltiples flujos de datos a través de la red, lo que puede mejorar la utilización del ancho de banda y la flexibilidad en la gestión del tráfico.

Funcionamiento de cFlow

cFlow se basa en codificar datos en nodos intermedios de la red, no solo en reenviar paquetes de datos. Esta codificación permite combinar y transmitir datos de manera más eficiente. Los pasos principales de una operación de cFlow se describen a continuación:

1. Codificación de origen: los datos se dividen en bloques y se codifican en el nodo de origen utilizando un algoritmo de codificación lineal. Esta codificación crea combinaciones lineales de datos sin procesar.
2. Transmisión de red: los paquetes cifrados se envían a través de la red. En cada nodo intermedio, los paquetes recibidos se decodifican y recodifican parcialmente antes de reenviarlos. Este proceso continuo de codificación y decodificación permite que los datos se mezclen y transmitan de manera eficiente.

3. Decodificación en el destino: los paquetes cifrados en el nodo de destino se descifran para restaurar los datos originales. La decodificación utiliza un algoritmo que resuelve un sistema de ecuaciones lineales para obtener la información original de las combinaciones recibidas.

Beneficios de cFlow

- Uso eficiente del ancho de banda: al permitir la agregación de datos en nodos intermedios, cFlow puede mejorar el uso del ancho de banda y reducir la redundancia de transmisión de datos.
- Durabilidad y flexibilidad: el cifrado de datos permite que la red sea resistente a fallas y cambios topológicos, ya que los datos se pueden recuperar a partir de combinaciones parciales.
- Optimización del rendimiento: cFlow puede optimizar el rendimiento de la red minimizando la congestión y mejorando la eficiencia de la transferencia de datos.

2. Algoritmo mFlow:

Concepto y definición

El algoritmo mFlow es una extensión de cFlow que se utiliza para procesar múltiples flujos de datos simultáneamente en una red. Mientras que cFlow se centra en codificar un único flujo de datos, mFlow está diseñado para optimizar la entrega de múltiples flujos de datos, aumentando la flexibilidad y eficiencia de la gestión del tráfico de red.

Cómo funciona mFlow

mFlow sigue los mismos principios que cFlow, pero con modificaciones para manejar múltiples flujos de datos:

1. Codificación de origen: cada flujo de datos se codifica inicialmente de forma independiente en un nodo. Esto crea múltiples combinaciones lineales, una para cada flujo de datos.
2. Transmisión y codificación en nodos intermedios: los paquetes de datos cifrados se envían a través de la red. En los nodos intermedios, los paquetes de diferentes flujos se pueden combinar y cifrar juntos, lo que permite una transmisión más eficiente y reduce la congestión.
3. Decodificación en destino: los paquetes recibidos en los nodos de destino se decodifican para restaurar los flujos de datos originales. La decodificación utiliza algoritmos que resuelven ecuaciones lineales para cada flujo de datos por separado.

Ventajas de mFlow

- Gestión de flujo múltiple: mFlow está especialmente diseñado para gestionar múltiples flujos de datos, lo que lo hace adecuado para redes con tráfico diverso.
- Mejora del rendimiento: al permitir que se unan y cocodifiquen múltiples transmisiones, mFlow puede mejorar la eficiencia de la transmisión de datos y reducir la congestión de la red.
- Flexibilidad y adaptabilidad: mFlow ofrece mayor flexibilidad y adaptabilidad en la gestión del tráfico de red, lo que permite una optimización más eficiente de los recursos de la red.

3. Algoritmo oFlow:

Concepto y definición

El algoritmo oFlow es otra extensión de cFlow, que se enfoca en optimizar la transmisión de flujos de datos en redes con una topología compleja y dinámica:

1. oFlow está diseñado para adaptarse a los cambios de topología de la red y mejorar la resiliencia ante fallas e interrupciones. oFlow se adapta según las condiciones de la red y los requisitos de calidad del servicio.
2. Reenvío dinámico y reconfiguración: los paquetes cifrados se envían a través de la red. En los nodos intermedios, oFlow utiliza técnicas de reconfiguración dinámica para ajustar las rutas de codificación y transmisión en tiempo real según la topología y las condiciones de la red.
3. Decodificación y recuperación de destino: en los nodos de destino, los paquetes se decodifican utilizando algoritmos que pueden manejar variaciones en las combinaciones de datos recibidos, lo que permite una recuperación eficiente de los datos originales.

Ventajas de oFlow

- Adaptabilidad dinámica: oFlow está diseñado para adaptarse a los cambios de topología de la red y optimizar la transferencia de datos en tiempo real.
- Tolerancia a fallos: la capacidad de reconfiguración dinámica de oFlow mejora la resiliencia de la red ante fallos e interrupciones, lo que permite una recuperación rápida y eficiente.
- Optimización de la calidad del servicio: oFlow puede adaptar la codificación y transmisión de datos según los requisitos de calidad del servicio, mejorando la experiencia del usuario final.

4. Steiner Tree Packing:

Definición

Steiner Tree Packing es una técnica utilizada para optimizar la transmisión de datos en redes descomponiendo la red en árboles Steiner ponderados. Un árbol de Steiner es un subárbol mínimo que conecta un conjunto determinado de nodos en una red minimizando el costo total de las conexiones.

Funcionamiento

Steiner Tree Packing opera en línea dependiendo de los requisitos de transferencia de datos:

1. Identificación de Nodos Clave: Se identifican los nodos clave que deben ser conectados en la red, en función de los requisitos de transmisión de datos.
2. Construcción de árboles Steiner: los árboles Steiner se construyen para conectar nodos clave, lo que reduce el costo total de las conexiones. Estos árboles pueden superponerse y compartir enlaces para optimizar el uso de los recursos de la red.
3. Compresión de árbol: los árboles Steiner construidos se comprimen de manera eficiente para maximizar la utilización del ancho de banda y minimizar la congestión de la red. Este paquete incluye asignación de recursos de red y enrutamiento de transmisión.
4. Transferencia de datos: los datos se transfieren a través de árboles Steiner comprimidos utilizando conexiones optimizadas para mejorar la eficiencia y el rendimiento de la red.

Ventajas de Steiner Tree Packing

- Optimización del ancho de banda: al minimizar el costo total de las conexiones y optimizar el uso del ancho de banda, Steiner Tree Packing puede mejorar significativamente el rendimiento de la red.
- Reducción de la congestión: la tecnología de compresión permite una planificación eficiente de las rutas de transmisión, reduce la congestión y mejora la calidad del servicio.
- Flexibilidad y escalabilidad: Steiner Tree Packing es adecuado para redes grandes y puede adaptarse a diferentes topologías y condiciones de red, proporcionando una solución flexible y escalable para la transmisión de datos.

Comparación de algoritmos

cFlow vs. mFlow: mientras que cFlow se centra en codificar un único flujo de datos, mFlow está diseñado para procesar múltiples flujos de datos simultáneamente. Esto hace que mFlow sea más adecuado para redes con tráfico diverso y múltiples sesiones de comunicación.

cFlow vs. oFlow: oFlow introduce optimización adaptativa y tecnología de reconfiguración dinámica, lo que la hace más robusta y adaptable que cFlow. oFlow es ideal para redes de topología dinámica donde la tolerancia a fallos es fundamental.

Steiner Tree Packing: a diferencia de los algoritmos de codificación de streaming, Steiner Tree Packing se centra en optimizar la estructura de la red mediante la creación de árboles Steiner. Esta tecnología complementa los algoritmos de codificación y se pueden utilizar en conjunto para mejorar la eficiencia de la transmisión de datos.

Capítulo 3. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para contextualizar la relevancia y necesidad de este proyecto, es importante investigar qué trabajos y soluciones existen actualmente en el mercado. Examinar investigaciones previas y tecnologías disponibles nos permite identificar los avances realizados hasta la fecha, las limitaciones observadas y las lagunas de conocimiento que este estudio pretende abordar. Este análisis inicial no solo respalda la justificación del proyecto, sino que también ayuda a desarrollar soluciones innovadoras y efectivas.

Primero, examinemos y probemos la validez de la conjetura anterior y evaluemos si el cifrado de red en realidad no proporciona beneficios adicionales en comparación con el enrutamiento estándar de sesión única en redes no dirigidas. El objetivo de esta parte de la investigación es contribuir al cuerpo de conocimientos existente y revelar la eficiencia y viabilidad del cifrado de redes en este contexto.

En segundo lugar, para reducir la complejidad, nos gustaría desarrollar un algoritmo de aproximación que se aproxime a los algoritmos de enrutamiento, como el empaquetado del árbol Steiner, basado en los algoritmos de cifrado de red existentes (cFlow). Este enfoque tiene como objetivo proporcionar una solución más práctica y manejable para la implementación y operación en la red manteniendo al mismo tiempo un alto nivel de seguridad y eficiencia.

Especulación sobre la efectividad del cifrado de red:

Varios estudios han discutido la especulación de que el cifrado de red no tiene beneficios adicionales sobre el enrutamiento estándar en redes no dirigidas. Esta conjetura se evalúa específicamente en [9]. Este estudio concluye que bajo ciertas condiciones de enrutamiento fraccionado en redes no dirigidas, el cifrado de red no ofrece ventajas de rendimiento significativas sobre el enrutamiento estándar.

Por otro lado, en cuanto al análisis de codificación de red en redes no dirigidas, en [9], también se analiza el rendimiento de la codificación de red en comparación con el enrutamiento en redes no dirigidas. En este mismo artículo, los autores muestran que, aunque en teoría el cifrado de red mejora la eficiencia del ancho de banda, en la práctica estas mejoras pueden ser mínimas debido a la complejidad adicional que añade el cifrado de red. Este estudio sugiere que la utilidad del cifrado de red en redes no dirigidas es limitada y puede no justificar la complejidad adicional en términos de implementación y gestión. Cuando se trata de métricas de rendimiento, las investigaciones muestran que las mejoras teóricas a menudo no se traducen en beneficios prácticos debido a las limitaciones físicas y operativas de las redes del mundo real. La complejidad computacional y la necesidad de coordinación adicional para implementar el cifrado de red son factores que pueden afectar la viabilidad en comparación con el enrutamiento estándar.

Desarrollo de algoritmos de aproximación basados en cFlow:

Para abordar la reducción de la complejidad de las implementaciones de cifrado de red, es importante desarrollar algoritmos de aproximación efectivos. Aquí es donde resulta útil utilizar cFlow y el empaquetado en árbol de Steiner.

Algoritmo cFlow:

El algoritmo cFlow se utiliza para manejar eficientemente la codificación de flujo en una red. Este enfoque permite enviar múltiples flujos de datos a través de la red simultáneamente, lo que aumenta la utilización del ancho de banda y la flexibilidad de la gestión del tráfico.

Al aplicar cFlow en el contexto de redes no dirigidas, podemos desarrollar algoritmos aproximados que brinden beneficios de rendimiento sin tener que aceptar por completo la complejidad del cifrado de red.

Steiner Tree Packing:

Steiner Tree Packing es una técnica que tiene como objetivo maximizar el rendimiento de la red descomponiendo una red en árboles Steiner ponderados. Esta técnica es particularmente

útil para minimizar la complejidad computacional al procesar redes grandes. Al combinar cFlow y Steiner Tree Packing, puede crear algoritmos de aproximación que equilibren el rendimiento y la complejidad.

La justificación para iniciar este proyecto se basa en dos objetivos principales: Validar una suposición: el cifrado de red es útil para redes no dirigidas con múltiples sesiones de unidifusión para ver si ofrece alguna ventaja significativa sobre el enrutamiento estándar. Esto tiene implicaciones teóricas y prácticas para el diseño de redes eficientes.

Desarrollo de algoritmos eficientes: Utilización del marco cFlow y el paquete de árbol Steiner para crear algoritmos aproximados que reduzcan la complejidad computacional sin afectar significativamente el rendimiento. Este enfoque puede proporcionar una solución práctica para mejorar la eficiencia del enrutamiento en redes grandes.

3.1 ROUTING ALGORITHMS

3.1.1 FORD-FULKERSON

Para analizar los ejemplos del Capítulo 5, se empezarán estudiando los algoritmos de routing explicados verbalmente en los apartados anteriores. El objetivo de este apartado es la demostración y exploración de los algoritmos de manera analítica.

Por un lado, existen algoritmos de ruteo que hacen referencia al concepto de multicommodity flow, como las situaciones óptimas del corte mínimo o el máximo flujo, que es el algoritmo de Ford-Fulkerson.

En el siguiente ejemplo se muestra un ejemplo muy sencillo de máximo flujo para el caso de una red single-unicast con aristas dirigidas.

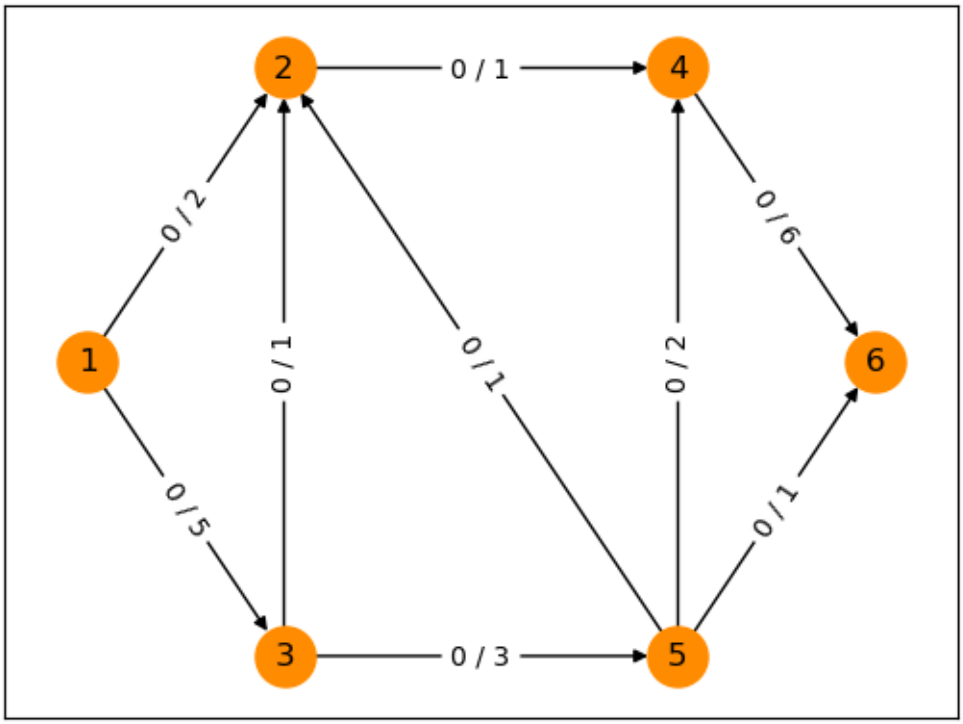


Figura 2: Situación inicial. Fuente: elaboración propia

En la figura anterior se muestra la situación inicial, y el objetivo es llevar el máximo flujo desde el nodo 1 al nodo 6, por eso es una situación de single-unicast.

La metodología es muy sencilla, simplemente hay que buscar las aristas con mínimo flujo y empezar a enviar flujo por esas aristas, hasta el punto en el que el flujo que circula por esas aristas iguale a la capacidad de estas. Siguiendo estos pasos, llegamos a la siguiente situación:

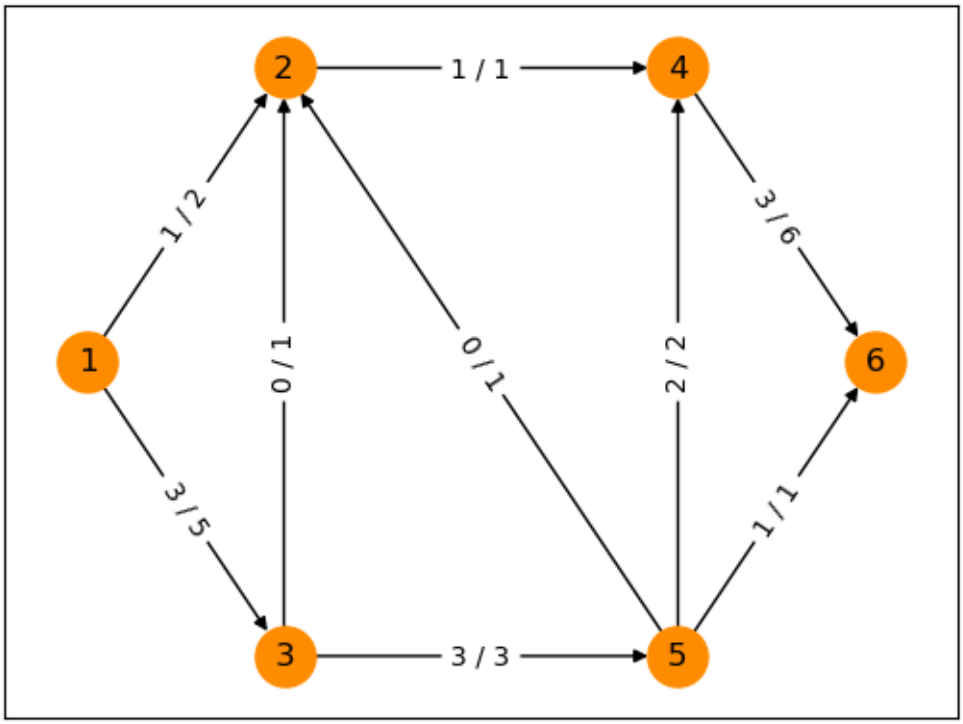


Figura 3: situación final

La pregunta es, ¿cómo sabemos que hemos acabado? Aquí entra la representación de otro tipo de diagrama, que son los llamados diagramas residuales. El gráfico residual es una representación de la red original que refleja la funcionalidad restante de las aristas después de considerar el flujo de corriente en la red. Este diagrama le ayuda a identificar rutas adicionales que pueden aumentar el flujo. Por un lado, la capacidad residual es la capacidad adicional que la arista puede soportar dependiendo del flujo de corriente.

Si la arista $(u\ ,\ v)$ tiene una capacidad $c(u\ ,\ v)$ y un flujo $f\backslash left(u, v\backslash right)$, la capacidad residual es $r(u, v) = c(u, v) - f(u, v)$

Además, en cuanto a las aristas, el gráfico residual contiene las aristas en la dirección opuesta, lo que puede permitir disminuir el flujo. Si hay un flujo $f(u, v)$ en la arista (u, v), entonces en el grafo residual se añade una arista inversa (v, u) con capacidad residual $f(u, v)$.

Con estas directrices, llegamos al siguiente grafo:

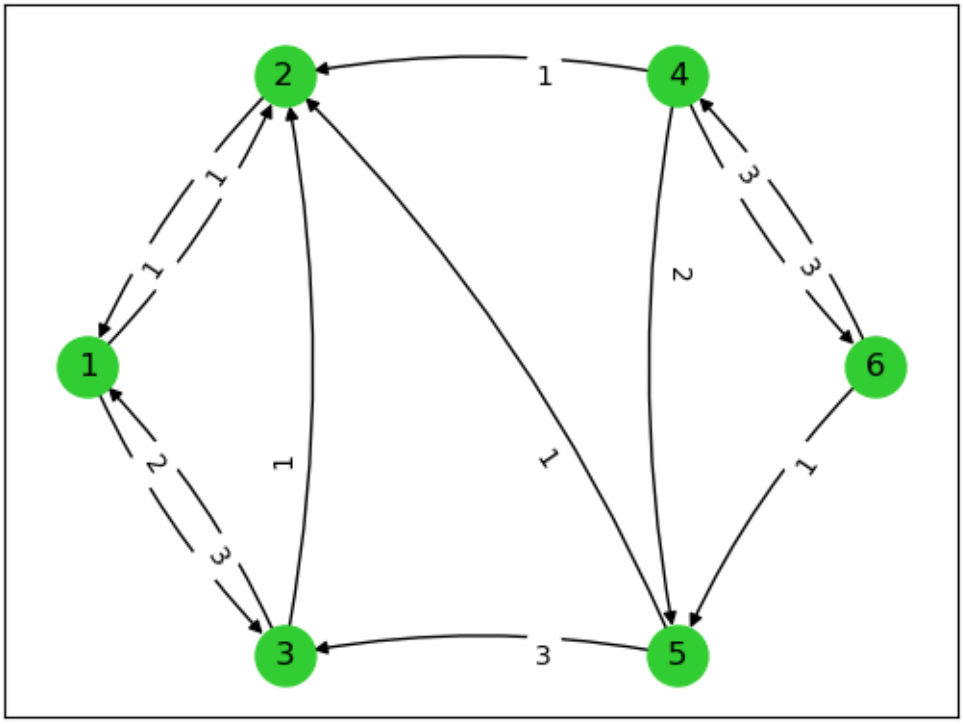


Figura 4: grafo residual final

En la figura anterior se muestra la situación final, y sabemos que no podemos avanzar más ya que en el grafo residual no hay ningún camino por el cual llegar desde el origen al destino.

Además, sabemos cuál es el máximo flujo y el mínimo corte, ya que únicamente tenemos que hacer un corte precisamente por las aristas que no nos dejan llegar desde el origen al destino. Es decir, en este caso tenemos un flujo máximo de 5 unidades, que representan la suma de los flujos al hacer un corte perpendicular por el medio del diagrama.

3.1.2 STEINER PACKING Y ROUTING FRACCIONAL

Para ilustrar cómo el algoritmo del árbol de Steiner corresponde a un rendimiento óptimo alcanzable sin network coding, se muestra un ejemplo en Figura 5. En la ilustración, cada letra corresponde a un árbol de Steiner distinto que conecta al grupo de multidifusión, compuesto por m_0, m_1, m_2 y m_3. Existen nueve árboles de Steiner en este esquema de

empaquetamiento (de la a a la i), donde el árbol correspondiente a la letra que está resaltada. Dado que cada arista con capacidad unitaria necesita acomodar 5 árboles de Steiner, el rendimiento alcanzable en cada árbol es, por lo tanto, 0.2. Esto lleva a un rendimiento de multidifusión de 1.8, que es óptimo sin codificación.

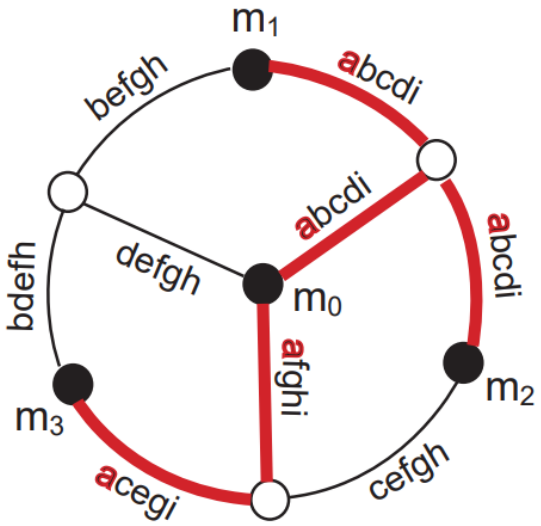


Figura 5: red ejemplo de aristas bidireccionales

Desafortunadamente, se ha demostrado que el algoritmo del árbol de Steiner es NP-complete y APX-Hard, y el mejor algoritmo conocido en tiempo polinomial tiene una razón de aproximación de alrededor de 1.55 [6]. No es lo suficientemente preciso para ser utilizado en la práctica.

No obstante, el algoritmo de los árboles de Steiner puede no ser regularmente fraccional. Es decir, se pueden aplicar algoritmos para deducir cuál es el peso más adecuado para cada árbol, y de esta forma maximizar el rendimiento y velocidad de transmisión. Este algoritmo se conoce como el simplex-method, que consiste en lo siguiente:
En primer lugar, debemos encontrar una función objetivo la cual queremos maximizar. En este caso, esa función objetivo es el flujo que circula por las aristas desde origen a destino. En segundo lugar, tenemos una serie de restricciones, que son las condiciones lineales que deben cumplirse. Estas restricciones se expresan generalmente en forma de desigualdades

lineales (o ecuaciones), por ejemplo, que el flujo que circula por las aristas no puede ser negativo o mayor que uno… Este sistema de ecuaciones se puede resolver introduciendo las condiciones en la función objetivo, hasta llegar al resultado final. No obstante, ak ser un procedimiento tedioso, explicaré el funcionamiento de este con los propios ejemplos de explicación para los casos de single y multiple unicast.

3.1.3 CFLOW LP

cFlow es un algoritmo de network coding que consiste en maximizar el flujo que circula por las aristas de manera conceptual. Con esto último, debemos tener en cuenta que tenemos que olvidar el concepto del multicommodity flow, ya que esas barreras naturales como que el flujo que circula por las aristas tiene que ser menor o igual que uno, puede que no se cumpla, y veamos el flujo mediante símbolos.

En la siguiente imagen, se muestra la comparativa de velocidad de transmisión dependiendo del número de nodos que haya en la red, tomando el sistema utilizando flows conceptuales o multicommodity flows. Como se puede observar, vemos que en cuanto a rendimiento según el número de nodos, los commodity flows no son escalables y limitados.

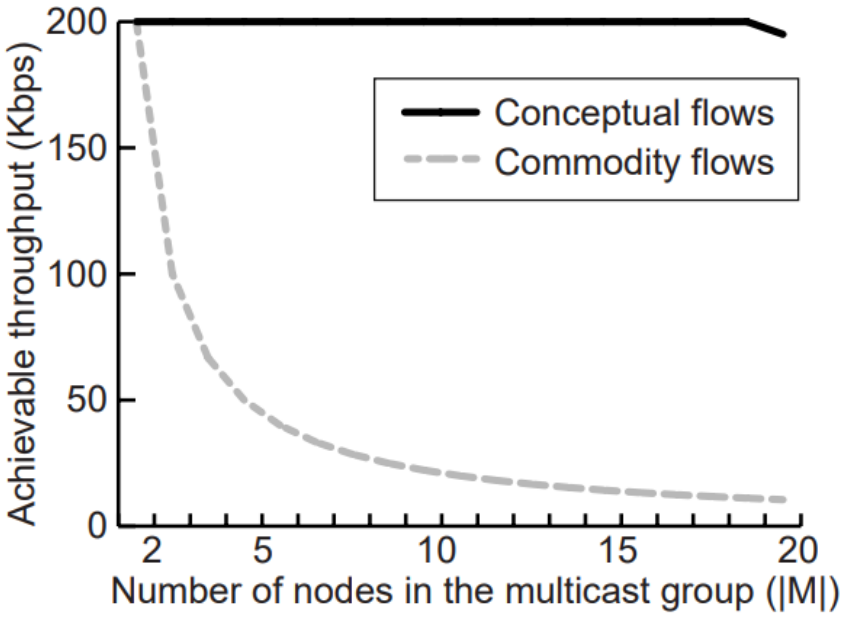


Figura 6: comparativa de throughput. Fuente: [6]

En las líneas siguiente, se presenta el cFlow LP y su explicación analítica.

Maximizar: f ∗

Sujeto a:

E. 1 $$0 \leq C(a) \qquad \forall a \in D$$

E. 2 $$C(a_1) + C(a_2) = C(e) \qquad \forall e \in E$$

E. 3 $$0 \leq f^i(a) \qquad \forall i \in [1..k], \forall a \in D$$

E. 4 $$f^i(a) \leq C(a) \qquad \forall i \in [1..k], \forall a \in D$$

E. 5 $$f^i_{in}(v) = f^i_{out}(v) \qquad \forall i \in [1..k], \forall v \in V - \{m_0, m_i\}$$

E. 6 $$f^i_{in}(m_0) = 0 \qquad \forall i \in [1..k]$$

E. 7 $$f^i_{out}(m_i) = 0 \qquad \forall i \in [1..k]$$

E. 8 $$f* = f_{in}^{i}(m_i) \quad \forall\, i \in [1..k]$$

En LP, m_0 es la fuente y m_1,…, m_k son los receptores de multidifusión. f_1,…,f_k son los flujos conceptuales hacia cada receptor. Cada $f\hat{}i$ especifica una tasa de flujo $f^i(a)$ para cada arista dirigida $a \in D$. $f_{in}^{i}(v)$ es la tasa de flujo entrante de f^i en un nodo v, de manera similar para $f_{out}^{i}(v)$. Finalmente, $f*$ es el flujo objetivo a maximizar.

Además de las restricciones de orientación, el LP de cFlow también incluye las restricciones de flujo de red para cada flujo conceptual y las restricciones de tasa igual. Las restricciones de flujo de red se especifican en una forma compacta para todos los flujos conceptuales, lo que requiere lo siguiente: Las tasas de flujo deben ser no negativas y estar limitadas superiormente por las capacidades de los enlaces; la conservación del flujo, que requiere que la tasa de flujo entrante en el flujo conceptual f^i sea igual a la tasa de flujo saliente en f^i en un nodo de retransmisión para f^i; y la tasa de flujo entrante en la fuente y el flujo saliente en los receptores son todas cero, para cada f^i.

Por otro lado, en la siguiente figura se muestra la comparativa en cuanto a capacidad de enlace, mostrando la eficiencia en ancho de banda. No obstante, para los ejemplos que se mostrarán en el siguiente capítulo, las capacidades siempre serán unitarias.

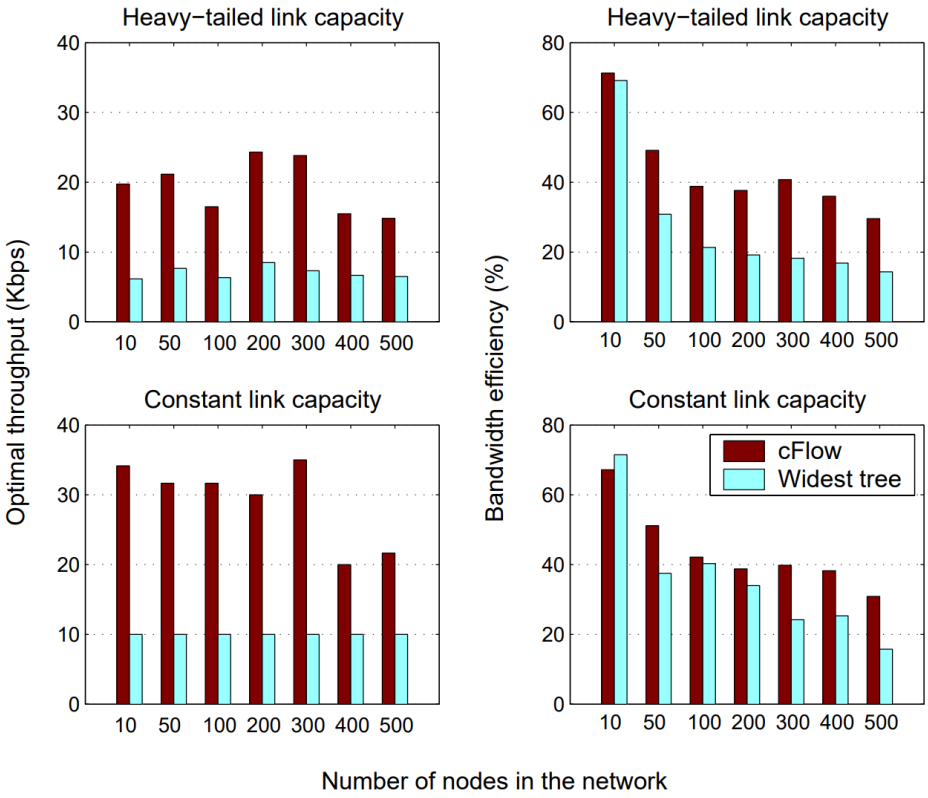


Figura 7: Throughput y ancho de banda: Fuente: [6]

3.1.4 MFLOW LP

Por otro lado, mientras que en cFlow se trataban los flujos conceptuales independientes para cada receptor de la multidifusión, mFlow está diseñado para optimizar la transmisión de datos en escenarios de multidifusión. Es decir, se enfoca en la eficiencia de la transmisión desde un origen a múltiples receptores simultáneamente. mFlow es especialmente útil en aplicaciones de transmisión en tiempo real, como streaming o contenido multimedia de gran volumen. A continuación, se va a presentar el LP de mFlow, que como se verá, es muy similar al de cFlow pero con algunas peculiaridades.

Maximizar: $f*$

Sujeto a:

E. 9	$0 \leq C'(a)$	$\forall a \in D$
E. 10	$C(a_1) + C(a_2) = C(e)$	$\forall e \in E$
E. 11	$0 \leq f^{ij}(a)$	$\forall i \in [1..s], \forall j \in [1..k_i], \forall a \in D$
E. 12	$f^{ij}(a) \leq f^{i}(a)$	$\forall i \in [1..s], \forall j \in [1..k_i], \forall a \in D$
E. 13	$\sum_{i=1}^{s} f^{i}(a) \leq C(a)$	$\forall a \in D$
E. 14	$f_{in}^{ij}(v) = f_{\text{out}}^{ij}(v)$	$\forall i \in [1..s], \forall j \in [1..k_i], \forall v \in V - \{m_{i_0}, m_{i_j}\}$
E. 15	$f_{in}^{ij}(m_{i_0}) = 0$	$\forall i \in [1..s], \forall j \in [1..k_i]$
E. 16	$f_{\text{out}}^{ij}(m_{i_j}) = 0$	$\forall i \in [1..s], \forall j \in [1..k_i]$
E. 17	$f^{i*} = f_{in}^{ij}(m_{i_j})$	$\forall i \in [1..s], \forall j \in [1..k_i]$
E. 18	$f^{*} = f^{i*}/w_i$	$\forall i \in [1..s]$

El mFlow LP se basa en la corrección del cFlow LP, que se ha mostrado anteriormente, pero en este caso es idóneo para múltiples sesiones de multicast. Solo se tiene que verificar que la relajación de $f^i(a) = max_{j\in[1\ldots k_i]} f^{ij}(a)$ a $f^i(a) \geq f^{ij}(a)$, $\forall j \in [1 \ldots k_i]$, no afecta el resultado general de la optimización. Cabe señalar que las variables $f^i(a)$ aparecen solo una vez. Por esta razón, si se encuentra una solución en la que $f^i(a) = max_{j\in[1\ldots k_i]} f^{ij}(a)$, si se alteran los valores de $f^i(a)$ de forma que $f^i(a) = max_{j\in[1\ldots k_i]} f^{ij}(a)$ debe dar salida a otra solución posible con el mismo resultado de optimización. Por ello, la relajación no llevará a una generalización óptima.

En la siguiente figura, podemos ver una comparativa del rendimiento de la red frente al número de sesiones y número de nodos de la red. Como se puede observar, conforme el

número de sesiones crece, el rendimiento disminuye de manera notable. No obstante, para el caso de mFlow, no afecta significativamente el número de nodos de la red.

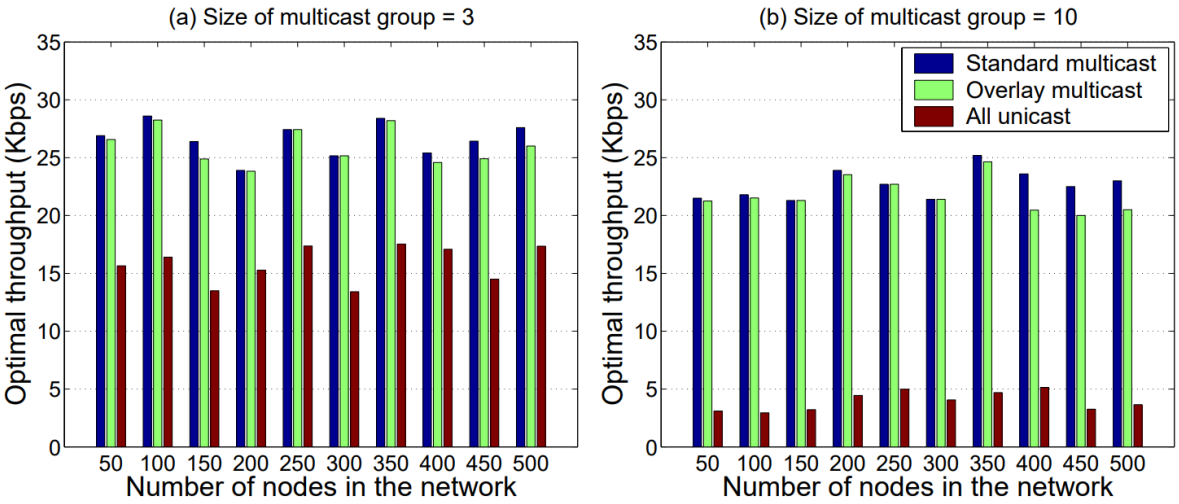


Figura 8: Throughput según el número de nodos. Fuente: [6]

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

El desarrollo de esta investigación surgió de la necesidad de abordar y resolver importantes problemas en el campo del cifrado y enrutamiento de redes en redes no dirigidas. A través de una revisión integral de estudios previos y un análisis crítico de sus resultados, se identificaron brechas críticas que justifican el desarrollo de este proyecto. Las investigaciones anteriores han proporcionado una base sólida sobre la cual construir, pero también han identificado áreas donde se necesita más investigación y desarrollo. A continuación, se ofrece un análisis crítico de las obras revisadas y las razones concretas que justifican este proyecto.

La sugerencia de que el cifrado de red no tiene ninguna ventaja sobre el enrutamiento estándar de sesión única en redes no dirigidas es un tema de gran interés y debate en la comunidad académica. Aunque varios estudios han abordado este tema, los resultados no son concluyentes y a menudo están limitados por los métodos y supuestos utilizados.

Codificación de red de unidifusión múltiple: Este estudio investiga el impacto del cifrado de red en múltiples escenarios de unidifusión. Aunque esto proporciona una descripción general completa, no proporciona una respuesta definitiva sobre las ventajas del cifrado de red en comparación con el enrutamiento estándar. La metodología utilizada se basa en simulaciones y modelos teóricos y, si bien es sólida, no tiene en cuenta todos los factores prácticos que pueden afectar el rendimiento en un entorno del mundo real.

Multidifusión no dirigida: Se centra en problemas de multidifusión en redes no dirigidas y la aplicación de técnicas de cifrado de red. Aunque este resultado sugiere potencial de mejora, también muestra que es muy complejo desde el punto de vista computacional y operativo. La falta de evidencia en casos a gran escala y la dependencia de modelos teóricos limitan la aplicabilidad práctica de los resultados.

Algoritmos de aproximación: Los algoritmos de aproximación han demostrado ser herramientas efectivas para abordar problemas de enrutamiento y cifrado de redes, especialmente cuando la complejidad computacional es un problema. Sin embargo, los estudios revisados muestran que hay margen de mejora significativa. Uno de los puntos de partida de los algoritmos de aproximación en este campo es el algoritmo de Steiner Tree Packing. Este enfoque ha sido ampliamente estudiado y aplicado en diversos contextos de enrutamiento. Aunque proporciona una solución eficaz, la implementación real puede resultar compleja y costosa. Las investigaciones actuales sugieren que se necesitan nuevos enfoques que puedan simplificar la implementación sin sacrificar el rendimiento.

cFlow, mFlow, oFlow: Se han propuesto algoritmos de cifrado de red como cFlow, mFlow y oFlow como soluciones para mejorar la eficiencia del enrutamiento. Aunque estos algoritmos son prometedores, todavía enfrentan desafíos en términos de escalabilidad y complejidad. Es difícil comprender completamente sus ventajas y desventajas debido a la falta de estudios comparativos completos y pruebas en diferentes entornos. Teniendo en cuenta el análisis anterior, está claro que existe una necesidad urgente de realizar más investigaciones y desarrollar soluciones que aborden las limitaciones identificadas.

Este proyecto se justifica por las siguientes razones:

Contribución al conocimiento existente: Probar la especulación sobre los beneficios del cifrado de red sobre el enrutamiento estándar representa una contribución significativa al cuerpo de conocimiento existente. Los resultados obtenidos pueden aclarar la discusión y orientar futuras investigaciones en este campo.

Desarrollo del algoritmo de aproximación: La creación de algoritmos de aproximación basados en cFlow y otros algoritmos de cifrado de red puede proporcionar soluciones prácticas y eficientes a problemas de enrutamiento complejos. Este desarrollo puede reducir la complejidad computacional y mejorar la viabilidad de estas técnicas en aplicaciones del mundo real.

Relevancia práctica: Este proyecto no sólo tiene implicaciones teóricas, sino que también aborda cuestiones prácticas relevantes en el campo de las redes no dirigidas. La solución propuesta se puede aplicar en diversas situaciones y mejorar la eficiencia y eficacia de la red.

Innovación y originalidad: Este estudio propone un nuevo enfoque combinando técnicas de cifrado de red y algoritmos de enrutamiento, explorando así enfoques que aún no han sido ampliamente estudiados. Las innovaciones en esta área pueden abrir nuevas oportunidades para el desarrollo de tecnologías de red avanzadas.

4.2 OBJETIVOS

4.2.1 DEMOSTRACIÓN DE LA INEFICIENCIA DE LA CODIFICACIÓN DE RED PARA MULTIDIFUSIÓN DE SESIÓN ÚNICA EN REDES NO DIRIGIDAS

El primer objetivo del proyecto fue demostrar los beneficios adicionales de la codificación de red para multidifusión de sesión única en redes con bordes no dirigidos en comparación con el enrutamiento tradicional. Este objetivo se basa en la suposición aún válida de que el cifrado de red no aporta beneficios significativos de eficiencia o rendimiento a este tipo de redes.

Para lograr este objetivo, el proyecto, se va a llevar a cabo un análisis exhaustivo de la investigación existente sobre codificación de red y su aplicación a la multidifusión de sesión única en redes no dirigidas. En segundo lugar, el modelado teórico. Se desarrollará un modelo teórico que nos permite comparar el rendimiento de la codificación de red con los métodos de enrutamiento tradicionales en estos escenarios específicos. Por último, se harán simulaciones y experimentos en la red no dirigida para validar el modelo teórico y obtener datos empíricos sobre el rendimiento relativo de ambas técnicas.

4.2.2 DESARROLLO DE UN ALGORITMO DE APROXIMACIÓN BASADO EN ÁRBOLES DE STEINER Y CFLOW

El segundo objetivo del proyecto es desarrollar un algoritmo de aproximación que adapte la solución de cFlow al caso de árboles de Steiner con matrices de sensores comprimidas. Los árboles de Steiner son conocidos por su capacidad para minimizar el costo general de la red al conectar una gran cantidad de nodos, pero a costa de una mayor complejidad computacional. Sin embargo, ofrecen importantes beneficios en términos de eficiencia de enrutamiento y utilización de recursos.

El desarrollo de este algoritmo incluye las siguientes tareas: Análisis de cFlow e indicar sus fortalezas y limitaciones en el contexto del enrutamiento y cifrado de redes. En segundo lugar, un estudio de árboles Steiner, sus propiedades y sus aplicaciones en la optimización de redes. Después se diseñará el algoritmo que combina soluciones cFlow y propiedades del árbol Steiner. Este algoritmo utiliza matrices de compresión sensorial para reducir la complejidad computacional y permitir una implementación más eficiente. Para verificar el funcionamiento del algoritmo, se llevarán a cabo validaciones y pruebas. Para ello, se implementará el algoritmo en simulaciones de red y pruebas experimentales y evaluar su desempeño en comparación con enfoques tradicionales y otras técnicas de cifrado de red.

Estos objetivos son importantes para avanzar en el conocimiento y la práctica de la gestión de redes de datos, especialmente en el contexto de redes no dirigidas y técnicas avanzadas de enrutamiento. Se pretende demostrar la ineficiencia de la codificación de redes en escenarios específicos contribuye a una comprensión más profunda de sus limitaciones y orienta investigaciones futuras hacia enfoques más prometedores.

Por otro lado, se quere innovar en algoritmos de aproximación. El desarrollo de nuevos algoritmos que combinan árboles cFlow y Steiner con técnicas de compresión sensorial representa una innovación importante que puede mejorar la eficiencia y reducir la complejidad de la gestión de la red.

Los resultados de este proyecto se pueden aplicar a redes reales, proporcionando una solución más eficiente y manejable para el enrutamiento y optimización de redes de datos. Esto es extremadamente importante en la era de la conectividad global y el crecimiento de archivos.

4.3 METODOLOGÍA

El objetivo principal de esta sección es describir la metodología utilizada para lograr los objetivos del proyecto, como validar los supuestos de ineficiencia de codificación de red en sesiones de multiple-unicast y desarrollar algoritmos de aproximación basados en árboles cFlow y Steiner. Se utiliza un enfoque inductivo, comenzando con un ejemplo específico, analizando sus propiedades y generalizando los resultados a través de la teoría de redes. Este enfoque proporciona una comprensión profunda del comportamiento y las propiedades de los algoritmos que se estudian.

Proceso inductivo

Este proyecto analiza una red de ejemplo concreta y evalúa el flujo máximo que puede circular por sus bordes. Luego se aplicarán conceptos teóricos a estos ejemplos para sacar conclusiones generales sobre la eficiencia y viabilidad de los algoritmos de enrutamiento y codificación de red. Primero, se seleccionarán varias topologías de red que representan diferentes configuraciones y tamaños, desde redes simples hasta estructuras más complejas. Estos ejemplos incluyen redes dirigidas y no dirigidas con diferentes patrones de demanda de flujo. En cada ejemplo, el flujo máximo que puede circular a lo largo del borde se calcula utilizando algoritmos clásicos de flujo máximo, como el algoritmo de Ford-Fulkerson o el algoritmo de Edmonds-Karp. Se identificarán cuellos de botella y limitaciones estructurales que afectan los flujos dentro de la red.

Utilizando datos obtenidos de un ejemplo específico, se aplicarán los teoremas fundamentales de la teoría de flujo máximo y la teoría de corte mínimo para comprender mejor las limitaciones y capacidades de la red analizada.

La dualidad LP (programación lineal) se utilizará para analizar en profundidad el comportamiento del algoritmo y optimizar el rendimiento.

La primera etapa de la metodología implica la aplicación de algoritmos específicos al concepto de flujos de multicommodity y la aplicación de los teoremas de flujo máximo y corte mínimo. Los algoritmos de flujo de multicommodity se utilizarán para encontrar soluciones óptimas en redes. Estos algoritmos se aplicarán a ejemplos seleccionados para determinar cómo interactúan los diferentes flujos e impactan la capacidad de la red.

Se espera determinar cómo la coexistencia de múltiples flujos afecta la eficiencia del enrutamiento y la utilización de los recursos de la red. Los teoremas de flujo máximo y corte mínimo, como el teorema de Ford-Fulkerson, proporcionan una base teórica sólida para analizar y optimizar el flujo en redes.

Algoritmos de enrutamiento

Una vez que se comprenda el comportamiento del flujo de múltiples productos y se identifiquen los límites máximos de flujo, se aplicarán algoritmos de enrutamiento para optimizar la transferencia de datos a través de la red. Los algoritmos de enrutamiento tradicionales, como Dijkstra y Bellman-Ford, se utilizan primero para determinar la ruta óptima en términos de costo y distancia. Más adelante, se introducirán técnicas más avanzadas para mejorar la eficiencia del enrutamiento en escenarios más complejos. Estos algoritmos se evalúan por su capacidad para manejar grandes cantidades de tráfico y su adaptabilidad a los cambios en la topología de la red. La evaluación incluye una comparación de los resultados obtenidos con el algoritmo de flujo multicommodity y las observaciones teóricas derivadas de los teoremas de flujo máximo y corte mínimo.

Introducción a los conceptos de codificación de red

Se aplicarán algoritmos como cFlow y mFlow para evaluar cómo estas técnicas mejoran el rendimiento en muestras seleccionadas. En particular, el algoritmo cFlow se utiliza para comprender cómo se pueden utilizar las técnicas de codificación de red en redes no dirigidas y cómo se comparan con los métodos de enrutamiento tradicionales. En segundo lugar,

mFlow proporciona una perspectiva adicional sobre la aplicación de la codificación de red en redes con múltiples flujos simultáneos. Estos algoritmos se implementarán y evaluarán por su capacidad para mejorar la eficiencia computacional y el rendimiento de la red en comparación con los métodos de enrutamiento tradicionales.

Desarrollo y análisis de Duality LP para cFlow

Para comprender mejor el algoritmo cFlow, se desarrollará el Duality LP (Programación lineal dual). Este enfoque nos permite descifrar los detalles internos del algoritmo y obtener una comprensión más profunda de su comportamiento y limitaciones. La dualidad LP proporciona una perspectiva matemática rigurosa que puede revelar aspectos importantes de los algoritmos que no son obvios en implementaciones simples. Este análisis implica formular el problema de cableado como un problema de programación lineal y luego desarrollar su problema dual. Una comprensión profunda de este algoritmo permite aplicarlo de manera más efectiva a ejemplos seleccionados y optimizar el rendimiento en términos de costo computacional y eficiencia del cableado.

Evaluación y comparación de los costos computacionales de algoritmos

Uno de los principales objetivos de este proyecto es evaluar los costos computacionales de cada algoritmo y compararlos en términos de eficiencia y rendimiento. Para ello se realizarán pruebas exhaustivas sobre ejemplos seleccionados, aplicando cada algoritmo y midiendo el tiempo de ejecución, el consumo de recursos y la calidad de las soluciones obtenidas. Estas pruebas incluyen tanto algoritmos de enrutamiento tradicionales como técnicas avanzadas de codificación de red. Los resultados obtenidos permiten una comparación directa de los costos computacionales y los beneficios de rendimiento de cada enfoque. Esta evaluación incluye análisis estadístico y comparativo para identificar patrones y tendencias que sirvan como base para desarrollar futuras mejoras y optimizaciones del algoritmo.

Capítulo 5. MODELO DESARROLLADO

5.1 APLICACIÓN DE ALGORITMOS DE RUTEO Y CODIFICACIÓN DE RED EN MÚLTIPLES ESCENARIOS

En primer lugar, se van a analizar ejemplos específicos que nos permiten observar las diferencias entre los algoritmos de enrutamiento tradicionales y las técnicas de codificación de redes. Estos ejemplos cubren diferentes topologías de red y patrones de tráfico para evaluar cómo cada enfoque maneja el enrutamiento de datos y optimiza el uso de los recursos de la red. Inicialmente se utilizan algoritmos de enrutamiento como Dijkstra y Bellman-Ford para determinar rutas óptimas en términos de costo y distancia.

A continuación, se introducen algoritmos de codificación de red como cFlow y mFlow para comparar su eficiencia y eficacia en los mismos escenarios. Estos ejemplos miden la corriente máxima que circula a través de los bordes de la red e identifican cuellos de botella y restricciones estructurales. Se evalúa el efecto de la coexistencia de múltiples flujos y se aplican los teoremas de flujo máximo y desplazamiento mínimo para comprender mejor las características y limitaciones de cada enfoque.

El análisis evalúa los costos computacionales de cada algoritmo y la calidad de las soluciones resultantes en términos de eficiencia de enrutamiento y utilización de recursos. Este enfoque comparativo identifica las ventajas y desventajas de los algoritmos de enrutamiento tradicionales y las técnicas de codificación de red, proporcionando una base sólida para desarrollar un algoritmo de aproximación que combine las fortalezas de ambos enfoques.

Primer escenario

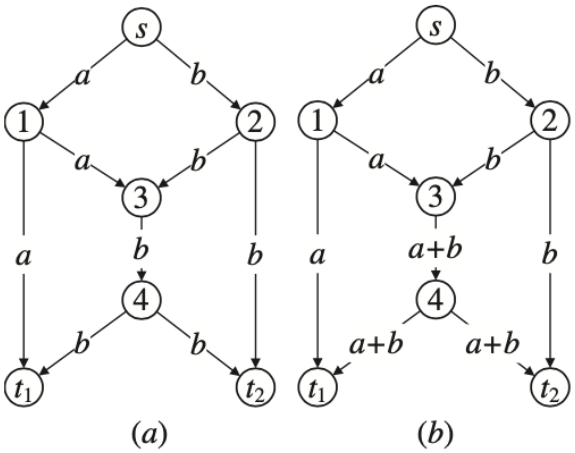


Figura 9: topología de butterfly. Fuente: [5]

La figura anterior ilustra un ejemplo clásico de cómo la codificación de red ayuda a mejorar el rendimiento de extremo a extremo. Como t1 recibe tanto a como a + b (codificados sobre GF(2) [8]), es capaz de descodificar y recuperar tanto a como b. Si las capacidades de enlace son 1, resulta evidente que el rendimiento máximo alcanzable con codificación de red es 2. Sin codificación, se puede calcular que el rendimiento óptimo es 1,875. Si sólo se utiliza un árbol de multidifusión (como en la multidifusión IP), el rendimiento alcanzado es 1.

En el ejemplo anterior, cada enlace tiene capacidad unitaria. El nodo s se asigna como nodo de origen donde se distribuirá la información, mientras que los nodos t1 y t2 son los destinos del sumidero. Consideremos el caso en el que queremos transmitir 2 bits, bit a y bit b, a los nodos t1 y t2. La figura (a) muestra la estrategia de encaminamiento tradicional que podemos utilizar sin codificación de red, y la figura (b) muestra el caso con codificación de red. Tanto en el caso de codificación de red como en el tradicional, senvía su paquete a los nodos t1 y t2. Sin embargo, la diferencia de rendimiento se produce en el nodo 3. Sin la codificación de red, el nodo 3 tendrá que hacer dos rondas para entregar todos los paquetes recibidos al nodo 4. En este caso, s enviará su paquete a los nodos t1 y t2. Los paquetes son enviados al nodo 4. El nodo 4 podrá entonces transmitir a los dos nodos de destino. Sin embargo, con la codificación de red, el nodo 3 podría transmitir sólo un paquete (a+b) tanto a t1 y t2. Por

tanto, en este ejemplo de mariposa dirigida se podría encontrar una ventaja de codificación de 2.

Vamos a analizar este ejemplo como si fuera un caso de multicommofity flow. Simplemente con mirar el ejemplo, podemos observar que si queremos hacer pasar el máximo flujo por las dos aristas iniciales y llegan a destino, el grafo residual ya mostraría que después del primer paso, esas flechas no permitirían el paso del nodo origen a los siguientes, por lo que ya habríamos encontrado el flujo máximo, que en este caso es 2.

No obstante, si aplicamos el algoritmo de cFlow, que en palabras mundanas consiste en ver los caminos que hay de origen a destino y tratarlos por separado, vemos que hay dos caminos que salen del origen al destino, por lo que si los tratamos por separado, como le llegan dos flujos a cada destino, inmediatamente sabemos que el flujo máximo es 2, que es lo que se explica con ecuaciones en E. 1.

Por otro lado, ¿Si aplicáramos el algoritmo de los árboles de Steiner, qué resultado obtendríamos? He aquí el inicio de lo que se comentaba en el inicio de este proyecto, en cuanto a la complejidad de los algoritmos. Si destripamos este ejemplo para conseguir todos los árboles de Steiner, vemos que encontrar todos los árboles tiene una complejidad elevada, y se necesita mucha computación. En este caso, los libros demuestran que existen 17 árboles diferentes. Durante mi estudio de este caso, encontré solo 11, pero conseguí llegar al mismo rendimiento óptimo (1.85). En la siguiente figura se muestra la comparativa de rendimiento dependiendo del tipo de procedimiento seguido, si árboles de Steiner o network coding, para este caso concreto. A la derecha, podemos ver el número de árboles encontrado por los libros, y el beneficio network coding/routing. |V| es el número de vértices, |M| el número de nodos origen+destino, y |E| es el número de aristas.

$\|V\|$	$\|M\|$	$\|E\|$	$\chi(N)$	$\pi(N)$	$\frac{\chi(N)}{\pi(N)}$	# of trees
7	3	9	2	1.875	1.067	17

Figura 10: tabla resumen de rendimientos. Fuente: [6]

Ahora vamos a demostrar el resultado del rendimiento obtenido con el algoritmo de los árboles de Steiner. En primer lugar, tenemos que encontrar todos los caminos que nos llegan de origen a destino por los diferentes caminos que existen en la red.

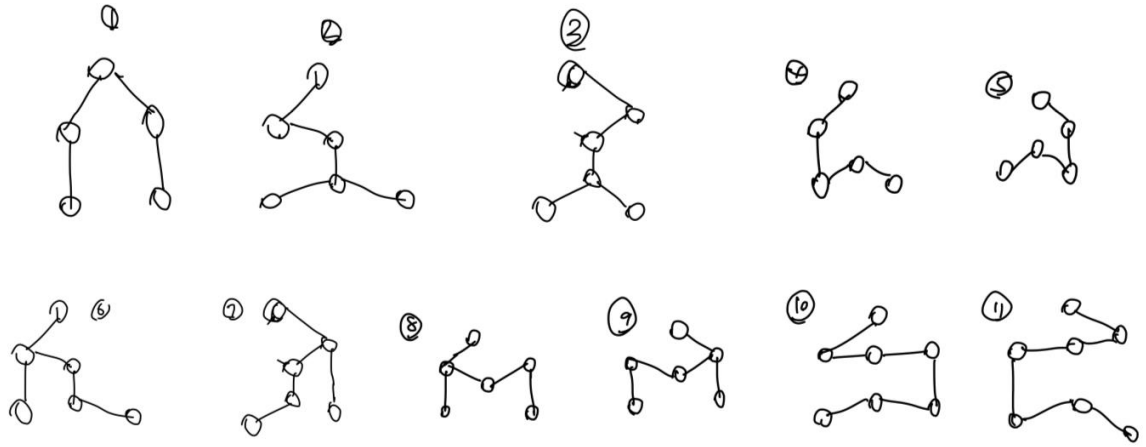


Figura 11: árboles de Steiner. Fuente: elaboración propia

En la figura anterior se muestran los 11 árboles de Steiner. El procedimiento es el siguiente, vamos a nombrar con $x_1, x_2, \dots, x_n$ aquellas aristas que son compartidas por los distintos árboles. Por ejemplo, del nodo origen al nodo 1, hay 6 árboles que tienen esa arista en su árbol, por lo que la primera condición debe ser:

E. 19 $$x_1 + x_2 + x_4 + x_6 + x_8 + x_{10} \leq 1$$

Si seguimos con este procedimiento llegamos al siguiente LP:

Maximizar: $Z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11}$

Sujeto a:

E. 20 $x_1 + x_2 + x_4 + x_6 + x_8 + x_{10} \leq 1$

E. 21 $x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} \leq 1$

E. 22 $x_2 + x_6 + x_8 + x_9 + x_{10} + x_{11} \leq 1$

E. 23 $x_3 + x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 1$

E. 24 $x_1 + x_4 + x_6 + x_8 + x_9 + x_{10} + x_{11} \leq 1$

E. 25 $x_2 + x_3 + x_6 + x_7 \leq 1$

E. 26 $x_1 + x_5 + x_7 + x_8 + x_9 + x_{11} \leq 1$

E. 27 $x_2 + x_3 + x_4 + x_5 + x_7 + x_{10} + x_{11} \leq 1$

E. 28 $x_2 + x_3 + x_4 + x_5 + x_6 + x_{10} + x_{11} \leq 1$

$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}$ *no restringidos en signo.*

Si resolvemos este sistema (Python), obtenemos los siguientes pesos para los árboles:

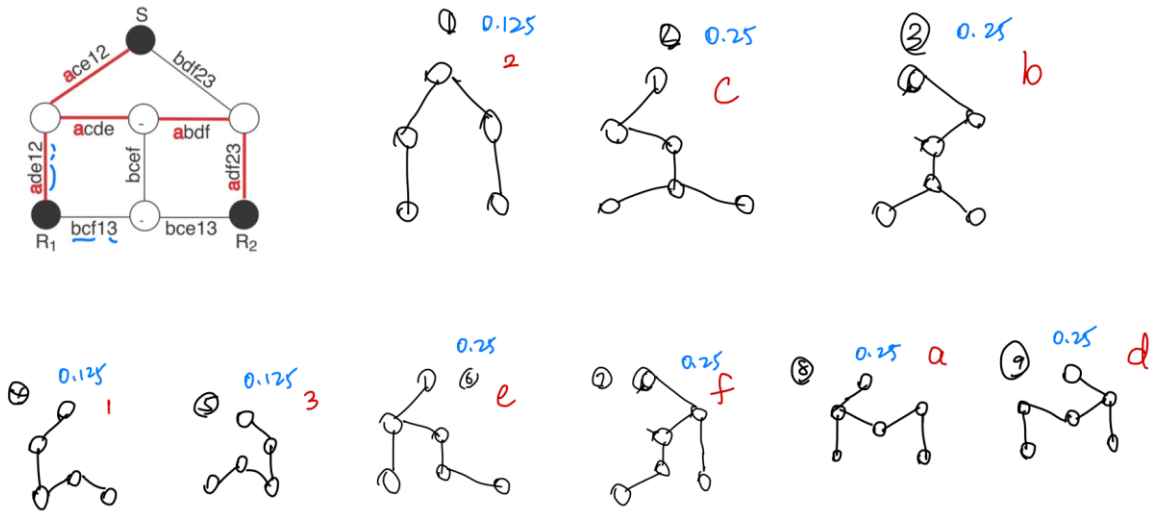


Figura 12: árboles de Steiner. Fuente: elaboración propia

Si vemos la distribución de los árboles, podemos observar que llegamos a la misma solución (1.85) tan solo usando 9 de los 17 árboles que decían los libros (yo encontré 11).

Por lo tanto, vemos que en este ejemplo si lo analizamos con el algoritmo de los árboles de Steiner obtenemos un rendimiento de 1.85, si lo analizamos con Ford-Fulkerson obtenemos un rendimiento de 2, y si lo analizamos con network coding con el análisis de símbolos

obtenemos un rendimiento de 2. No obstante, estos resultados los analizaremos en el siguiente capítulo.

Segundo escenario

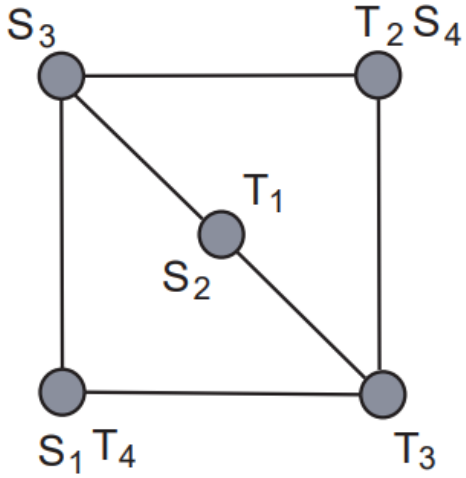


Figura 13: segundo escenario. Fuente: [9]

Empezamos analizando este ejemplo mediante algoritmos de routing. En primer lugar, desglosamos en árboles la red para ver el número total y los distintos caminos que existen entre el origen y los destinos. Llevando a cabo este análisis, llegamos a lo siguiente:

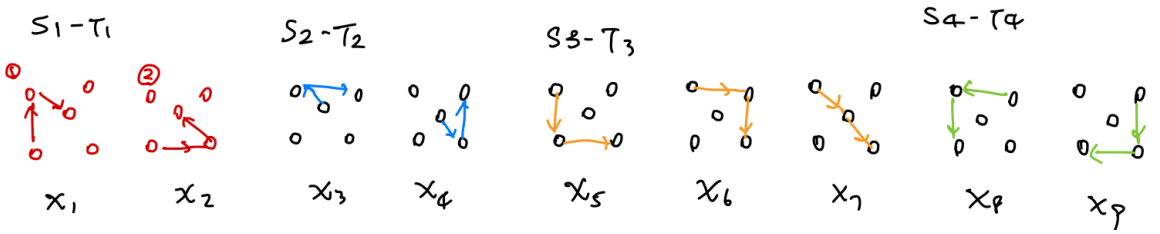


Figura 14: árboles de Steiner del segundo escenario. Fuente: elaboración propia

En la figura anterior se muestran todos los árboles encontrados, mediante los cuales montaremos un sistema de ecuaciones similar al del ejemplo anterior.

$Maximizar$: $x_1 + x_2$

$Sujeto\ a$:

E. 29 $x_3 + x_6 + x_8 \leq 1$

E. 30 $x_1 + x_5 + x_8 \leq 1$

E. 31 $x_2 + x_5 + x_9 \leq 1$

E. 32 $x_4 + x_6 + x_9 \leq 1$

E. 33 $x_1 + x_3 + x_7 \leq 1$

E. 34 $x_2 + x_4 + x_7 \leq 1$

E. 35 $x_1 + x_2 = x_3 + x_4$

E. 36 $x_5 + x_6 + x_7 = x_1 + x_2$

E. 37 $x_8 + x_9 = x_1 + x_2$

Introduciendo el modelo en Python obtenemos lo siguiente:

```
Optimal Target Flow Rate (f*): 0.7499999995001766
x1: 0.37499999975008813
x2: 0.37499999975008824
x3: 0.37499999975008813
x4: 0.3749999997500882
x5: 0.24999999983339213
x6: 0.2499999998333922
x7: 0.24999999983339208
x8: 0.3749999997500882
x9: 0.37499999975008813
```

Figura 15: flow rate obtenido

Estos resultados son obtenidos mediante el algoritmo de los árboles de Steiner, que es un algoritmo de ruteo. El valor que se encuentra a la derecha de las variables es el peso que se le ha dado a cada árbol.

A continuación, se mostrará el mismo procedimiento salvo que en esta ocasión utilizaremos un algoritmo propio de network coding, y al ser múltiples orígenes y destinos, usaremos mFlow, ya que es más práctico para casos de múltiples sesiones.

Si resolvemos el problema con el algoritmo mFlow, obtenemos los siguientes resultados:

```
{0, 1, 2, 3, 4}
{(4, 0), (0, 4), (2, 1), (3, 4), (4, 3), (1, 2), (2, 0), (1, 4), (2, 3), (0, 2), (3, 2), (4, 1)}
Optimal Target Flow Rate (f*): 0.7499999999642175
```

Figura 16: flow rate con mFlow

Como se puede observar, el rendimiento logrado en el escenario de multidifusión multisesión es comparable al uso de algoritmos de enrutamiento tradicionales y técnicas de codificación de red. Este resultado sugiere que la codificación de red no ofrece ventajas significativas en términos de capacidad de transmisión en comparación con los métodos de enrutamiento estándar en la situación específica de redes no dirigidas. Esta observación es importante porque valida nuestra conjetura inicial de que la codificación de red no mejora el rendimiento en ciertos escenarios de red.

Tercer escenario

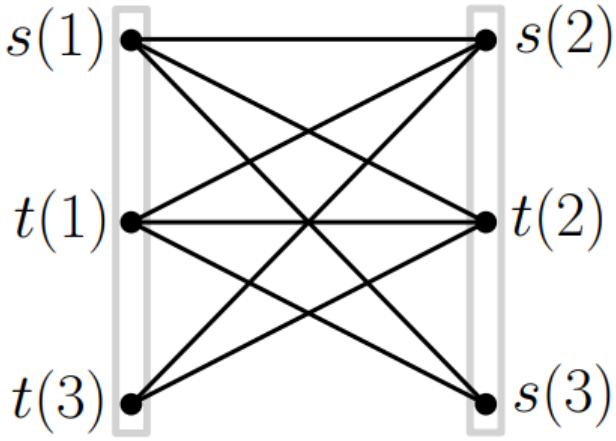


Figura 17: tercer escenario. Fuente: [7]

Este caso es incluso más particular que los anteriores, por el número de aristas que tiene y la complejidad que existe entre los emisores y receptores. No obstante, vamos a seguir el mismo procedimiento que en los casos anteriores, aplicando primero el algoritmo de los árboles de Steiner y seguidamente mFlow o cFlow.

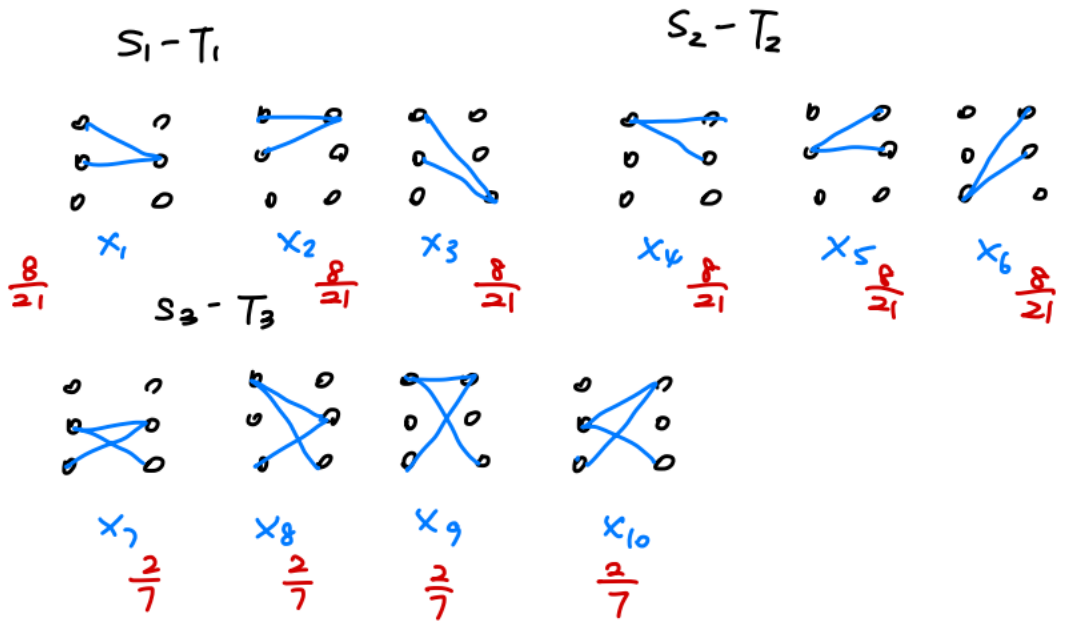


Figura 18: árboles de Steiner. Fuente: elaboración propia

En la figura anterior se muestran todos los árboles que a priori he encontrado. Volcando la figura anterior en ecuaciones como se demostró en el primer ejemplo, obtenemos el siguiente sistema:

$Maximizar: x_1 + x_2 + x_3$

Sujeto a:

E. 38 $x_2 + x_4 + x_9 \leq 1$

E. 39 $x_1 + x_4 + x_8 \leq 1$

E. 40 $x_3 + x_8 + x_9 \leq 1$

E. 41 $x_2 + x_5 + x_{10} \leq 1$

E. 42 $x_1 + x_5 + x_7 \leq 1$

E. 43 $x_3 + x_7 + x_{10} \leq 1$

E. 44 $x_6 + x_9 + x_{10} \leq 1$

E. 45 $x_6 + x_7 + x_8 \leq 1$

De nuevo, si resolvemos este sistema con el solver, obtenemos el siguiente resultado:

```
Optimal Target Flow Rate (f*): 1.1428571423306586
x1: 0.35714285697833076
x2: 0.3571428569783307
x3: 0.42857142837399687
x4: 0.3571428569783307
x5: 0.35714285697833076
x6: 0.42857142837399687
x7: 0.28571428558266454
x8: 0.28571428558266465
x9: 0.28571428558266454
x10: 0.28571428558266465
```

Figura 19: flow rate obtenido

En la figura anterior se muestran tanto los pesos de los árboles como el rendimiento óptimo.

Por otro lado, si aplicamos el algoritmo de mFlow, que viene mejor al ser multisesión, obtenemos los siguientes resultados:

```
Optimal Target Flow Rate (f*): 1.1428571428543588
```

Figura 20: flow rate con mFlow

Como se puede observar, ha vuelto a coincidir el rendimiento independientemente del tipo de algoritmo utilizado. En este punto acaba la parte experimental con ejemplos y vamos a continuar con la elaboración del algoritmo de aproximación. No obstante, después de estos tres ejemplos se ha podido demostrar que en ciertas tipologías, precisamente las que se han expuesto, el network coding no mejora el rendimiento comparado con los algoritmos de routing. El problema ahora radica en lo siguiente, los algortimos de ruteo son muy costosos en términos de computación, ya que el número de árboles que se tienen que encontrar crece

de forma exponencial con el número de ejes de la topología de la red. Los casos expuestos son casos muy sencillos. Sin embargo, si queremos escalar estas tipologías a casos reales grandes, el algoritmo de los árboles de Steiner sería imposible de aplicar, Es por ello, que en algunas ocasiones es muy beneficioso aplicar el network coding, pero para casos de aristas bidireccionales y en multisesión, el network coding no mejora la velocidad de transmisión. Para salvar estas peculiaridades, en el siguiente apartado se va a elaborar un algoritmo de aproximación que, partiendo de una solución propia de cFlow o de mFlow, sea el input de un algoritmo de árboles de Steiner, de forma que esos árboles que se tienen que encontrar ya no son necesarios, ya que partiendo de la solución de un algoritmo de network coding, ya podemos saber qué arboles van a tener un peso distinto de cero. Con esto, eliminamos la complejidad elevada de la búsqueda de árboles y tenemos un algoritmo eficaz aplicable a todos los ejemplos que se deseen.

5.2 ALGORITMO DE APROXIMACIÓN BASADO EN ÁRBOLES DE STEINER Y CFLOW

A partir de una solución basada en cFlow, se ha desarrollado un algoritmo innovador que utiliza una matriz de sensores comprimida para transformarla y finalmente introducirla en un algoritmo de árbol de Steiner. El objetivo principal de este enfoque es eliminar la complejidad asociada con los métodos tradicionales y mejorar la eficiencia del enrutamiento en las redes de comunicación.

A continuación, se detalla el desarrollo y fundamento de este algoritmo, así como los beneficios que aporta en términos de rendimiento y optimización de recursos. Como hemos visto anteriormente, cFlow es un algoritmo de codificación de red que optimiza la distribución de flujos en una red, permitiendo que los datos se mezclen en nodos intermedios en lugar de simplemente reenviarse. Esto aumenta la capacidad de la red y mejora la eficiencia en determinadas topologías y condiciones de tráfico. Sin embargo, la codificación de la red también implica una importante sobrecarga computacional, lo que puede ser un

cuello de botella en redes grandes o con recursos limitados. Por ello, se introduce el concepto de la detección comprimida, que es una técnica para reducir la cantidad de datos necesarios para representar eficientemente una señal.

Como parte del algoritmo, se utilizará una matriz de compresión sensorial para transformar las soluciones obtenidas de cFlow a un formato más manejable y menos intensivo desde el punto de vista computacional. Esta transformación nos permite preservar las propiedades beneficiosas de la codificación de red al tiempo que reduce la complejidad asociada con la implementación de los árboles de Steiner.

Por otro lado, hemos visto que el algoritmo de los árboles de Steiner es una herramienta clásica en teoría de grafos y optimización combinatoria que se utiliza para encontrar el camino más corto o más barato que conecta un conjunto de nodos en una red. Los árboles de Steiner pueden proporcionar una solución muy eficiente, pero son computacionalmente costosos, especialmente cuando se trata de redes grandes y densas. El desafío es encontrar formas de explotar los árboles de Steiner sin incurrir en altos costos computacionales. Por esta razón, si combinamos cFlow, matrices de compresión sensorial y los árboles de Steiner, podemos obtener una solución computacionalmente eficiente y con un rendimiento óptimo considerable.

5.2.1 DISEÑO

El algoritmo desarrollado en este trabajo se divide en tres fases principales:

1. **Generación de solución cFlow**: En esta fase inicial, el algoritmo cFlow se aplica a la red para obtener una solución inicial de codificación de red. Este paso implica la combinación y distribución óptima de los flujos de datos dentro de la red, maximizando así el uso del ancho de banda disponible y optimizando la capacidad de la red.
2. **Transformación utilizando una matriz de detección comprimida**: La solución obtenida en cFlow se transforma utilizando una matriz de compresión sensorial. Esta

matriz reduce la dimensionalidad de los datos y simplifica la representación del flujo al tiempo que preserva características importantes de la solución original. Este paso es importante para reducir la complejidad computacional y preparar los datos para ingresarlos al algoritmo del árbol de Steiner.

3. **Implementación en el Algoritmo del Árbol Steiner**: Finalmente, la solución transformada se introduce en el Algoritmo del Árbol Steiner. Este paso explota la eficiencia de los árboles de Steiner para encontrar la ruta de enrutamiento óptima utilizando la solución simplificada obtenida en la fase anterior.

El resultado es una solución de enrutamiento que combina la eficiencia de la codificación de red con la precisión y optimización de los árboles de Steiner.

5.2.2 IMPLEMENTACIÓN

El algoritmo utilizado en el código (Anexo II) se centra en optimizar el flujo de datos en una red, aplicando técnicas avanzadas como la informática de redes sociales y los árboles de Steiner. A continuación, se presenta el funcionamiento del algoritmo y cómo se integran estos conceptos para solucionar el problema de flujo en una red de tipo "butterfly".

En primer lugar, se establece la forma de la red. En este caso, se dispone de una lista de nodos y aristas que indican cómo están conectados los nodos. Cada extremo tiene una capacidad asociada, lo cual indica el flujo máximo de datos que puede transportar. Asimismo, se identifican los nodos fuente y receptores, que son los puntos de entrada y salida del flujo de datos en la red.

La red se puede representar mediante gráficos. Se establece un grafo no dirigido y otro no dirigido. El grafo no dirigido describe todas las conexiones posibles entre nodos sin tener en cuenta la dirección del flujo. Se convierte en un grafo dirigido, en el que las conexiones tienen una dirección específica, lo cual representa el flujo de datos desde un nodo hacia otro.

Para cada receptor, se establecen todas las opciones posibles desde la fuente. Esto se realiza mediante una función que encuentra todas las rutas sencillas en el grafo que se dirigen desde el nodo fuente hasta cada receptor. Esta información es esencial para los pasos siguientes,

ya que el algoritmo requiere comprender todas las formas en que los datos pueden fluir desde la fuente hasta los receptores.

El objetivo principal del algoritmo es la determinación del flujo óptimo mediante la aplicación de redes. Se trata de un problema de programación convexa en el cual se incrementa el flujo total que puede ser enviado desde la fuente a los receptores, respetando las habilidades de las aristas y asegurando que el flujo que entra y sale de cada nodo sea equilibrado para los nodos que no son fuente ni receptores.

En esta etapa, se establecen diversas restricciones:

- Las capacidades de las aristas deben ser respetadas.
- El flujo en cada arista debe no ser negativo.
- Para cada receptor y arista, el flujo no deberá exceder la capacidad requerida por la arista.
- La velocidad entre la entrada y la salida debe ser igual para los nodos intermedios.

El propósito es incrementar la velocidad de flujo desde la fuente hasta los receptores, asegurando que todas las limitaciones se cumplan.

Un momento después de calcular el flujo óptimo, se generan matrices de compresión para cada receptor. Estas matrices son fundamentales para comprender cómo se distribuye el flujo a través de diversas conexiones en la red. Cada matriz dispone de dos dimensiones distintas:

- El eje X indica el número de pasos desde la fuente hasta el receptor.
- El eje Y indica el número de aristas que se encuentran en la red.

Cada elemento de la matriz indica si una arista específica se encuentra en un sendero determinado. Por ejemplo, si la arista (u, v) se encuentra presente en el trayecto i, la entrada correspondiente a la matriz será 1; de lo contrario, será 0. Estas matrices brindan la oportunidad de contemplar de forma clara y estructurada cómo se distribuyen los flujos a través de la red.

Al emplear las matrices de compresión, se optimizan los flujos de datos para cada receptor. Se trata de hallar la forma más eficiente de distribuir el flujo a través de los caminos posibles, aumentando el flujo total que puede ser enviado a cada receptor sin exceder las capacidades de las aristas.

Se presenta otro problema de programación convexa en el cual se incrementa la suma de los flujos a través de las aristas, sujeto a la limitación de que el flujo en cada arista no exceda la capacidad calculada previamente.

Después de optimizar los flujos, se identifican los caminos significativos, es decir, aquellos que tienen un flujo no nulo. Esto es relevante debido a que se simplifica la estructura de la red al enfocarse solo en los caminos que realmente transportan datos, eliminando aquellos que no contribuyen al flujo total.

En última instancia, se generan combinaciones de caminos significativos para formar árboles de Steiner, que son subgrafos que conectan la fuente con los receptores de manera eficaz. Se generan todos los posibles árboles de Steiner, combinando los caminos relevantes para cada receptor. Estos árboles simbolizan las estructuras adecuadas de conexión en la red con el fin de optimizar el flujo de datos.

Se emplea un último paso de programación lineal con el fin de encontrar la combinación más adecuada de estos árboles de Steiner que maximice el flujo total en la red. Se asignan pesos a cada árbol y se ajusta la suma de estos pesos, asegurando que el flujo total a través de cualquier arista no exceda su capacidad.

Capítulo 6. ANÁLISIS DE RESULTADOS

6.1 APLICACIÓN DE ALGORITMOS DE RUTEO Y CODIFICACIÓN DE RED EN MÚLTIPLES ESCENARIOS

Primer escenario:

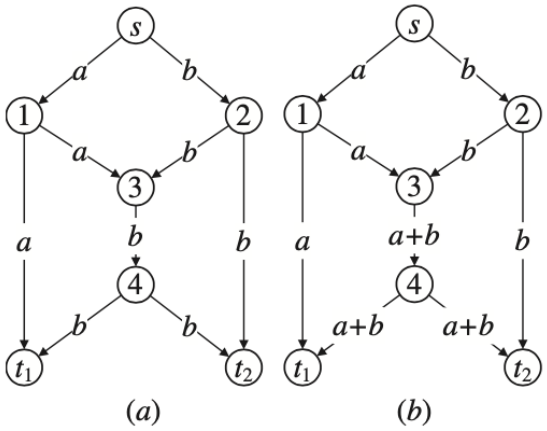


Figura 21: topología de butterfly. Fuente:[5]

En el análisis de este primer caso de la tipología mariposa, se podemos ver la comparación en términos de velocidad de transmisión según los distintos procedimientos utilizados para la distribución de información en la red. Recordamos los parámetros clave de la tipología:

$\|V\|$	$\|M\|$	$\|E\|$	$\chi(N)$	$\pi(N)$	$\frac{\chi(N)}{\pi(N)}$	# of trees
7	3	9	2	1.875	1.067	17

Figura 22: resultado de rendimientos: Fuente: elaboración propia

- |V| indica el número de vértices en la red.
- |M| indica el número de fuentes.
- |E| indica el número de aristas.

- $\chi(N)$ representa la velocidad de transmisión con network coding.
- $\pi(N)$ representa la velocidad de transmisión con árboles de Steiner.
- $\chi(N)/\pi(N)$ relación entre los rendimientos.
- # indica el número de árboles

El proceso de optimización identificó 11 árboles Steiner en nuestra implementación, pero la tabla indica que puede haber hasta 17 árboles. Hay varias razones posibles para esta discrepancia.

- Pesos de árboles: Muchos de los 17 árboles identificados en la tabla tienen pesos de 0. Esto significa que no contribuyen al flujo general. Estos árboles se pueden eliminar con algunas optimizaciones, simplificando así el conjunto de soluciones relacionadas.
- Complejidad computacional: Identificar todos los árboles de Steiner posibles es una tarea computacionalmente intensiva. Es posible que el algoritmo utilizado solo haya optimizado los árboles más relevantes o los árboles con pesos significativos y haya omitido los árboles con pesos cero.
- Limitaciones del Algoritmo: El algoritmo implementado puede tener ciertas limitaciones o criterios de selección que limiten el número de árboles considerados. Estos criterios pueden diseñarse para mejorar la eficiencia computacional o centrarse en soluciones más prácticas.

Los resultados muestran que, aunque la codificación de la red no mejoró la velocidad de transmisión en este caso particular, el análisis detallado del árbol de Steiner proporciona información valiosa sobre la estructura de la red y las posibles rutas de transmisión. Al identificar y comprender los árboles de Steiner, puede optimizar su red de manera más eficiente eliminando rutas redundantes y centrándose en las rutas que realmente transmiten datos.

Segundo escenario:

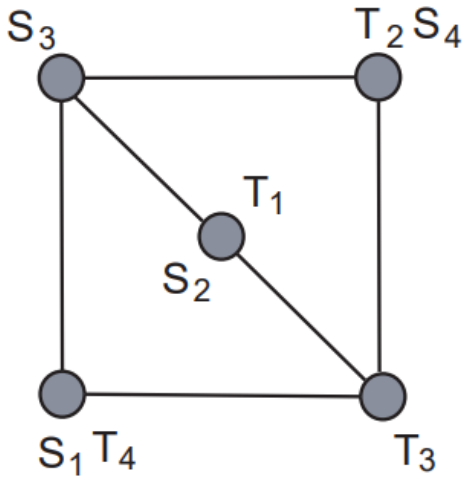


Figura 23: segundo escenario: Fuente: [9]

En el segundo escenario, hay cuatro nodos origen y cuatro nodos sumidero. Se evaluaron dos métodos: árbol de mFlow y árbol de Steiner para múltiples sesiones de unidifusión con bordes bidireccionales. El rendimiento obtenido con ambos métodos fue de 0.74. Esta es una prueba más de que la codificación de red no aporta ningún beneficio de rendimiento a este tipo de red.

El análisis del segundo escenario refuerza la idea de que la codificación de red no siempre proporciona beneficios de rendimiento, especialmente en ciertas configuraciones de red, como múltiples sesiones de unidifusión con bordes bidireccionales. Tanto el enfoque mFlow como el Steiner Tree lograron un rendimiento de 0.74. Esto sugiere que las técnicas de optimización tradicionales son igualmente efectivas para este tipo de red. Esto resalta la importancia de evaluar cuidadosamente las características de la red y los requisitos de transmisión antes de decidir implementar la codificación de red. La elección del método de optimización debe basarse en un análisis detallado del tipo de red y las condiciones de transmisión específicas. Esto se debe a que la implementación de la codificación de red puede no justificar el esfuerzo adicional en todos los casos.

```
Optimal Target Flow Rate (f*): 0.7499999995001766
x1: 0.37499999975008813
x2: 0.37499999975008824
x3: 0.37499999975008813
x4: 0.3749999997500882
x5: 0.24999999983339213
x6: 0.2499999998333922
x7: 0.24999999983339208
x8: 0.3749999997500882
x9: 0.37499999975008813
```

Figura 24: flow rate obtenido con routing

```
{0, 1, 2, 3, 4}
{(4, 0), (0, 4), (2, 1), (3, 4), (4, 3), (1, 2), (2, 0), (1, 4), (2, 3), (0, 2), (3, 2), (4, 1)}
Optimal Target Flow Rate (f*): 0.7499999999642175
```

Figura 25: flow rate con mFlow

Tercer escenario:

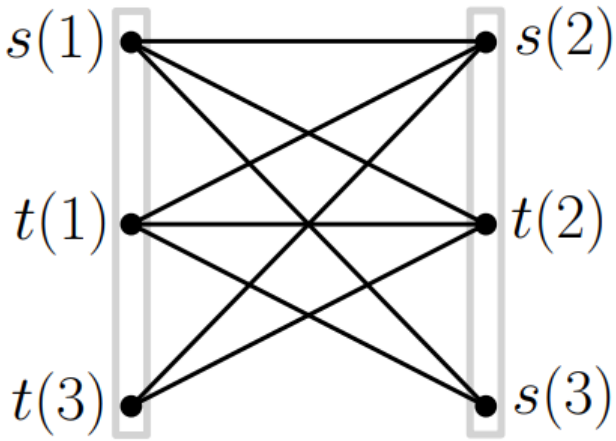


Figura 26: tercer escenario. Fuente: [7]

El análisis del tercer escenario confirma que, para una configuración de red con tres nodos de origen y tres nodos de destino y bordes bidireccionales, la codificación de red no proporciona ningún beneficio de rendimiento adicional. Tanto el método mFlow como el Steiner Tree lograron un rendimiento de 1.14. Esto sugiere que las técnicas tradicionales de optimización de rutas son igualmente efectivas para maximizar el rendimiento en este tipo de red.

A pesar de que se encontraron 10 árboles de Steiner, lo que demuestra la eficiencia del método en este caso específico, es importante reconocer el elevado costo computacional asociado con la identificación y construcción de estos árboles. A gran escala, este enfoque puede no ser práctico debido a los recursos computacionales necesarios.

Este resultado resalta la importancia de considerar las características específicas de la red y los requisitos de transmisión antes de implementar la codificación de red. Aunque la codificación de red puede potencialmente optimizar la utilización de la capacidad de la red en ciertos escenarios, este no es siempre el caso, especialmente en redes con múltiples sesiones de unidifusión y bordes bidireccionales.

6.2 ALGORITMO DE APROXIMACIÓN BASADO EN ÁRBOLES DE STEINER Y CFLOW

Para validar la eficacia y eficiencia de nuestro algoritmo de aproximación, hemos realizado pruebas de implementación en distintos escenarios. A continuación, se presentan los ejemplos analizados y los resultados obtenidos.

Para empezar, en la siguiente imagen se muestran los resultados obtenidos de los distintos ejemplos:

Network	$\|V\|$	$\|M\|$	$\|E\|$	$\chi(N)$	$\psi(N)$	$\pi(N)$	$\pi(N)/\psi(N)$	# of trees
C(3,2)	7	4	9	2	1.5	1.8	1.2	26
C(4,3)	9	5	16	3	2	2.667	1.333	1,113
C(4,2)	11	7	16	2	1.333	1.778	1.333	1,128
C(5,4)	11	6	25	4	2.5	3.571	1.428	75,524
C(5,2)	16	11	25	2	1.25	1.786	1.428	119,104

Figura 27: cuadro solución de rendimientos. Fuente: elaboración propia

ψ(N) representa el rendimiento obtenido con el algoritmo. Como se puede observar, obtenemos un rendimiento que se encuentra entre el obtenido con el algoritmo de los árboles de Steiner y cFlow. Se han hecho pruebas con cinco ejemplos, mediante los cuales el algoritmo ha mejorado con el entrenamiento del mismo, hasta llegar a una ratio de aproximación de 1.428, y el encontrado en estudios oficiales de algoritmia de telecomunicaciones fue de 1.55 [6], por lo que considero que ha sido un milestone haber podido desarrollar en tan poco tiempo un algoritmo eficaz que se aproxime tanto a lo descubierto mundialmente.

Para explicar el funcionamiento y, sobre todo, la estructura de la matriz intermediaria, se van a mostrar los outputs del algoritmo únicamente para el primer caso, que es el que más sencillo computacionalmente, ya que de otra forma este proyecto podría ser eterno.

6.2.1 C(3,2)

En este caso, el algoritmo nos ha brindado un rendimiento de 1.5, mientras que cFlow nos ha dado 2 y los árboles de Steiner 1.8. Aprovecho esta ocasión para señalar que esta topología que es la más sencilla de los casos probados, tiene ni más ni menos que 18 árboles de Steiner, es decir, computacionalmente ya es carga notable. Si nos fijamos en la tabla de resultados, podemos ver que el último ejemplo probado tiene 119104 árboles de Steiner, ¡y mediante el algoritmo hemos resuelto ese problema!

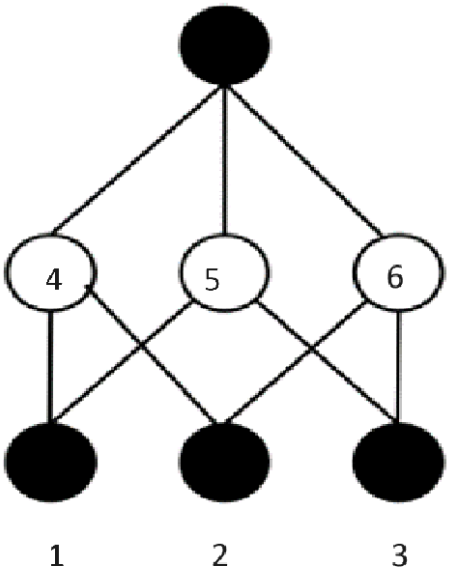


Figura 28: C(3,2). Fuente: elaboración propia

Sin más dilación, vamos a analizar los resultados que nos ha proporcionado el algoritmo:

El ejemplo tiene los siguientes nodos y aristas:

Nodos: [0, 1, 2, 3, 4, 5, 6]

Aristas: {(6, 2), (4, 0), (0, 4), (2, 4), (1, 5), (5, 1), (4, 2), (0, 6), (1, 4), (5, 0), (2, 6), (0, 5), (6, 0), (5, 3), (3, 6), (6, 3), (4, 1), (3, 5)}

Caminos desde el nodo 0 al nodo 1:

[[0, 4, 1], [0, 4, 2, 6, 3, 5, 1], [0, 5, 1], [0, 5, 3, 6, 2, 4, 1], [0, 6, 2, 4, 1], [0, 6, 3, 5, 1]]

Caminos desde el nodo 0 hasta el nodo 2:

[[0, 4, 1, 5, 3, 6, 2], [0, 4, 2], [0, 5, 1, 4, 2], [0, 5, 3, 6, 2], [0, 6, 2], [0, 6, 3, 5, 1, 4, 2]]

Caminos desde el nodo 0 hasta el nodo 3:

[[0, 4, 1, 5, 3], [0, 4, 2, 6, 3], [0, 5, 1, 4, 2, 6, 3], [0, 5, 3], [0, 6, 2, 4, 1, 5, 3], [0, 6, 3]]

Flujo Óptimo

El flujo objetivo óptimo calculado (f*) (cFlow) es aproximadamente 1.9999999997668672.

Flujos por Receptor

Para cada receptor, el algoritmo determina los flujos a través de cada arista. A continuación, se detallan los resultados para cada receptor:

Receptor 1:

(6, 2): 0.33668

(4, 0): -0.0

(0, 4): 1.0

(2, 4): 0.0

(1, 5): 0.0

(5, 1): 1.0

(4, 2): 0.0

(0, 6): 0.67336

(1, 4): 0.0

(5, 0): -0.0

(2, 6): 0.0

(0, 5): 1.0

(6, 0): 0.0

(5, 3): 0.0

(3, 6): 0.0

(6, 3): 0.33668

(4, 1): 1.0

(3, 5): 0.0

Receptor 2:

(6, 2): 1.0

(4, 0): -0.0

(0, 4): 1.0

(2, 4): 0.0

(1, 5): 0.0

(5, 1): 0.33668

(4, 2): 1.0

(0, 6): 1.0

(1, 4): 0.0

(5, 0): 0.0

(2, 6): 0.0

(0, 5): 0.67336

(6, 0): -0.0

(5, 3): 0.33668

(3, 6): 0.0

(6, 3): 0.0

(4, 1): 0.0

(3, 5): 0.0

Receptor 3:

(6, 2): 0.0

(4, 0): 0.0

(0, 4): 0.67336

(2, 4): 0.0

(1, 5): 0.0

(5, 1): 0.0

(4, 2): 0.33668

(0, 6): 1.0

(1, 4): 0.0

(5, 0): -0.0

(2, 6): 0.0

(0, 5): 1.0

(6, 0): -0.0

(5, 3): 1.0

(3, 6): 0.0

(6, 3): 1.0

(4, 1): 0.33668

(3, 5): 0.0

Matrices de intermediación entre algoritmos:

<u>Receptor 1</u>:

[[1 1 0 0 0 0]

[0 0 1 1 0 0]

[0 0 0 0 1 1]

[0 0 0 0 0 0]

[0 0 0 1 1 0]

[0 1 0 0 0 1]

[0 0 0 0 0 0]

[1 0 0 1 1 0]

[0 1 0 0 0 0]

[0 0 0 0 0 0]

[0 1 1 0 0 1]

[0 0 0 1 0 0]

[0 0 0 1 1 0]

[0 1 0 0 0 0]

[0 1 0 0 0 1]

[0 0 0 1 0 0]

[0 0 0 0 0 0]

[0 0 0 0 0 0]]

```
Significant paths for receiver 1 with non-zero flow values:
[0, 4, 1]
[0, 5, 1]
Significant paths for receiver 2 with non-zero flow values:
[0, 4, 2]
[0, 6, 2]
Significant paths for receiver 3 with non-zero flow values:
[0, 5, 3]
[0, 6, 3]
```

Figura 29: paths significantes hasta destino

Flujo Acumulado por Nodo

Se muestra el flujo acumulado en cada nodo para cada receptor:

Receptor 1:

[0.0, 0.0, 0.0, 0.0, 0.33668, 0.0, 0.0]

Receptor 2:

[0.0, 0.0, 0.0, 0.0, 0.33668, 0.0, 0.0]

Receptor 3:

[0.0, 0.0, 0.0, 0.0, 0.33668, 0.0, 0.0]

Por último, con esta información calculada, se muestra el resultado del flujo encontrado:

```
Maximum sum of weights: 1.5000000000341152
```

Figura 30: throughput final del algoritmo

El proceso de combinar el algoritmo cFlow y los árboles de Steiner para analizar y optimizar redes complejas implica una serie de pasos fundamentales para obtener soluciones eficientes y robustas. A partir del resultado inicial del algoritmo cFlow, se determinan los pesos asignados a cada árbol, que representan la importancia relativa de los bordes en términos de conexiones y flujos dentro de la red. Estos pesos son muy importantes porque proporcionan una base sólida para la optimización posterior utilizando árboles de Steiner, que buscan el subgrafo más pequeño que conecta un conjunto determinado de nodos.

Una vez obtenida la solución cFlow, se crea una matriz de conmutación que refleja la disponibilidad de rutas entre pares de nodos. Estas matrices son importantes porque nos permiten determinar la viabilidad de rutas alternativas y qué aristas deben considerarse para optimizar la conectividad global de la red. Cada entrada en estas matrices está representada por un valor binario (1 indica que existe una posible ruta; 0 indica que no existe ninguna ruta). Esta representación matricial facilita el proceso de entrada de datos para el algoritmo del árbol de Steiner.

El algoritmo del árbol de Steiner utiliza esta información para construir una solución que minimice el costo total de las aristas seleccionadas. El algoritmo de árbol de Steiner procesa una matriz de conmutación y genera un conjunto de rutas optimizadas o rutas activas que conectan eficientemente los nodos. Este paso es muy importante porque los árboles de Steiner tienen como objetivo principal reducir la cantidad de bordes necesarios para conectar un subconjunto de nodos, aumentando así la eficiencia de la red y optimizando el rendimiento de los flujos de datos o recursos. Además de proporcionar una solución de conectividad optimizada, el algoritmo también calcula el flujo acumulado por nodo a lo largo de una ruta determinada. Este análisis de flujo detallado es crucial para comprender cómo se

distribuyen y utilizan los recursos dentro de las redes, especialmente para la planificación y gestión eficiente de comunicaciones de red.

Una vez obtenida la solución final del algoritmo del árbol de Steiner (1.5), se evalúa el rendimiento del flujo resultante. Esta evaluación compara el desempeño actual con los estándares de desempeño establecidos o esperados para la red. Se pueden tener en cuenta varios indicadores de desempeño, como la capacidad máxima de carga, la eficiencia en el uso de recursos y la minimización de la congestión. Esta fase de evaluación es crítica porque permite identificar áreas de mejora y realizar ajustes estratégicos si es necesario.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Los resultados obtenidos hasta ahora revelan un panorama complejo y matizado en el que los beneficios de la codificación de red varían ampliamente dependiendo de la topología y las características específicas de la red en cuestión. Este estudio muestra que, si bien la codificación de red no ofrece beneficios significativos sobre los métodos de enrutamiento tradicionales en ciertos casos, los beneficios pueden ser significativos o ilimitados en otros escenarios.

Para las redes de unidifusión de sesión única y las redes de transmisión de sesión única, encontramos que el rendimiento de la codificación de red es comparable al del enrutamiento tradicional. Este resultado sugiere que la implementación de técnicas de codificación de red en estas situaciones específicas puede no justificar la inversión en términos de complejidad y recursos adicionales. Sin embargo, estos resultados no descartan por completo los beneficios de la codificación de red en estos escenarios, ya que puede haber beneficios secundarios que no se consideraron en este estudio.

Por otro lado, las redes multidifusión omnidireccionales de sesión única tienen el doble del límite superior en la utilidad de codificación de red. Este resultado es particularmente interesante porque sugiere que, aunque la codificación de red puede mejorar el rendimiento de la red en estos casos, la mejora en el rendimiento de la red es limitada y cuantificable.
Esta información es valiosa para los diseñadores y administradores de redes porque les permite evaluar con mayor precisión si una implementación de codificación de red está justificada desde el punto de vista de la rentabilidad para redes de multidifusión no dirigidas.

Por el contrario, el escenario de red de multidifusión de sesión única dirigida presenta una imagen completamente diferente. En este caso, se ha demostrado que los beneficios de la codificación de red son potencialmente ilimitados. Este hallazgo es muy relevante ya que sugiere que la codificación de red puede generar ganancias de rendimiento excepcionales en

ciertas configuraciones de red de destino. Sin embargo, es importante señalar que estos beneficios ilimitados pueden plantear importantes desafíos de implementación y escalabilidad.

Quizás uno de los resultados más atractivos de este trabajo se refiere al caso de múltiples sesiones de unidifusión en redes no dirigidas que utilizan enrutamiento fraccionario. Hasta la fecha, no hemos identificado ningún escenario en el que la codificación de red proporcione más de un beneficio en estas condiciones. Este resultado es particularmente desconcertante, ya que es contradictorio que la codificación de red proporcione una ventaja en configuraciones de red más complejas. Esta limitación obvia de la codificación de red en esta situación particular plantea muchas preguntas y abre nuevas vías de investigación. Teniendo en cuenta estos resultados, queda claro que es necesario continuar y profundizar la investigación en esta área. Uno de los aspectos más prometedores de la investigación futura es la exploración de escenarios de red más complejos y realistas. Aunque la investigación actual se centra en configuraciones de red relativamente simples y bien definidas, las redes reales a menudo se caracterizan por topologías híbridas dinámicas y altamente complejas.

Las investigaciones futuras deberían abordar estos escenarios más complejos y determinar si la codificación de red puede proporcionar beneficios significativos en condiciones más cercanas a las redes operativas reales. Otra área importante para futuras investigaciones es la optimización de los algoritmos de codificación de redes. Aunque los resultados actuales indican limitaciones en ciertos escenarios, algoritmos más sofisticados y eficientes pueden superar estas limitaciones o descubrir beneficios que no son evidentes con los enfoques actuales. Este cuerpo de investigación explora nuevos algoritmos diseñados específicamente para abordar los desafíos identificados en redes no dirigidas con múltiples sesiones de unidifusión, donde los beneficios de la codificación de red han sido difíciles de lograr anteriormente. La validación empírica de resultados teóricos en condiciones reales de red es un área importante para futuras investigaciones.

Aunque los modelos teóricos proporcionan una base sólida para comprender los principios básicos, la implementación real de la codificación de redes puede presentar desafíos y oportunidades que no existen en los modelos abstractos.

Los estudios empíricos en redes operativas pueden revelar ventajas y limitaciones de la codificación de redes que no son evidentes en el análisis teórico, proporcionando así una comprensión más completa y matizada de su aplicabilidad práctica. La exploración de nuevos paradigmas que combinen la codificación de redes con otras tecnologías emergentes también es un área prometedora para futuras investigaciones.

Aunque los resultados actuales indican que existen limitaciones en determinadas situaciones, la codificación de red puede proporcionar beneficios excepcionales en determinadas condiciones o en determinadas configuraciones de red. Identificar y caracterizar estos casos extremos no solo puede ampliar la comprensión teórica de la codificación de redes, sino también revelar aplicaciones prácticas en áreas específicas donde sus beneficios son más obvios. El impacto de la seguridad y la resiliencia de la codificación de red en comparación con el enrutamiento tradicional es un área importante para futuras investigaciones. A medida que las redes se vuelven cada vez más importantes para la infraestructura global y manejan datos cada vez más confidenciales, es importante comprender cómo la codificación de la red afecta la seguridad y la resiliencia.

En resumidas palabras, aunque la investigación actual ha proporcionado una base sólida para comprender la codificación de redes y sus aplicaciones, todavía quedan muchas preguntas abiertas y áreas por investigar. La investigación futura en esta área tiene el potencial de transformar fundamentalmente nuestra comprensión de las redes de comunicación y descubrir nuevas formas de optimizar su rendimiento. A medida que avanzamos, es importante mantener un enfoque equilibrado que reconozca tanto las limitaciones como las oportunidades que presenta la codificación de redes y continúe encontrando soluciones innovadoras a los desafíos de las redes del siglo XXI.

Capítulo 8. BIBLIOGRAFÍA

[1] Zongpeng Li, Baochun Li. "Network Coding: The Case of Multiple Unicast Sessions".

[2] Dahai Xu, Mung Chang and Jennifer Rexford. "Link-State Routing with Hop-by-Hop Forwarding Can Achieve Optimal Traffic Engineering".

[3] Loeffler, B. "Cloud Computing: What is Infrastructure as a Service", Microsoft Technet Magazine, October 211. https://technet.microsoft.com/en-us/magazine/hh509051.aspx

[4] Vlassis, N.A.; Papakonstantinou, G.; Tsanakas, P. Dynamic sensory probabilistic maps for mobile robot localization.

[5] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, Senior Member, IEEE, and Raymond W. Yeung, Senior Member, IEEE "Network Information Flow".

[6] Zongpeng Li, Baochun Li, Dan Jiang, Lap Chi Lau. "On Achieving Optimal End-to-End Throughput in Data Networks: Theoretical and Empirical Studies".

[7] Satyajit Thakor and Mohammad Ishtiyaq Qureshi. "Undirected Unicast Network Capacity: A Partition Bound".

[8] Galois Fields: https://medium.com/@srupa.thota/introduction-to-galois-fields-8dd259d40ca1.

[9] Zongpeng Li, Baochun Li. "Network Coding in Undirected Networks". Department of Electrical and Computer Engineering. University of Toronto.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Este proyecto está principalmente en línea con varios Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas al contribuir indirectamente a la infraestructura y la eficiencia técnica. A continuación, se explica cómo se relaciona este proyecto con algunos ODS específicos.

ODS 9: Industria, Innovación e Infraestructura

Este proyecto contribuye directamente a este objetivo al tener como objetivo mejorar la eficiencia y el rendimiento de las redes de comunicación. La optimización de la red mediante técnicas avanzadas como la codificación de red conduce a una infraestructura de comunicaciones más sólida y eficiente, lo cual es esencial para el desarrollo económico y tecnológico sostenible.

ODS 11: Ciudades y Comunidades Sostenibles

Este proyecto puede contribuir al desarrollo de ciudades inteligentes y sostenibles mejorando la eficiencia de las redes de comunicación. Mejorar la eficiencia de la red puede mejorar la gestión de los recursos urbanos, aumentar la conectividad y reducir el consumo de energía asociado con la infraestructura de comunicaciones.

ODS 12: Producción y Consumo Responsables

La optimización de la red puede conducir a un uso más eficiente de los recursos técnicos, reduciendo potencialmente el consumo de energía y la necesidad de hardware adicional. Esto es consistente con el objetivo de promover patrones de producción y consumo más sostenibles en el sector tecnológico.

ODS 13: Acción por el Clima

Mejorar la eficiencia de la red puede ayudar indirectamente a reducir las emisiones de carbono asociadas con la infraestructura de comunicaciones. Las redes más eficientes requieren menos energía para funcionar, lo que puede tener un impacto positivo en la huella de carbono del sector tecnológico.

ODS 17: Alianzas para Lograr los Objetivos

La naturaleza colaborativa de la investigación en codificación y optimización de redes fomenta la colaboración entre instituciones académicas, empresas de tecnología y agencias gubernamentales. Estas colaboraciones son esenciales para el desarrollo y la implementación de soluciones tecnológicas sostenibles a escala global.

ANEXO II

cFlow LP:

```
import cvxpy as cp
import numpy as np
import networkx as nx


inputnodes = [0,6]
inputedges = [(0,5),(0,3),(3,4),(4,5),(4,6),(5,2),(3,1),(1,6),(6,2)]

non_member = [3,4,5,6]
source = 0
recievers = [1,2]


preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))


G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = np.arange(inputnodes[1]+1)
print(vertices)

edges = set(directE)
print(edges)


f_star = cp.Variable()  # Target flow rate


orientation_constraints = []
flow_constraints = []
equalrate_constraints = []

c =  {a : cp.Variable() for a in edges}
```

```
for a in edges:
    orientation_constraints += [c[a] >= 0]      #for every in 2E

for a1 in undirected_edges:         # for every in E
    a2 = (a1[1], a1[0])
    orientation_constraints += [ c[a1] + c[a2] == 1]


fia = {(i, a): cp.Variable() for a in edges for i in recievers}

for i in recievers:
     for a in edges:
          flow_constraints += [fia[i,a] <= c[a]]
          flow_constraints += [fia[i,a] >= 0]


for i in recievers:
    for j in non_member:
        incoming_edges = [(u, v) for u, v in edges if v == int(j)]
        outcoming_edges = [(u, v) for u, v in edges if u == int(j)]
        incoming_flow = sum(fia[i, x] for x in incoming_edges)
        outcoming_flow = sum(fia[i, y] for y in outcoming_edges)
        flow_constraints += [incoming_flow == outcoming_flow]
        print(incoming_edges)

    incoming_flow_source = sum(fia[i, x] for x in edges if x[1] == source)

    outcoming_flow_reciever = sum(fia[i, x] for x in edges if x[0] == int(i))
    incoming_flow_reciever = sum(fia[i, x] for x in edges if x[1] == int(i))

    flow_constraints += [incoming_flow_source == 0]
    flow_constraints += [outcoming_flow_reciever == 0]
    equalrate_constraints += [f_star == incoming_flow_reciever]


objective = cp.Maximize(f_star)
constraints = orientation_constraints + flow_constraints + equalrate_constraints
problem = cp.Problem(objective, constraints)


problem.solve()

print("Optimal Target Flow Rate (f*):", f_star.value)
```

```
import cvxpy as cp
import numpy as np
import networkx as nx



inputnodes = [0,6]
```

```
inputedges = [(0,5),(0,3),(3,4),(4,5),(4,6),(5,2),(3,1),(1,6),(6,2)]

C = {
    (0,5):1,
    (0,3):1,
    (3,4):1,
    (4,5):1,
    (4,6):1,
    (5,2):1,
    (3,1):1,
    (1,6):1,
    (6,2):1
}

non_member = [3,4,5,6]
source = 0
recievers = [1,2]


preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))


G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = np.arange(inputnodes[1]+1)
print(vertices)

edges = set(directE)
print(edges)


f_star = cp.Variable()  # Target flow rate


orientation_constraints = []
flow_constraints = []
equalrate_constraints = []

c =  {a : cp.Variable() for a in edges}

for a in edges:
    orientation_constraints += [c[a] >= 0]      #for every in 2E
```

```
for a1 in undirected_edges:          # for every in E
    a2 = (a1[1], a1[0])
    orientation_constraints += [ c[a1] + c[a2] == C[a1]]


fia = {(i, a): cp.Variable() for a in edges for i in recievers}

for i in recievers:
     for a in edges:
          flow_constraints += [fia[i,a] <= c[a]]
          flow_constraints += [fia[i,a] >= 0]


for i in recievers:
    for j in non_member:
        incoming_edges = [(u, v) for u, v in edges if v == int(j)]
        outcoming_edges = [(u, v) for u, v in edges if u == int(j)]
        incoming_flow = sum(fia[i, x] for x in incoming_edges)
        outcoming_flow = sum(fia[i, y] for y in outcoming_edges)
        flow_constraints += [incoming_flow == outcoming_flow]
        print(incoming_edges)

    incoming_flow_source = sum(fia[i, x] for x in edges if x[1] == source)

    outcoming_flow_reciever = sum(fia[i, x] for x in edges if x[0] == int(i))
    incoming_flow_reciever = sum(fia[i, x] for x in edges if x[1] == int(i))

    flow_constraints += [incoming_flow_source == 0]
    flow_constraints += [outcoming_flow_reciever == 0]
    equalrate_constraints += [f_star == incoming_flow_reciever]


objective = cp.Maximize(f_star)
constraints = orientation_constraints + flow_constraints + equalrate_constraints
problem = cp.Problem(objective, constraints)


problem.solve()

print("Optimal Target Flow Rate (f*):", f_star.value)

for i in recievers:
    for edge in edges:
        ivalue = i
        evalue = edge
        flow = fia[i,edge].value
        print("session: ", ivalue, "edge: ", evalue, ", flow: ",flow)
```

mFlow LP:

```
import cvxpy as cp
import numpy as np
import networkx as nx


inputnodes = [0,5]
inputedges = [(0,2),(4,5),(1,3),(0,4),(4,1),(2,5),(5,3)]

non_member = [4,5]
source = [0,1]
recievers = [[3],[2]]

sessions = [0,1]



preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))


G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = set(np.arange(inputnodes[1]+1))
# print(vertices)

edges = set(directE)
# print(edges)


f_star = cp.Variable()  # Target flow rate


orientation_constraints = []
flow_constraints = []
equalrate_constraints = []


c =  {a : cp.Variable() for a in edges}

for a in edges:
    orientation_constraints += [c[a] >= 0]      #for every in 2E

for a1 in undirected_edges:         # for every in E
    a2 = (a1[1], a1[0])
```

```
    orientation_constraints += [ c[a1] + c[a2] == 1]



fia = {(i, a): cp.Variable() for a in edges for i in sessions}
fija = {(i, j, a): cp.Variable() for i in sessions for j in recievers[i] for a in
edges}


for i in sessions:
    for j in recievers[i]:
        for a in edges:
            flow_constraints += [fija[i,j,a] >= 0]
            flow_constraints += [fija[i,j,a] <= fia[i,a]]

for a in edges:
    session_flow_edge = sum(fia[i,a] for i in sessions)
    flow_constraints += [session_flow_edge <= c[a]]



for i in sessions:
    for j in recievers[i]:
        non_member = vertices - {i} - {j}
        for vt in non_member:
            incoming_edges = [(u, v) for u, v in edges if v == int(vt)]
            outcoming_edges = [(u, v) for u, v in edges if u == int(vt)]
            incoming_flow = sum(fija[i, j, x] for x in incoming_edges)
            outcoming_flow = sum(fija[i, j, y] for y in outcoming_edges)
            flow_constraints += [incoming_flow == outcoming_flow]


        incoming_flow_source = sum(fija[i, j, x] for x in edges if x[1] ==
source[i])

        outcoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[0] ==
int(j))
        incoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[1] ==
int(j))

        flow_constraints += [incoming_flow_source == 0]
        flow_constraints += [outcoming_flow_reciever == 0]
        equalrate_constraints += [f_star == incoming_flow_reciever]



objective = cp.Maximize(f_star)
constraints = orientation_constraints + flow_constraints + equalrate_constraints
problem = cp.Problem(objective, constraints)


problem.solve()
```

```
print("Optimal Target Flow Rate (f*):", f_star.value)
```

cFlow Dual LP:

```
import cvxpy as cp
import numpy as np
import networkx as nx


inputnodes = [0,6]
inputedges = [(0,5),(0,3),(3,4),(4,5),(4,6),(5,2),(3,1),(1,6),(6,2)]

C = {
    (0,5):1,
    (0,3):1,
    (3,4):1,
    (4,5):1,
    (4,6):1,
    (5,2):1,
    (3,1):1,
    (1,6):1,
    (6,2):1
}

non_member = [3,4,5,6]
source = 0
recievers = [1,2]


preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))


G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = np.arange(inputnodes[1]+1)
print(vertices)

edges = set(directE)
```

```
print(edges)


# start building program
fstar_constraints = []
flow_constraints = []
capacity_constraints = []
variable_constraints = []


phi = {(i): cp.Variable() for i in recievers} # f* == incoming reciever flow

fstar_constraints = [(-1 + sum(phi[i] for i in recievers )) == 0]

mu = {(i, a): cp.Variable() for a in edges for i in recievers}   # fia <= c(a)
xi = {(i, a): cp.Variable() for a in edges for i in recievers}   # fia <= c(a)

for i in recievers:
    for a in edges:
        variable_constraints += [xi[i,a] >= 0]
        variable_constraints += [mu[i,a] >= 0]

rho = {(i, v): cp.Variable() for v in non_member for i in recievers}   # outfrom
non-member == into non-memebr
sigma = {(i): cp.Variable() for i in recievers}   # into source == 0
tau = {(i): cp.Variable() for i in recievers}   # outfrom reciever == 0




for i in recievers:
    for a in edges:
        equation = - mu[i,a] + xi[i,a]
        if a[1] in non_member:
            equation += rho[i,a[1]]
        if a[0] in non_member:
            equation -= rho[i,a[0]]
        if a[1] == int(source):
            equation += sigma[i]
        if a[0] == int(i):
            equation += tau[i]
        if a[1] == int(i):
            equation -= phi[i]
        flow_constraints += [equation == 0]


lambd = {a: cp.Variable() for a in edges}

nu = {a1: cp.Variable() for a1 in undirected_edges}   # fia <= c(a)


for a1 in undirected_edges:
    a2 = (a1[1], a1[0])
```

```
    equation1 = nu[a1]
    equation2 = nu[a1]
    for i in recievers:
        equation1 -= (xi[i, a1] + lambd[a1])
        equation2 -= (xi[i, a2] + lambd[a2])

    variable_constraints += [lambd[a1] >= 0]
    variable_constraints += [lambd[a2] >= 0]

    capacity_constraints += [equation1 == 0]
    capacity_constraints += [equation2 == 0]


objective = cp.Minimize(sum(nu[a1]*C[a1] for a1 in undirected_edges))

constraints = fstar_constraints + flow_constraints + capacity_constraints +
variable_constraints

problem = cp.Problem(objective, constraints)
problem.solve()

print("minimal objective:", sum(nu[a1]*C[a1] for a1 in undirected_edges).value)


for i in recievers:
    for edge in edges:
        ivalue = i
        evalue = edge
        xivalue = xi[i,edge].value
        print("session: ", ivalue, "edge: ", evalue, ", flow: ",xivalue)

sum_xi = {}

for edge in edges:
    sum_xi_edge = 0
    for i in recievers:
        sum_xi_edge += xi[i,edge].value
    sum_xi[edge] = sum_xi_edge
    print("edge:", edge, ", sumxivalue: ", sum_xi_edge)
```

Testing:

```
# graph: undirected multicast figure-4
# routing only

import cvxpy as cp

variables = cp.Variable(9)
```

```
fstar = cp.Variable()

def x(index):
    return variables[index - 1]

constraints = [
    x(3) + x(6) + x(8) <= 1,
    x(1) + x(5) + x(8) <= 1,
    x(2) + x(5) + x(9) <= 1,
    x(4) + x(6) + x(9) <= 1,
    x(1) + x(3) + x(7) <= 1,
    x(2) + x(4) + x(7) <= 1,
    x(1) + x(2) == x(3) + x(4),
    x(5) + x(6) + x(7) == x(1) + x(2),
    x(8) + x(9) == x(1) + x(2),
    fstar == x(1) + x(2)
]

objective = cp.Maximize(fstar)
problem = cp.Problem(objective, constraints)
problem.solve()
print("Optimal Target Flow Rate (f*):", fstar.value)
for i in range(1, 10):
    print(f"x{i}:", x(i).value)
```

```
# optimal solution

import cvxpy as cp
import numpy as np
import networkx as nx


inputnodes = [0,4]
inputedges = [(0,2),(2,3),(3,4),(4,0),(2,1),(1,4)]

non_member = []
source = [0,1,2,3]
recievers = [[1],[3],[4],[0]]

sessions = [0,1,2,3]


preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))
```

```
G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = set(np.arange(inputnodes[1]+1))
print(vertices)

edges = set(directE)
print(edges)


f_star = cp.Variable()  # Target flow rate


orientation_constraints = []
flow_constraints = []
equalrate_constraints = []


c =  {a : cp.Variable() for a in edges}

for a in edges:
    orientation_constraints += [c[a] >= 0]      #for every in 2E

for a1 in undirected_edges:         # for every in E
    a2 = (a1[1], a1[0])
    orientation_constraints += [ c[a1] + c[a2] == 1]



fia = {(i, a): cp.Variable() for a in edges for i in sessions}
fija = {(i, j, a): cp.Variable() for i in sessions for j in recievers[i] for a in
edges}


for i in sessions:
    for j in recievers[i]:
        for a in edges:
            flow_constraints += [fija[i,j,a] >= 0]
            flow_constraints += [fija[i,j,a] <= fia[i,a]]

for a in edges:
    session_flow_edge = sum(fia[i,a] for i in sessions)
    flow_constraints += [session_flow_edge <= c[a]]



for i in sessions:
    for j in recievers[i]:
```

```
        non_member = vertices - {i} - {j}
        for vt in non_member:
            incoming_edges = [(u, v) for u, v in edges if v == int(vt)]
            outcoming_edges = [(u, v) for u, v in edges if u == int(vt)]
            incoming_flow = sum(fija[i, j, x] for x in incoming_edges)
            outcoming_flow = sum(fija[i, j, y] for y in outcoming_edges)
            flow_constraints += [incoming_flow == outcoming_flow]


        incoming_flow_source = sum(fija[i, j, x] for x in edges if x[1] ==
source[i])

        outcoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[0] ==
int(j))
        incoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[1] ==
int(j))

        flow_constraints += [incoming_flow_source == 0]
        flow_constraints += [outcoming_flow_reciever == 0]
        equalrate_constraints += [f_star == incoming_flow_reciever]



objective = cp.Maximize(f_star)
constraints = orientation_constraints + flow_constraints + equalrate_constraints
problem = cp.Problem(objective, constraints)


problem.solve()

print("Optimal Target Flow Rate (f*):", f_star.value)
print(fia[0,(3,4)].value)
```

```
# graph: multiple unicast 2004 figure-3


import cvxpy as cp
import numpy as np
import networkx as nx


inputnodes = [0,5]
inputedges = [(0,2),(4,5),(1,3),(0,4),(4,1),(2,5),(5,3)]

non_member = [4,5]
source = [0,1]
recievers = [[3],[2]]

sessions = [0,1]
```

```
preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))


G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = set(np.arange(inputnodes[1]+1))
# print(vertices)

edges = set(directE)
# print(edges)


f_star = cp.Variable()  # Target flow rate


orientation_constraints = []
flow_constraints = []
equalrate_constraints = []


c =  {a : cp.Variable() for a in edges}

for a in edges:
    orientation_constraints += [c[a] >= 0]      #for every in 2E

for a1 in undirected_edges:         # for every in E
    a2 = (a1[1], a1[0])
    orientation_constraints += [ c[a1] + c[a2] == 1]



fia = {(i, a): cp.Variable() for a in edges for i in sessions}
fija = {(i, j, a): cp.Variable() for i in sessions for j in recievers[i] for a in
edges}


for i in sessions:
    for j in recievers[i]:
        for a in edges:
            flow_constraints += [fija[i,j,a] >= 0]
```

```
            flow_constraints += [fija[i,j,a] <= fia[i,a]]

for a in edges:
    session_flow_edge = sum(fia[i,a] for i in sessions)
    flow_constraints += [session_flow_edge <= c[a]]



for i in sessions:
    for j in recievers[i]:
        non_member = vertices - {i} - {j}
        for vt in non_member:
            incoming_edges = [(u, v) for u, v in edges if v == int(vt)]
            outcoming_edges = [(u, v) for u, v in edges if u == int(vt)]
            incoming_flow = sum(fija[i, j, x] for x in incoming_edges)
            outcoming_flow = sum(fija[i, j, y] for y in outcoming_edges)
            flow_constraints += [incoming_flow == outcoming_flow]


        incoming_flow_source = sum(fija[i, j, x] for x in edges if x[1] ==
source[i])

        outcoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[0] ==
int(j))
        incoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[1] ==
int(j))

        flow_constraints += [incoming_flow_source == 0]
        flow_constraints += [outcoming_flow_reciever == 0]
        equalrate_constraints += [f_star == incoming_flow_reciever]




objective = cp.Maximize(f_star)
constraints = orientation_constraints + flow_constraints + equalrate_constraints
problem = cp.Problem(objective, constraints)


problem.solve()

print("Optimal Target Flow Rate (f*):", f_star.value)
```

```
# graph: example 3 in my notes
# routing only

import cvxpy as cp

variables = cp.Variable(10)
fstar = cp.Variable()
```

```
def x(index):
    return variables[index - 1]

# 8 edges
constraints = [
    x(2) + x(4) + x(9) <= 1,
    x(1) + x(4) + x(8) <= 1,
    x(3) + x(8) + x(9) <= 1,
    x(2) + x(5) + x(10) <= 1,
    x(1) + x(5) + x(7) <= 1,
    x(3) + x(7) + x(10) <= 1,
    x(6) + x(9) + x(10) <= 1,
    x(6) + x(7) + x(8) <= 1,

    x(1) + x(2) + x(3) == x(4) + x(5) + x(6),
    x(1) + x(2) + x(3) == x(7) + x(8) + x(9) + x(10),
    fstar == x(1) + x(2) + x(3)
]

objective = cp.Maximize(fstar)
problem = cp.Problem(objective, constraints)
problem.solve()
print("Optimal Target Flow Rate (f*):", fstar.value)
for i in range(1, 11):
    print(f"x{i}:", x(i).value)
```

```
# graph: multiple unicast 2004 figure-3


import cvxpy as cp
import numpy as np
import networkx as nx


inputnodes = [0,5]
inputedges = [(0,1),(0,4),(0,2),(3,1),(3,4),(3,2),(5,1),(5,4)]

non_member = []
source = [0,1,2]
recievers = [[3],[4],[5]]

sessions = [0,1,2]



preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
```

```
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))


G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = set(np.arange(inputnodes[1]+1))
# print(vertices)

edges = set(directE)
# print(edges)


f_star = cp.Variable()  # Target flow rate


orientation_constraints = []
flow_constraints = []
equalrate_constraints = []


c =  {a : cp.Variable() for a in edges}

for a in edges:
    orientation_constraints += [c[a] >= 0]      #for every in 2E

for a1 in undirected_edges:         # for every in E
    a2 = (a1[1], a1[0])
    orientation_constraints += [ c[a1] + c[a2] == 1]



fia = {(i, a): cp.Variable() for a in edges for i in sessions}
fija = {(i, j, a): cp.Variable() for i in sessions for j in recievers[i] for a in
edges}


for i in sessions:
    for j in recievers[i]:
        for a in edges:
            flow_constraints += [fija[i,j,a] >= 0]
            flow_constraints += [fija[i,j,a] <= fia[i,a]]

for a in edges:
    session_flow_edge = sum(fia[i,a] for i in sessions)
    flow_constraints += [session_flow_edge <= c[a]]
```

```
for i in sessions:
    for j in recievers[i]:
        non_member = vertices - {i} - {j}
        for vt in non_member:
            incoming_edges = [(u, v) for u, v in edges if v == int(vt)]
            outcoming_edges = [(u, v) for u, v in edges if u == int(vt)]
            incoming_flow = sum(fija[i, j, x] for x in incoming_edges)
            outcoming_flow = sum(fija[i, j, y] for y in outcoming_edges)
            flow_constraints += [incoming_flow == outcoming_flow]


        incoming_flow_source = sum(fija[i, j, x] for x in edges if x[1] ==
source[i])

        outcoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[0] ==
int(j))
        incoming_flow_reciever = sum(fija[i, j, x] for x in edges if x[1] ==
int(j))

        flow_constraints += [incoming_flow_source == 0]
        flow_constraints += [outcoming_flow_reciever == 0]
        equalrate_constraints += [f_star == incoming_flow_reciever]




objective = cp.Maximize(f_star)
constraints = orientation_constraints + flow_constraints + equalrate_constraints
problem = cp.Problem(objective, constraints)


problem.solve()

print("Optimal Target Flow Rate (f*):", f_star.value)

import cvxpy as cp
import networkx as nx

# Example graph setup
inputnodes = [0, 5]
inputedges = [(0, 1), (0, 4), (0, 2), (3, 1), (3, 4), (3, 2), (5, 1), (5, 4)]
G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(inputedges)

# Sessions: source node and destination nodes
sessions = {
    0: {'source': 0, 'destinations': [3, 4]},
    1: {'source': 5, 'destinations': [1, 2]}
}

# Flow variables for each edge in each session
```

```
f = {(s, e): cp.Variable(nonneg=True) for s in sessions for e in G.edges}

# Objective: Maximize the minimum flow across all sessions
# Objective: Maximize the minimum flow across all sessions
objective = cp.Maximize(cp.minimum([cp.sum([f[s, e] for e in
G.out_edges(sessions[s]['source'])]) for s in sessions]))

# Constraints
constraints = []

# Flow conservation for each session
for s in sessions:
    for node in G.nodes:
        if node == sessions[s]['source']:
            # Total outgoing flow from source
            constraints.append(cp.sum(f[s, e] for e in G.out_edges(node)) ==
cp.sum(f[s, e] for e in G.in_edges(node)))
        elif node in sessions[s]['destinations']:
            # Flow into destinations
            constraints.append(cp.sum(f[s, e] for e in G.in_edges(node)) ==
cp.sum(f[s, e] for e in G.out_edges(node)))
        else:
            # Flow conservation at intermediate nodes
            constraints.append(cp.sum(f[s, e] for e in G.in_edges(node)) ==
cp.sum(f[s, e] for e in G.out_edges(node)))

# Capacity constraints (assuming each edge has a capacity of 1)
for e in G.edges:
    constraints.append(cp.sum(f[s, e] for s in sessions) <= 1)

# Problem
problem = cp.Problem(objective, constraints)

# Solve
problem.solve()

# Output
print("Optimal Routing Throughput:", problem.value)
for s in sessions:
    for e in G.edges:
        print(f"Session {s}, Flow on edge {e}: {f[s, e].value}")
```

Algoritmo Final:

```
import cvxpy as cp
import numpy as np
import networkx as nx
```

```
#----------------- Butterfly flow=1.5/1.875 --------------------------------
# inputnodes = [0,6]
# inputedges = [(0,5),(0,3),(3,4),(4,5),(4,6),(5,2),(3,1),(1,6),(6,2)]

# C = {
#      (0,5):1,
#      (0,3):1,
#      (3,4):1,
#      (4,5):1,
#      (4,6):1,
#      (5,2):1,
#      (3,1):1,
#      (1,6):1,
#      (6,2):1
# }

# non_member = [3,4,5,6]
# source = 0
# receivers = [1,2]

#----------------- C(3,2), flow=1.5/1.8--------------------------------

inputnodes = [0,6]
inputedges = [(0,4),(0,5),(0,6),(4,1),(4,2),(5,1),(5,3),(6,2),(6,3)]

C = {
     (0,4):1,
     (0,5):1,
     (0,6):1,
     (4,1):1,
     (4,2):1,
     (5,1):1,
     (5,3):1,
     (6,2):1,
     (6,3):1,
}

non_member = [4,5,6]
source = 0
receivers = [1,2,3]



#----------------- Fig7. in mflow paper, flow=13.5/13.5 -------------------------
-------

# inputnodes = [0,7]
# inputedges = [(1, 3), (1, 4), (3, 4), (3, 5), (4, 5), (2, 3), (2, 5), (0, 5),
(0, 7), (2, 7), (6, 7), (2, 6), (1, 6), (0, 4), (4, 7), (4, 6)]

# C = {(1,3):4, (1,4):8, (3,4):1, (3,5):1, (4,5):1, (2,3):4, (2,5):4, (0,5):4,
(0,7):4, (2,7):4, (6,7):1, (2,6):4, (1,6):4, (0,4):8, (4,7):1, (4,6):1}
```

```
# non_member = [3,4,5,6,7]
# source=0
# receivers = [1,2]


#----------------- C(4,3), flow=2/2.667 --------------------------------

# inputnodes = [0,7]
# inputedges = [(0,5),    (0,6),    (0,7),    (0,8),    (5,1),    (5,2),    (5,3),
(6,1),    (6,2),    (6,4),    (7,1),    (7,3),    (7,4),    (8,2),    (8,3),
(8,4)]

# C = {
#     (0,5):1,
#     (0,6):1,
#     (0,7):1,
#     (0,8):1,
#     (5,1):1,
#     (5,2):1,
#     (5,3):1,
#     (6,1):1,
#     (6,2):1,
#     (6,4):1,
#     (7,1):1,
#     (7,3):1,
#     (7,4):1,
#     (8,2):1,
#     (8,3):1,
#     (8,4):1
# }

# non_member = [5,6,7,8]
# source = 0
# receivers = [1,2,3,4]

#----------------- C(4,2), flow=1.333/1.778 --------------------------------
# inputnodes = [0,10]
# inputedges =
[(0,7),(0,8),(0,9),(0,10),(7,1),(7,2),(7,3),(8,1),(8,5),(8,4),(9,2),(9,4),(9,6),(
10,3),(10,5),(10,6)]
# C = {(0, 7): 1, (0, 8): 1, (0, 9): 1, (0, 10): 1, (7, 1): 1, (7, 2): 1, (7, 3):
1, (8, 1): 1, (8, 5): 1, (8, 4): 1, (9, 2): 1, (9, 4): 1, (9, 6): 1, (10, 3): 1,
(10, 5): 1, (10, 6): 1}

# non_member = [7,8,9,10]
# source=0
# receivers = [1,2,3,4,5,6]


#----------------- C(5,4), flow=2.5/3.571 --------------------------------
# inputnodes = [0,10]
```

```
# inputedges =
[(0,6),(0,7),(0,8),(0,9),(0,10),(6,1),(6,2),(6,3),(6,4),(7,1),(7,2),(7,3),(7,5),(
8,1),(8,2),(8,4),(8,5),(9,1),(9,3),(9,4),(9,5),(10,2),(10,3),(10,4),(10,5)]
# C = {(0, 6): 1, (0, 7): 1, (0, 8): 1, (0, 9): 1, (0, 10): 1, (6, 1): 1, (6, 2):
1, (6, 3): 1, (6, 4): 1, (7, 1): 1, (7, 2): 1, (7, 3): 1, (7, 5): 1, (8, 1): 1,
(8, 2): 1, (8, 4): 1, (8, 5): 1, (9, 1): 1, (9, 3): 1, (9, 4): 1, (9, 5): 1, (10,
2): 1, (10, 3): 1, (10, 4): 1, (10, 5): 1}
# non_member = [6,7,8,9,10]
# source=0
# receivers = [1,2,3,4,5]

#----------------- C(5,2), flow=1.25/1.786 --------------------------------

# inputnodes = [0, 10]
# inputedges = [(0,1), (0,2), (0,3), (0,4), (0,5), (1,6), (2,6), (1,7), (3,7),
(1,8), (4,8), (1,9), (5,9), (2,10), (3,10), (2,11), (4,11), (2,12), (5,12),
(3,13), (4,13), (3,14), (5,14), (4,15), (5,15)]
# C = {(0, 1): 1, (0, 2): 1, (0, 3): 1, (0, 4): 1, (0, 5): 1, (1, 6): 1, (2, 6):
1, (1, 7): 1, (3, 7): 1, (1, 8): 1, (4, 8): 1, (1, 9): 1, (5, 9): 1, (2, 10): 1,
(3, 10): 1, (2, 11): 1, (4, 11): 1, (2, 12): 1, (5, 12): 1, (3, 13): 1, (4, 13):
1, (3, 14): 1, (5, 14): 1, (4, 15): 1, (5, 15): 1}
# non_member = [1, 2, 3, 4, 5]
# source = 0
# receivers = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15]




#----------------- C(5,3), flow=?/? --------------------------------
# inputnodes = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
# inputedges = [(0,11), (0,12), (0,13), (0,14), (0,15),
#                 (11,1), (12,1), (13,1),
#                 (11,2), (12,2), (14,2),
#                 (11,3), (13,3), (15,3),
#                 (11,4), (14,4), (15,4),
#                 (12,5), (13,5), (14,5),
#                 (12,6), (13,6), (15,6),
#                 (12,7), (14,7), (15,7),
#                 (13,8), (14,8), (15,8),
#                 (11,9), (12,9), (15,9),
#                 (13,10), (14,10), (15,10)]
# C = {(0, 11): 1, (0, 12): 1, (0, 13): 1, (0, 14): 1, (0, 15): 1,
#        (11, 1): 1, (12, 1): 1, (13, 1): 1,
#        (11, 2): 1, (12, 2): 1, (14, 2): 1,
#        (11, 3): 1, (13, 3): 1, (15, 3): 1,
#        (11, 4): 1, (14, 4): 1, (15, 4): 1,
#        (12, 5): 1, (13, 5): 1, (14, 5): 1,
#        (12, 6): 1, (13, 6): 1, (15, 6): 1,
#        (12, 7): 1, (14, 7): 1, (15, 7): 1,
#        (13, 8): 1, (14, 8): 1, (15, 8): 1,
#        (11, 9): 1, (12, 9): 1, (15, 9): 1,
#        (13, 10): 1, (14, 10): 1, (15, 10): 1}
# non_member = [11, 12, 13, 14, 15]
```

```
# source = 0
# receivers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]



#----------------- coding started  --------------------------------
preG = nx.Graph()
preG.add_nodes_from(inputnodes)
preG.add_edges_from(inputedges)
directE = []
for n, nbrs in preG.adj.items():
    for nbr, eattr in nbrs.items():
        directE.append((n,nbr))


G = nx.DiGraph()
G.add_nodes_from(inputnodes)
G.add_edges_from(directE)

undirected_edges = set(inputedges)

vertices = np.arange(inputnodes[1]+1)
print(vertices)

edges = set(directE)
print(edges)



def find_all_paths(G, source, receivers):
    all_paths = {}
    for receiver in receivers:
        # problem: all_simple_paths is NP hard!
        paths = list(nx.all_simple_paths(G, source=source, target=receiver))
        all_paths[receiver] = paths
        print(f"All paths from {source} to {receiver}: {paths}")
    return all_paths


all_paths = find_all_paths(G, source, receivers)
print(all_paths)




#----------------------------- Cflow LP --------------------------#


def cflow(source, receivers, non_member, undirected_edges, edges, C):
    f_star = cp.Variable()  # Target flow rate


    orientation_constraints = []
```

```
    flow_constraints = []
    equalrate_constraints = []

    c =  {a : cp.Variable() for a in edges}

    for a in edges:
        orientation_constraints += [c[a] >= 0]      #for every in 2E

    for a1 in undirected_edges:         # for every in E
        a2 = (a1[1], a1[0])
        orientation_constraints += [ c[a1] + c[a2] == C[a1]]


    fia = {(i, a): cp.Variable() for a in edges for i in receivers}

    for i in receivers:
        for a in edges:
            flow_constraints += [fia[i,a] <= c[a]]
            flow_constraints += [fia[i,a] >= 0]


    for i in receivers:
        for j in non_member:
            incoming_edges = [(u, v) for u, v in edges if v == int(j)]
            outcoming_edges = [(u, v) for u, v in edges if u == int(j)]
            incoming_flow = sum(fia[i, x] for x in incoming_edges)
            outcoming_flow = sum(fia[i, y] for y in outcoming_edges)
            flow_constraints += [incoming_flow == outcoming_flow]

        incoming_flow_source = sum(fia[i, x] for x in edges if x[1] == source)

        outcoming_flow_reciever = sum(fia[i, x] for x in edges if x[0] == int(i))
        incoming_flow_reciever = sum(fia[i, x] for x in edges if x[1] == int(i))

        flow_constraints += [incoming_flow_source == 0]
        flow_constraints += [outcoming_flow_reciever == 0]
        equalrate_constraints += [f_star == incoming_flow_reciever]


    objective = cp.Maximize(f_star)
    constraints = orientation_constraints + flow_constraints +
equalrate_constraints
    problem = cp.Problem(objective, constraints)


    problem.solve()

    print("Optimal Target Flow Rate (f*):", f_star.value)

    optimal_flow = f_star.value
    flowia = {(i, a): round(float(fia[i,a].value), 6) for a in edges for i in
receivers}
```

```
    for i in receivers:
        for edge in edges:
            ivalue = i
            evalue = edge
            flow = round(float(fia[ivalue,evalue].value), 6)
            print("Receiver:", ivalue, ", edge:", evalue, ", flow: ",flow)

    return(optimal_flow, flowia)


optimal_flow, flowia = cflow(source, receivers, non_member, undirected_edges,
edges, C)
print(flowia)

#-----------------------------  Splitting cflow Function with LP ----------------
-----------#

# single matrix for one receiver
def create_edge_path_matrix(G, all_paths, receiver):
    edges = list(G.edges())
    paths = all_paths[receiver]

    matrix = np.zeros((len(edges), len(paths)), dtype=int)
    edge_index = {edge: idx for idx, edge in enumerate(edges)}

    for path_idx, path in enumerate(paths):
        for u, v in zip(path[:-1], path[1:]):  # Create edges from path
            if (u, v) in edge_index:
                matrix[edge_index[(u, v)], path_idx] = 1

    return matrix

# save matrices for all receivers
matrices = {}
for receiver in receivers:
    matrices[receiver] = create_edge_path_matrix(G, all_paths, receiver)
    print(f"Matrix for receiver {receiver}:\n{matrices[receiver]}\n")



def optimize_path_flows(matrix, edge_flows):
    num_paths = matrix.shape[1]          # column number
    print("number of paths is", num_paths)
    num_edges = matrix.shape[0]          # row number
    x = cp.Variable(num_paths, nonneg=True)

    objective = cp.Maximize(cp.sum(x))
    constraints = [matrix @ x <= edge_flows]

    problem = cp.Problem(objective, constraints)
    result = problem.solve()
    return x.value, result
```

```
# Applying the optimization to each receiver
optimized_flows = {}
for receiver, matrix in matrices.items():
    # Get the flows for each edge for this receiver from flowia
    edge_flows = np.array([flowia.get((receiver, edge), 0) for edge in
G.edges()])

    # Run the optimization
    optimized_path_flow, max_flow_sum = optimize_path_flows(matrix, edge_flows)
    optimized_flows[receiver] = (optimized_path_flow, max_flow_sum)
    print(f"Optimized path flows for receiver {receiver}: {optimized_path_flow}")
    print(f"Maximum sum of flows for receiver {receiver}: {max_flow_sum}\n")


significant_paths = {}

threshold = 1e-6

for receiver, (optimized_path_flow, _) in optimized_flows.items():
    # Fetch the paths from all_paths for this receiver
    paths = all_paths[receiver]

    # Filter paths based on the optimized path flow values
    non_zero_paths = [paths[i] for i in range(len(paths)) if
optimized_path_flow[i] > threshold]

    # Save or print the non-zero paths
    significant_paths[receiver] = non_zero_paths
    print(f"Significant paths for receiver {receiver} with non-zero flow
values:")
    for path in non_zero_paths:
        print(path)

print(significant_paths)


#----------------------------  combine to steiner tree -------------------------
--#

from itertools import product

# Generate all possible combinations of paths, one from each receiver's
significant paths
path_combinations = list(product(*[significant_paths[receiver] for receiver in
significant_paths]))

resulting_graphs = []

# Create a graph for each combination of paths
for combination in path_combinations:
    # Create a new directed graph for this combination
    combo_graph = nx.DiGraph()
```

```
    # Add paths to the graph
    for path in combination:
        nx.add_path(combo_graph, path)

    # Add this graph to the list of resulting graphs
    resulting_graphs.append(combo_graph)

print(f"Total combinations (resulting graphs) created: {len(resulting_graphs)}")
edge_lists = []

for graph in resulting_graphs:
    edge_list = list(graph.edges())
    edge_lists.append(edge_list)

print("resulting steiner trees are ", edge_lists)

# missed: cycle detecting and remove #

#-----------------------------  packing steiner trees LP ------------------------
---#


n_graphs = len(edge_lists)              # Number of resulting graphs
packing_x = cp.Variable(n_graphs, nonneg=True)
packing_objective = cp.Maximize(cp.sum(packing_x))
packing_constraints = []

for (u, v), capacity in C.items():
    total_flow = cp.sum([packing_x[i] for i, edges in enumerate(edge_lists) if
(u, v) in edges or (v, u) in edges])

    packing_constraints.append(total_flow <= capacity)

packing_problem = cp.Problem(packing_objective, packing_constraints)
packing_result = packing_problem.solve()
optimized_weights = packing_x.value

print("Optimized weights for each graph:", optimized_weights)
print("Maximum sum of weights:", packing_result)
```