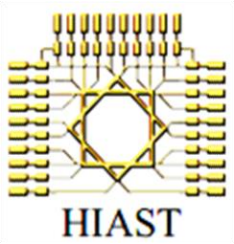
الجمهورية العربية السورية

المعهد العالي للعلوم التطبيقية والتكنولوجيا

قسم الاتصالات



موازنة الحمل في الشبكات المعرفة برمجيا

Load Balancing in Software–Defined Networks (SDN)

إعداد

رسول رفعت بشر

إشراف

ما. محمد بشار دسوقي

2024

أهدي هذا العمل ...

إلى من وددناه ان يكون معنا ولكن أبى إلا أن يرحل وقلوبنا معه الى جوار ربه...

المرحوم خضر محمد خليل

إلى أعز الناس، اللذين كانا ولا يزالان سندي ودعمي الأكبر في كل خطوة أخطوها.

والديّ العزيزين

إلى الذين كانوا دائماً بجانبي، يشاركونني أفراحي وأحلامي.

إخوتي الأحباء

إلى الذين كانوا عوناً لي في كل خطوة على هذا الطريق.

أصدقائي الأعزاء

كلمة شكر...

أود أن أعبر عن عميق شكري وامتناني لأستاذي الكريم المهندس **محمد بشار دسوقي**، على توجيهاته السديدة ودعمه المتواصل طوال فترة إعداد هذا المشروع.

خلاصة

يهدف هذا المشروع إلى تحسين أداء الشبكات المعرفة برمجياً (SDN) من خلال دراسة وتطبيق تقنيات موازنة الحمل. تمثل SDN تحولاً جذرياً في تصميم الشبكات التقليدية عبر فصل طبقة التحكم عن طبقة التوجيه، مما يمنح مرونة أكبر في إدارة الشبكة. يركز المشروع على تطوير خوارزمية موازنة حمل مخصصة لبيئة SDN، بحيث تساهم في توزيع الأحمال بشكل فعّال وتقليل زمن التأخير وتحسين استخدام الموارد. بعد تصميم الخوارزمية، سيتم تنفيذها في بيئة محاكاة باستخدام Mininet واختبارها من خلال سيناريوهات مختلفة. سيتم تقييم فعالية الخوارزمية عبر تحليل البيانات المتعلقة بأداء الشبكة حيث تم تناول جانبي لموازنة الحمل وذلك بتطبيق موازنة الحمل على المخدمات وعلى اختيار الوصلات ولاحظنا تحسن أداء الشبكة بالنسبة لبعض المقاييس الموضوعة ك معدل النقل وزمن التأخير، مما يوفر أساساً لتوصيات مستقبلية لتحسين الخوارزمية وتوسيع نطاق تطبيقها.

Abstract

This project aims to enhance the performance of Software-Defined Networks (SDN) by studying and applying load balancing techniques. SDN represents a radical shift in traditional network design by separating the control plane from the data plane, providing greater flexibility in network management. The project focuses on developing a custom load-balancing algorithm specifically for the SDN environment, contributing to effective load distribution, reduced latency, and improved resource utilization. After designing the algorithm, it will be implemented in a simulation environment using **Mininet** and tested through various scenarios. The effectiveness of the algorithm will be evaluated by analyzing network performance data, addressing both aspects of load balancing: applying load balancing on servers and link selection. We observed improved network performance metrics, such as throughput and latency, providing a foundation for future recommendations to enhance the algorithm and expand its applicability.

فهرس المحتويات

قائمة الأشكال

قائمة بأهم الاختصارات

الاختصار	الشرح
SDN	Software-Defined Networks
IP	Internet Protocol
SNMP	Simple Network Management Protocol
API	Application Programming Interface
NOS	Network Operating System
IDS	Intrusion Detection System
OSPF	Open Shortest Path First
BGP	Border Gateway Protocol
SPF	Single Point of Failure
QoS	Quality of Service
HTTP	Hypertext Transfer Protocol
BFS	Breadth-First Search
DFS	Depth-First Search
RTT	Round Trip Time
DNS	Domain Name Server
NAT	Network Address Translation
DHCP	Dynamic Host Configuration Protocol
VM	Virtual Machine
STP	Spanning Tree Protocol
IoT	Internet of Things

مقدمة

تحتاج المواقع الإلكترونية ذات الحركة المرورية العالية في الوقت الحالي إلى التعامل مع مئات الآلاف، أو حتى الملايين، من الطلبات المتزامنة من المستخدمين أو العملاء، وتقديم النصوص أو الصور أو الفيديوهات أو بيانات التطبيقات بشكل صحيح وبسرعة وموثوقية. وللتوسع بكفاءة من حيث التكلفة لمواكبة هذا الحجم الكبير من الطلبات، تعتمد أفضل الممارسات الحديثة في الحوسبة عادةً على إضافة المزيد من المخدمات. موازنة الحمل هي عملية توزيع حركة المرور بشكل متساوٍ عبر مجموعة من الموارد أو الأجهزة في الشبكة لضمان أقصى قدر من الكفاءة والأداء. في الشبكات التقليدية، كانت موازنة الحمل تُعتبر تحديًا معقدًا نظرًا لعدم وجود مرونة كافية للتحكم في مسارات البيانات وتوزيعها بفعالية. ومع ظهور الشبكات المعرفة برمجياً (SDN)، تغيّرت ديناميكيات الشبكات بشكل جذري، حيث تم فصل طبقة التحكم عن طبقة التوجيه، مما أتاح إمكانية التحكم المركزي في حركة المرور عبر الشبكة. في بيئة SDN، يُمكن تحقيق موازنة الحمل بشكل أكثر فعالية بفضل القدرة على برمجة سلوك الشبكة ديناميكياً. يمكن لمتحكمات SDN المركزية تحليل تدفق البيانات بشكل مستمر وتوزيع الأحمال بناءً على المعطيات اللحظية وظروف الشبكة. هذه المرونة تجعل من الممكن تحسين استخدام الموارد، تقليل زمن التأخير، ومنع الاختناقات التي قد تحدث في حالة زيادة حركة المرور على موارد معينة. استخدام موازنة الحمل في SDN لا يساهم فقط في تحسين أداء الشبكة، ولكنه يتيح أيضاً إمكانية تطبيق سياسات محددة لكل تدفق بيانات على حدة، مما يوفر إدارة أفضل للموارد وتحقيق تجربة استخدام أكثر استجابة. وبفضل القدرة على التكيّف مع تغييرات الشبكة في الوقت الفعلي، تُعد موازنة الحمل في SDN خطوة حيوية نحو تطوير شبكات أكثر كفاءة وموثوقية.

يتحدث الفصل الأول الدراسة النظرية والمرجعية حيث تم تناول تعريف الشبكات المعرفة برمجيا وموازنة الحمل وأهداف موازنة الحمل والمقاييس التي تستخدم في الاختبارات وأنواع موازنة الحمل من حيث الطبقة التي يتم فيها تطبيق موازنة الحمل و الخوارزميات المستخدمة ومن ثم الإشارة الى الخوارزميات المقترحة للمقارنة معها بينما الفصل الثاني يتحدث عن أهداف المشروع والمتطلبات الوظيفية والغير الوظيفية التي يجب ان يحققها النظام بينما يتحدث الفصل الثالث عن تصميم الخوارزميات وآلية عملها ويتحدث الفصل الرابع عن بيئة العمل وأدوات التنفيذ بينما يتحدث الفصل الخامس عن الاختبارات والنتائج وأخيرا يتحدث الفصل السادس عن الخاتمة والافاق المستقبلية.

الفصل الأول: الدراسة النظرية والمرجعية

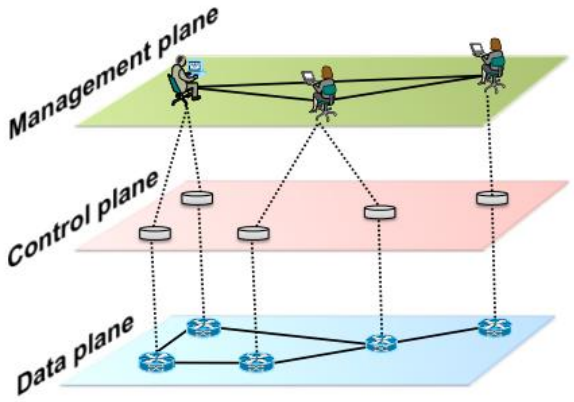
نقدم في هذا الفصل تعريف الشبكات المعرفة برمجيا وبماذا تختلف عن الشبكات التقليدية ومن ثم الحديث عن موازنة الحمل والخوارزميات المستخدمة واهداف موازنة الحمل والمقاييس التي تستخدم في الاختبارات وأنواع موازنة الحمل من حيث الطبقة التي يتم فيها تطبيق موازنة الحمل. كما نقدم في هذا الفصل الدراسات المنجزة سابقا والخوارزميات المستخدمة.

1.1. الشبكات المعرفة برمجيا

1.1.1. تعريف الشبكات المعرفة برمجيا

في الشبكات التقليدية المعتمدة على بروتوكول الإنترنت (IP)، تكون طبقتي التحكم وتمرير المعطيات مرتبطتين بشكل وثيق، مضمنتين في نفس أجهزة الشبكة، وتكون البنية كلها لا مركزية بشكل كبير. كان هذا يعتبر مهماً لتصميم الإنترنت في الأيام الأولى: كان يبدو أنه أفضل طريقة لضمان مرونة الشبكة، وهو هدف تصميمي حاسم. في الواقع، كانت هذه الطريقة فعالة جداً من حيث أداء الشبكة، مع زيادة سريعة في معدل الخط وكثافة المنافذ.

يمكن تقسيم الشبكات الحاسوبية إلى ثلاث طبقات وظيفية: طبقة تمرير المعطيات، وطبقة التحكم، وطبقة الإدارة (الشكل 1). تتوافق طبقة تمرير المعطيات مع أجهزة الشبكة، التي تكون مسؤولة عن توجيه المعطيات بكفاءة. تمثل طبقة التحكم البروتوكولات المستخدمة لملء جداول التوجيه في عناصر طبقة المعطيات. تشمل طبقة الإدارة خدمات البرمجيات، مثل الأدوات المعتمدة على SNMP [1]، المستخدمة لمراقبة وتكوين الوظائف التحكمية عن بعد. يتم تعريف سياسة الشبكة في طبقة الإدارة، تقوم طبقة التحكم بفرض السياسة، وتنفذها طبقة المعطيات من خلال توجيه المعطيات وفقاً لذلك.

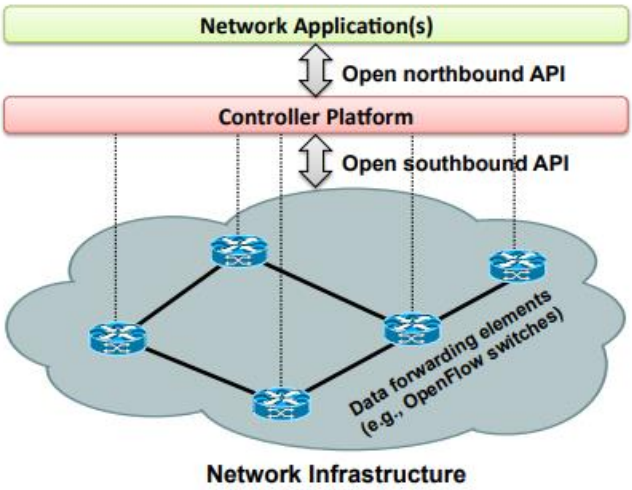


الشكل 1 الطبقات الوظيفية بالشبكة

تمت صياغة مصطلح الشبكات المعرفة برمجياً (SDN) في الأصل لتمثيل الأفكار والعمل حول OpenFlow في جامعة ستانفورد [2]. وهي نموذج شبكي ناشئ يبعث الأمل في تغيير قيود البنية التحتية الحالية للشبكات. كما تم تعريفه في الأصل، يشير SDN إلى بنية شبكية حيث تتم إدارة حالة التوجيه في طبقة تمرير المعطيات بواسطة طبقة تحكم عن بُعد منفصلة عن طبقة المعطيات وذلك من خلال فصل منطق التحكم في الشبكة (طبقة التحكم) عن أجهزة التوجيه والمبدلات الأساسية التي توجه حركة مرور المعطيات (طبقة تمرير المعطيات). مع فصل طبقتي التحكم وتمرير المعطيات عن بعضهم، تصبح المبدلات

الشبكية أجهزة توجيه بسيطة، ويتم تنفيذ منطق التحكم في وحدة تحكم مركزية منطقية (او نظام التشغيل الشبكي). مما يبسط فرض السياسات وإعادة تكوين وتطوير الشبكة. يتم عرض رؤية مبسطة لهذه البنية في الشكل 2. من المهم التأكيد على أن النموذج البرمجي المركزي منطقياً لا يفترض نظاماً مركزياً فيزيائياً. في الواقع، فإن الحاجة إلى ضمان مستويات مناسبة من الأداء وقابلية التوسع والاعتمادية تستبعد مثل هذا الحل. بدلاً من ذلك، تعتمد تصاميم SDN على مستوى الإنتاج على طبقات تحكم مركزية موزعة فيزيائياً.

يمكن تحقيق فصل طبقة التحكم عن طبقة تمرير المعطيات من خلال API معرفة جيداً بين المبدلات ووحدة التحكم. تتحكم وحدة التحكم مباشرة في حالة عناصر طبقة المعطيات عبرAPI، كما هو موضح في الشكل 2. المثال الأبرز لمثل هذه الواجهة البرمجية هو بروتوكول OpenFlow. يحتوي مبدل OpenFlow على واحد أو أكثر من جداول التعامل مع الحزم (جدول الدفق). كل قاعدة تطابق جزءاً من حركة المرور وتقوم بأداء إجراءات معينة (إسقاط، توجيه، تعديل، إلخ) على حركة المرور. اعتماداً على القواعد المثبتة من قبل تطبيق وحدة التحكم، يمكن لمبدل OpenFlow بناءً على تعليمات وحدة التحكم أن يتصرف كجهاز توجيه، مبدل، جدار ناري، أو أداء أدوار أخرى (مثل موازن الحمل).



الشكل 2 بنية الشبكات المعرفة برمجيا

أحد النتائج المهمة لمبادئ الشبكات المعرفة برمجياً هو الفصل بين تعريف سياسات الشبكة، وتنفيذها في تبديل الأجهزة، وتوجيه حركة المرور. هذا الفصل هو المفتاح لتحقيق المرونة المرغوبة، حيث تتغلب على مشكلة التحكم في الشبكة إلى أجزاء قابلة للحل، مما يسهل إنشاء وتقديم تجريدات جديدة في الشبكات، ويبسط إدارة الشبكة ويسهل تطورها وابتكارها.

سنقدم في هذا القسم تعريف أكثر دقة للشبكات المعرفة برمجيا

تعتمد الشبكات المعرفة برمجيا على أربع ركائز أساسية وهي:

1- فصل طبقتي التحكم وتمرير المعطيات :يتم إزالة وظيفة التحكم من أجهزة الشبكة التي ستصبح عناصر توجيه (حزم) بسيطة.
2- قرارات التوجيه تعتمد على التدفقات بدلاً من الوجهة :يتم تعريف الدفق بشكل عام بواسطة مجموعة من قيم حقول الحزمة التي تعمل كمعيار مطابقة (مرشح) ومجموعة من الإجراءات (التعليمات). في سياق SDN/OpenFlow، يُعرف الدفق على أنه تسلسل من الحزم بين مصدر ووجهة. جميع حزم الدفق تتلقى سياسات خدمة متطابقة في أجهزة التوجيه [3]. تتيح تجريد الدفق توحيد سلوك أنواع مختلفة من أجهزة الشبكة، بما في ذلك الموجهات، والمبدلات، وجدران الحماية. يتيح برمجة الدفق مرونة غير مسبوقة، مقيدة فقط بقدرات جداول الدفق المنفذة [4].
3- نقل منطق التحكم إلى كيان خارجي :يتم نقل منطق التحكم إلى كيان خارجي يعرف بمتحكم SDN أو نظام تشغيل الشبكة (NOS). هو منصة برمجية وتوفر الموارد والتجريدات الأساسية لتسهيل برمجة أجهزة التوجيه بناءً على رؤية شبكة منطقية مركزية ومجردة. لذلك، فإن غرضه مشابه لغرض نظام التشغيل التقليدي.
4- إمكانية برمجة الشبكة من خلال التطبيقات البرمجية :يتم برمجة الشبكة من خلال تطبيقات برمجية تعمل على قمة NOSوتتفاعل مع أجهزة طبقة المعطيات الأساسية. هذه خاصية أساسية في SDN وتعتبر قيمتها الرئيسية.

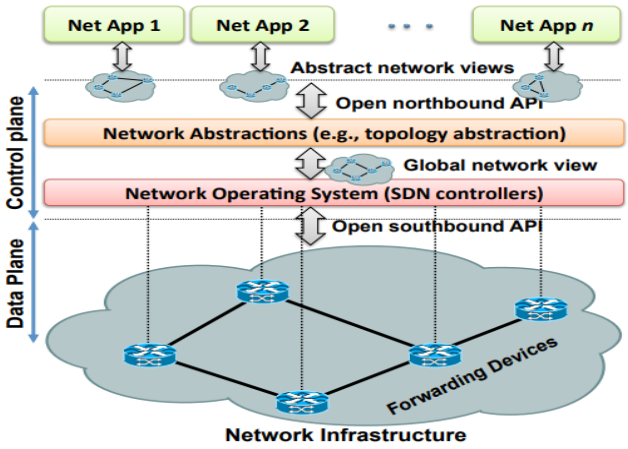
نلاحظ أن المركزية المنطقية لمنطق التحكم، على وجه الخصوص، توفر عدة فوائد إضافية. أولاً، يكون من الأسهل وأقل عرضة للأخطاء تعديل سياسات الشبكة من خلال لغات عالية المستوى ومكونات برمجية، مقارنة بالتكوينات المحددة للأجهزة ذات المستوى المنخفض. ثانياً، يمكن لبرنامج التحكم أن يتفاعل تلقائياً مع التغييرات العشوائية في حالة الشبكة وبالتالي يحافظ على السياسات عالية المستوى سليمة. ثالثاً، إن مركزية منطق التحكم في وحدة تحكم تتمتع بمعرفة شاملة بحالة الشبكة تُبسط تطوير وظائف وخدمات وتطبيقات شبكية أكثر تطوراً.

باتباع مفهوم SDN، يمكن تعريف SDN من خلال ثلاثة تجريدات أساسية: (i) التوجيه، (ii) التوزيع و(iii) التحديد. في الواقع، تعتبر التجريدات أدوات أساسية للبحث في علوم الكمبيوتر وتكنولوجيا المعلومات، وهي ميزة موجودة بالفعل في العديد من هياكل وأنظمة الكمبيوتر [5].

- يجب أن يسمح تجريد إعادة التوجيه بأي سلوك توجيهي تريده تطبيقات الشبكة (برنامج التحكم) مع إخفاء تفاصيل الأجهزة الأساسية. يعتبر OpenFlow تحقيقاً لمثل هذا التجريد، والذي يمكن اعتباره مكافئاً لـ "برنامج تشغيل الجهاز"(device-driver) في نظام التشغيل.
- يجب أن تحجب تجريدية التوزيع تطبيقات SDN عن تقلبات الحالة الموزعة، مما يجعل مشكلة التحكم الموزع مشكلة مركزية منطقياً. يتطلب تحقيق ذلك وجود طبقة توزيع مشتركة، والتي توجد في SDN في نظام تشغيل الشبكة

(NOS). لهذه الطبقة وظيفتان أساسيتان. أولاً، هي مسؤولة عن تثبيت أوامر التحكم على أجهزة التوجيه. ثانياً، تجمع معلومات الحالة حول طبقة التوجيه (أجهزة الشبكة والوصلات الشبكية)، لتقديم رؤية شاملة للشبكة لتطبيقات الشبكة.

- التجريد الأخير هو التحديد، والذي يهدف إلى تمكين تطبيقات الشبكة من تحديد السلوك المطلوب للشبكة دون الحاجة إلى التعامل مع تفاصيل تنفيذه. يمكن تحقيق ذلك من خلال تقنيات الافتراضية ولغات برمجة الشبكات، حيث تقوم هذه الأدوات بترجمة التكوينات المجردة التي تصفها التطبيقات، استناداً إلى نموذج مبسط للشبكة، إلى تكوينات مادية تتماشى مع الرؤية الشاملة التي يقدمها متحكم SDN. يوضح الشكل 3 هيكلية SDN والمفاهيم الأساسية التي يقوم عليها.



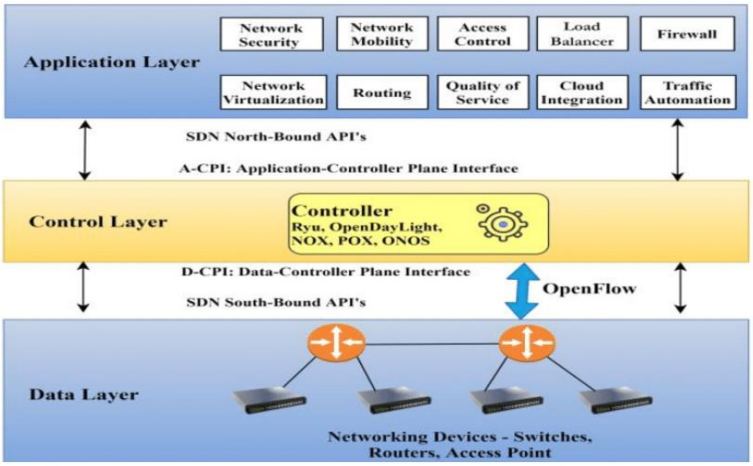
الشكل 3 التجريدات الأساسية الخاصة ببنية الشبكات المعرفة برمجيا

كما ذُكر سابقاً، فإن الربط القوي بين طبقتي التحكم والمعطيات قد جعل من الصعب إضافة ميزات جديدة إلى الشبكات التقليدية. إن الربط بين طبقتي التحكم والمعطيات (وتضمينها الفعلي في عناصر الشبكة) يجعل تطوير ونشر ميزات شبكية جديدة (مثل خوارزميات التوجيه) أمراً صعباً للغاية، حيث يتطلب تعديل طبقة التحكم لجميع أجهزة الشبكة من خلال تثبيت البرامج الراسخة (firmware) الجديدة، وفي بعض الحالات، تحديث الأجهزة فيزيائيا. وبالتالي، يتم تقديم الميزات الشبكية الجديدة عادةً من خلال معدات متخصصة ومكلفة وصعبة التكوين المعروفة أيضاً بالصناديق الوسطى(middleboxe) مثل موازن الحمل، وأنظمة كشف التسلل (IDS)، وجدران الحماية. تحتاج هذه الصناديق إلى وضعها بشكل استراتيجي في الشبكة، مما يجعل من الصعب تغيير بنية الشبكة وتكوينها ووظيفتها لاحقاً.

بالمقابل، تقوم SDN بفصل طبقتي التحكم عن أجهزة الشبكة وتجعلها كيانا خارجيا: نظام تشغيل الشبكة أو وحدة تحكم SDN. هذه الطريقة لها عدة مزايا:

- **تسهيل برمجة التطبيقات**: يصبح من الأسهل برمجة هذه التطبيقات حيث يمكن تبادل التجريدات المقدمة من منصة التحكم و/أو لغات برمجة الشبكات.
- **استفادة جميع التطبيقات من نفس معلومات الشبكة**: يمكن لجميع التطبيقات الاستفادة من نفس المعلومات الشبكية (الرؤية الشاملة للشبكة)، مما يؤدي (ربما) إلى اتخاذ قرارات سياسية أكثر اتساقاً وفعالية مع إعادة استخدام وحدات برمجيات طبقة التحكم.
- **إمكانية اتخاذ الإجراءات من أي جزء من الشبكة**: يمكن لتلك التطبيقات اتخاذ إجراءات (أي إعادة تكوين أجهزة التوجيه) من أي جزء من الشبكة، مما يلغي الحاجة إلى وضع استراتيجية دقيقة حول موقع الوظائف الجديدة.
- **تكامل التطبيقات المختلفة**: يصبح دمج التطبيقات المختلفة أكثر سهولة [6]. على سبيل المثال، يمكن دمج تطبيقات موازن الحمل والتوجيه بشكل متتابع، مع إعطاء قرارات موازنة الأحمال أولوية على سياسات التوجيه.

بشكل عام، يمكن تقسيم بنية الشبكة المعرفة برمجيا من الأسفل إلى الأعلى إلى ثلاث طبقات: طبقة تمرير المعطيات وطبقة التحكم وطبقة التطبيقات كما هو موضح بالشكل 4.



الشكل 4 طبقات الشبكات المعرفة برمجيا

- طبقة تمرير المعطيات

تتكون هذه الطبقة من العديد من المبدلات الشبكية المعرفة برمجياً والمتصلة ببعضها البعض عبر وسائل سلكية أو لاسلكية. كل مبدل هو جهاز بسيط مسؤول عن توجيه حزم الشبكة ويحتوي على جدول توجيه يسمى جدول الدفق. يحتوي هذا الجدول على آلاف القواعد المستخدمة في اتخاذ قرارات التوجيه.

كل قاعدة داخل جدول الدفق تتكون من ثلاثة حقول: الإجراء، العداد، والنمط. يتكون حقل النمط من عدة قيم من حقل ترويسة الحزمة لأغراض المطابقة. عندما تصل حزم المعطيات، يقوم المبدل بالبحث في جدول الدفق للعثور على إدخال يتطابق مع حقول الحزمة الواردة. عند العثور على إدخال مطابق، يزيد قيمة العداد ويتم تنفيذ الإجراء المرتبط. إذا لم يتم العثور على إدخال مطابق، سيقوم المبدل بإخطار وحدة التحكم للحصول على المساعدة أو ببساطة إسقاط الحزمة كما هو موضح في الشكل 5.

- طبقة التحكم

تعتبر طبقة التحكم العقل المدبر للشبكات المعرفة برمجياً (SDN)، حيث تتولى إدارة الشبكة والتحكم في جميع مكوناتها. هذه الطبقة توفر مستوى من التجريد يُمكن من خلاله التحكم بالشبكة بطريقة مركزية، مما يجعلها أكثر مرونة وكفاءة. يُطلق على الجهاز أو العقدة الشبكية التي تقوم بهذه الوظائف اسم "المتحكم" (Controller)، وهو جهاز مادي منفصل يعمل ببرنامج مخصص مصمم خصيصًا لهذا الغرض.

المتحكم يعمل كحلقة وصل بين التطبيقات التي تحدد سياسات الشبكة والمكونات المادية للشبكة مثل المبدلات (switches) وأجهزة التوجيه (routers). يتم هذا التواصل بين المتحكم والمبدلات عبر واجهة تسمى "الجسر الجنوبي" (Southbound Bridge)، وهي عبارة عن بروتوكول أو مجموعة من البروتوكولات التي تسمح للمتحكم بإرسال تعليمات وتوجيهات إلى أجهزة الشبكة. أشهر بروتوكول يستخدم في هذه العملية هو OpenFlow، الذي يتيح للمتحكم إمكانية برمجة سلوك المبدلات على مستوى تدفق الحزم، وبالتالي تعديل مسار المعطيات عبر الشبكة وفقًا للسياسات المحددة.

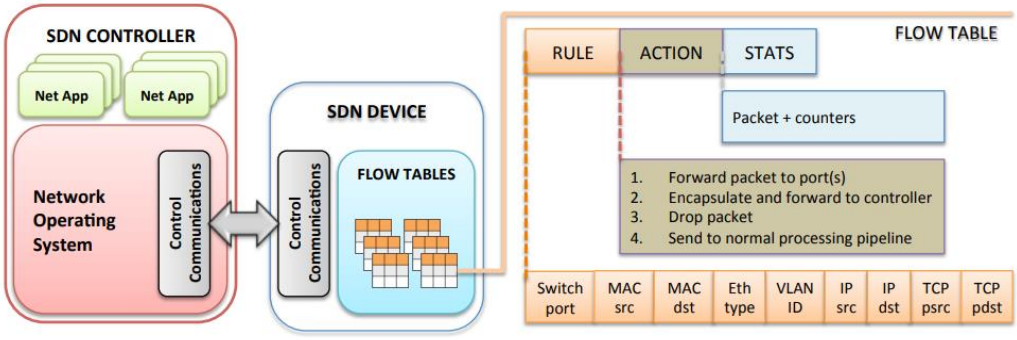
يمتلك المتحكم رؤية شاملة لكامل طوبولوجيا الشبكة في طبقة التوجيه، بما في ذلك المبدلات والتوصيلات. تعتمد العديد من بروتوكولات التوجيه، مثل OSPF وBGP، على المتحكم. لذلك، تعتمد جميع قرارات التوجيه ضمن طبقة التوجيه على التعليمات الصادرة عن المتحكم. في البداية، عندما تم تقديم معيار SDN، تم تصميم OpenFlow لاستيعاب متحكم واحد لسهولة التنفيذ. ومع ذلك، فإن هذا أدى إلى مشكلة نقطة الفشل واحدة (SPF). فيما بعد، تم تنفيذ بنية شبكة معرفة برمجياً تدعم المتحكمات الموزعة، مثل Floodlight وNOX وOpenDaylight، بهدف موازنة الحمل وتحسين التوافر وقابلية التوسع في موارد الشبكة.

في بنية متعددة المتحكمات، يكون كل متحكم مسؤولاً عن إدارة مجموعة من المبدلات. لتحقيق الاتساق في حالة الشبكة والتعاون بين المتحكمات، يمكن لمتحكم واحد التواصل مع المتحكمات الأخرى عبر واجهات تسمى الجسر الغربي والجسر الشرقي.

- طبقة التطبيقات

تسمح طبقة التطبيقات لمسؤولي الشبكات بالاستجابة بسرعة لمتطلبات الأعمال المختلفة. هناك العديد من التطبيقات البرمجية المبنية على متحكمات الشبكات المعرفة برمجياً، مثل الشبكات الافتراضية، واكتشاف الطوبولوجيا، ومراقبة دفق الشبكة، وتعزيزات الأمان، وموازنة الحمل، والعديد من التطبيقات الأخرى.

تتواصل طبقة التطبيقات مع طبقة التحكم من خلال واجهة برمجية تسمى الجسر الشمالي، مثل REST API. تقوم طبقة التحكم بتجريد موارد الشبكة الفيزيائية وتقديمها إلى طبقة التطبيقات. وهذا يعني أن مشغلي الشبكة يمكنهم تغيير مسار الحزم



الشكل 5 أجهزة SDN وأجزاء قواعد جدول الدفق

الرقمية باستخدام التطبيقات البرمجية الموجودة على المتحكمات فقط، دون الحاجة إلى التنقل بين كل مبدل وموجه لإعادة ضبطهم واحداً تلو الآخر.

2.1.1. تصنيف معمارية متحكم SDN

(1) البنية المعمارية المركزية (Centralized Architecture)

يحتوي هيكل SDN المركزي على وحدة تحكم واحدة، مبدلات ومخدمات متعددة. تقوم وحدة التحكم بإدارة تخصيص حركة المرور في الشبكة بأكملها. وفقًا لاستراتيجية توزيع الحمل المعتمدة، تجمع وحدة التحكم معلومات الحمل الخاصة بالشبكة وتقوم بضبطها في الوقت الفعلي وتوزيع الحمل بشكل ديناميكي على مخدمات ووصلات الشبكة. يمكن تجنب الازدحام غير الضروري عن طريق توجيه حركة المرور في الشبكة إلى الخوادم الغير مستغلة بالكامل. يمكن أيضًا تصحيح توازن حمل الوصلات، الذي يزيد من وقت الانتظار وزمن الإرسال، باتخاذ قرارات مناسبة لتوزيع التدفقات (الحمل)على المسارات الشبكية[13]. يوفر الهيكل المركزي للتحكم في شبكات SDN رؤية شاملة للشبكة مما يسمح بتحسين أداء الشبكة واستخدام الموارد. مع زيادة كمية التدفقات، يمكن أن تصبح وحدة التحكم عائق كبير للشبكة بأكملها.

(2) البنية المعمارية الموزعة (Distributed Architecture)

في الشبكات الكبيرة مثل مراكز البيانات وشبكات مزودي الخدمة التي تضم آلاف المبدلات، يعتبر الاعتماد على وحدة تحكم واحدة محفوفًا بالمخاطر بسبب إمكانية حدوث نقطة فشل واحدة (SPF). أظهرت الأبحاث في تصميم شبكات SDN بعض القيود المرتبطة باستخدام وحدات تحكم مركزية مادية، خاصة من حيث الاستجابة والموثوقية وقابلية التوسع. لتحقيق توازن فعال في الحمل داخل شبكة SDN، من الضروري تنفيذ طبقة تحكم موزعة لضمان قابلية التوسع، بالإضافة إلى الحاجة إلى رؤية في الوقت الفعلي للشبكة لتحسين أداء خوارزميات التوازن. كحل لهذه التحديات، تم اقتراح اعتماد التحكم الموزع في SDN.

يمكن تصنيف طبقة التحكم المتعددة إلى طريقتين مختلفتين للتنفيذ: المركزية المنطقية أو الموزعة المنطقية. الأولى هي طبقة التحكم الموزعة ماديًا التي تتكون من عدة وحدات تحكم ولكنها تعمل جميعها على طبقة تحكم مركزية منطقية، وهي بنية حيث تقوم وحدات التحكم بمزامنة رؤاها المحلية للشبكة وتعلن حالتها لبعضها البعض. يسمح لهم ذلك ببناء رؤية متماسكة وموثوقة في الوقت الفعلي للشبكة واتخاذ قرارات مثلى. ومع ذلك، قد تؤدي تبادلات المعلومات المتكررة إلى زيادة الحمل على الشبكة مما يؤثر على كفاءة الوقت لقرارات إدارة الحمل بما في ذلك قرارات توازن الحمل. لذلك، تقترح بعض الدراسات وحدة التحكم الموزعة بالكامل وهي بنية تتكون من عدة وحدات تحكم وكل واحدة تتحكم في جزء مختلف من الشبكة. في هذه الطريقة، تسمى عملية توازن الحمل بالقرار الموزع حيث تكون لوحدات التحكم في الشبكة نفس السلطات والواجبات وتقسم الحمل بالتساوي. لكل وحدة تحكم رؤية لمجالها وتتخذ قرارات توازن الحمل المطلوبة محليًا مما يقلل من الحمل الزائد للاتصالات [14].

2.1. تعريف موازن الحمل

موازنة الحمل هو الطريقة المستخدمة لتوزيع وتخصيص العملاء الواردين، الطلبات، والمهام بكفاءة إلى الموارد المتاحة في الشبكة. هدف توزيع الحمل هو تحسين استخدام الموارد لتجنب الازدحام وفقدان البيانات. يمكن تنفيذه على الأجهزة وكذلك على البرمجيات. يجب أيضًا أن يكون هناك نظام نسخ احتياطي لموازن الحمل لمنعه من أن يصبح نقطة فشل واحدة(SPF).

لبنية المتحكم تأثير كبير على أداء الشبكات المعرفة برمجياً (SDN). تم اقتراح العديد من بنى المتحكمات في الأبحاث، من بينها المتحكم المركزي، المتحكمات الموزعة ولكنها مركزية منطقياً، والمتحكمات الموزعة بالكامل. لكل بنية مزاياها وعيوبها. على سبيل المثال، أظهرت البنية المركزية ذات المتحكم الواحد قيوداً في قابلية التوسع. أما النشر الموزع الذي يعتمد على متحكمات متعددة، فإنه يحسن من قابلية التوسع لطبقة التحكم ولكنه يحمل خطرًا عاليًا من توزيع غير متساوٍ للحمل، حيث يمكن أن تكون بعض المتحكمات مثقلة بالعمل بينما قد تكون الأخرى غير مستخدمة. لذلك، من الضروري استخدام خوارزمية موازنة الأحمال المناسبة لكل بنية لإنشاء نظام إدارة حمل أمثل [7].

يمكن تصنيف خوارزميات توزيع الحمل إلى طرق ثابتة أو طرق ديناميكية [8]. الأولى تناسب فقط البيئة المستقرة لأنها تفتقر إلى المرونة والقدرة على التكيف مع التغيرات الديناميكية في الشبكات. اما الثانية فيتم تعديل توزيع الحمل بشكل مستمر بناءً على الظروف الفعلية للشبكة، مثل تغيرات حركة المرور أو توفر الموارد. يتم تحديث قواعد التوجيه بمرونة لتتوافق مع هذه التغيرات. في كل من الطريقتين، تكون قواعد التوجيه مثبتة بالفعل في عقدة إدارة الشبكة وبسبب الجانب غير المتوقع من سلوك المستخدمين، فإن الطرق الثابتة محكوم عليها بالفشل. ومع ذلك، فإن توزيع الحمل الديناميكي أكثر كفاءة لأن البيانات المنقولة توزع وفقًا للحالة الحالية للشبكة والقواعد المبرمجة في الشبكة بطريقة ديناميكية.

3.1. أهداف موازن الحمل

بالإضافة إلى ضمان توزيع أفضل للحمل بين موارد الشبكة المتاحة، يهدف توازن الحمل أيضًا إلى ضمان ما يلي:

- القابلية للتوسع[7]: مع زيادة حجم النظام في الشبكات الكبيرة، تصبح مسألة تحقيق القابلية للتوسع أكثر تعقيدًا. في شبكات SDN، يتم تقسيم القابلية للتوسع إلى توسع العناصر الشبكية ووحدة التحكم. تشير الأبحاث إلى أن زيادة عدد المستخدمين والمكونات قد تؤدي إلى مشاكل في أداء وحدة التحكم، مما يتسبب في تأخير تحديثات التوجيه. لحل هذه المشكلة، تم اقتراح بنية وحدة تحكم موزعة مرنة تتوسع مع زيادة التحميل، مما يتيح توزيع الأحمال ديناميكيًا بين وحدات التحكم ويعزز القابلية للتوسع. كما تم اقتراح حلول لمشاركة رؤية الشبكة بين وحدات التحكم المتعددة لضمان التوازن الأفضل للحمل وتلبية الطلب المتزايد على الخدمات.
- المرونة: من المتوقع أن تحقق SDN اتصالات مرنة في الحالات التي تكون فيها الشبكة تحت هجوم، في حالة فشل المكونات وعندما تواجه زيادة في التحميل. لضمان استمرارية الخدمات، يجب أن تكون شبكات SDN موثوقة ويجب أن تحتوي على مكونات احتياطية واستراتيجيات سريعة لإعادة استقرار الشبكة. يُعد توازن الحمل واحدًا من الآليات الرئيسية التي تضمن مرونة الشبكة من خلال توفير سياسات إدارة التحميل التفاعلية والوقائية، المسارات المتعددة والمتنوعة بين عقد الشبكة واستخدام وحدات تحكم متعددة. استخدام هذه الآليات يقلل من فقدان حزم البيانات والتأخير في الشبكة[7].
- استخدام الموارد: يقوم موازن الحمل بحل مشكلة ازدحام الشبكة وتحميل الخوادم الزائد عن طريق إرسال حركة المرور وطلبات الشبكة إلى موارد الشبكة مثل المخدمات والوصلات بطريقة لا تسبب ازدحامًا لتحسين أداء الشبكة. يهدف توازن الحمل إلى تحقيق استخدام فعال وعادل للموارد لتجنب إغراق عنصر ما على حساب عنصر آخر غير مستخدم بشكل كافٍ. لذلك يتم أمثلة استخدام الموارد مثل الوصلات، المبدلات، وحدات التحكم واستخدام الذاكرة عند استخدام خوارزمية توازن الحمل المناسبة. عندما يتم توزيع الحمل بشكل متساوٍ بين الموارد المتاحة المتعددة، يتحسن الأداء العام للشبكة، يقل التأخير ووقت الاستجابة، ويزداد معدل النقل. في [22]، يقترح المؤلفون إطارًا يسمح بعمليات إدارة الموارد التكيفية التي تتضمن إعادة تكوين الموارد المتاحة في وقت قصير. تم تقييم هذه

الطريقة باستخدام مقاييس حركة المرور الحقيقية وأظهرت النتائج تحسينًا كبيرًا في استخدام الوصلات واستهلاك الطاقة[7].

- جودة الخدمة(QoS) : الهدف الرئيسي من توازن الحمل هو توفير جودة الخدمة من طرف إلى طرف في الشبكة. لذلك، فإن توازن الحمل يحسن كفاءة النظام وأداءه بشكل عام. تهدف جودة الخدمة إلى تحقيق تجربة مستخدم أفضل من خلال تجنب التأخير الكبير في النظام، وتحقيق استخدام مثالي للموارد، وزيادة معدل النقل، وتقليل وقت الاستجابة. لكل عنصر في الشبكة، يزداد وقت الاستجابة والتأخير مع زيادة الحمل. لذلك، يتطلب توازن الحمل تمكين توزيع العمل بشكل متساوٍ، وتجنب نقاط الاختناق وسوء إدارة الموارد. لقياس تأثير توازن الحمل على جودة الخدمة، نحدد في وقت لاحق بعض المقاييس مثل التأخير، وقت الاستجابة، معدل فقدان الحزم، ومعدل النقل[7].

4.1. أماكن موازنة الحمل في SDN

1.4.1. موازنة الحمل المركزية Centralized Load-Balancing

مثلما تم تناوله في فقرة تصنيف معمارية متحكم SDN – البنية المعمارية المركزية (Centralized Architecture)، في حالة موازن الحمل المركزي، يمكن تنفيذ موازنة الحمل إما في طبقة تمرير المعطيات أو في طبقة التحكم، باستخدام خوارزميات معينة تعتمد على طبيعة الشبكة ومتطلباتها.

في طبقة تمرير المعطيات (Data Plane) تعتمد فكرة موازنة الحمل على تنظيم حركة البيانات ضمن أجهزة الشبكة مثل servers وhosts.

– موازنة حمل المخدم (Server Load Balancing) هي تقنية تُستخدم لتوزيع حركة المرور والشبكة عبر مجموعة من المخدمات لضمان استمرارية الخدمة وتحسين الأداء. تهدف هذه التقنية إلى منع المخدمات من التحميل الزائد وضمان استغلال الموارد بشكل أمثل. تصنف الخوارزميات المستخدمة على نوعان مع الإشارة إلى مدخلاتها ونقاط ضعفها.:

– الخوارزميات الثابتة (Static): وظيفتها توزيع الحمل بشكل ثابت دون مراعاة التغيرات الديناميكية التي تحصل في الشبكة، مثل خوارزمية الاختيار العشوائي Random selection وخوارزمية وRound Robin وغيرها.

ميزاتها:

- بسيطة في التنفيذ والفهم.
- لا تحتاج إلى موارد كبيرة للمراقبة والتحكم.

عيوبها:

- غير فعالة في التعامل مع التغيرات الديناميكية في حركة المرور.
- قد تؤدي إلى تحميل بعض الخوادم بشكل زائد بينما تبقى أخرى غير مستغلة بشكل كافٍ.

– الخوارزميات الديناميكية (Dynamic): في هذه الطريقة، يتم توزيع حركة المرور بناءً على حالة المخدمات الحالية والحمل الفعلي. تعتمد هذه الطريقة على المراقبة المستمرة للخوادم وتحديث قواعد التوزيع بناءً على البيانات الحالية، مثل خوارزمية Least Response Time وخوارزمية Weighted Round Robin (بأوزان ديناميكية تتبع لحالة الشبكة).

ميزاتها:

- قادرة على التكيف مع التغيرات في حركة المرور بشكل فعال.
- تحسن استجابة النظام والأداء العام.

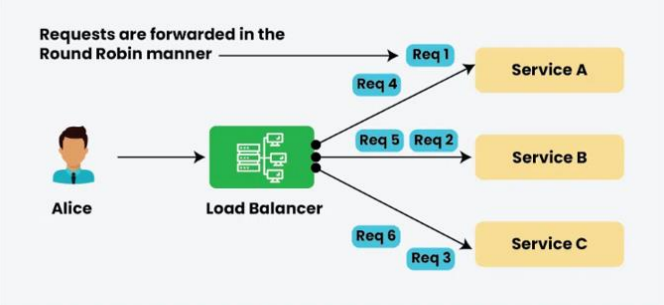
عيوبها:

- أكثر تعقيدًا في التنفيذ والإدارة.
- تحتاج إلى موارد أكبر للمراقبة والتحكم المستمر.

نستعرض في هذا القسم خوارزميات توزيع الحمل المستخدمة حاليًا على مستوى المخدمات.

- خوارزمية توزيع الحمل العشوائية(Random load-balancing) : هذه الخوارزمية هي استراتيجية توزيع حمل ثابتة ولا تتعلق بحالة الشبكة. تقوم بتوزيع الحمل على الموارد بشكل عشوائي. في هذه الاستراتيجية، يتم تعيين حركة المرور الواردة إلى مورد i باحتمالية $1/N$، حيث N هو عدد الخوادم. خلال هذه العملية، لا يمتلك المتحكم أي معلومات عن حالة الشبكة ولا يتابع حركة المرور[9] .
- خوارزمية توزيع الحمل بالأسلوب الدائري(Round Robin load-balancing) : تقوم هذه الخوارزمية بتوجيه حركة المرور الواردة بالتساوي إلى الموارد بناءً على خوارزمية الجدولة الدائرية (Round Robin Algorithm) [10]. يعني ذلك أن الموزع يختار الوجهة بالتتابع ويرسل حركة المرور إلى المورد الأول إذا تم الوصول إلى المورد N، حيثN هو عدد الموارد (المخدمات)كما هو موضح بالشكل 6. تستخدم الخوارزمية الدائرية جدولة تقوم بإعادة توجيه كل حزمة واردة أو طلب إلى الخادم التالي في القائمة حتى اكتمال الدورة. جميع الموارد (أي المخدمات) لها نفس الأولوية دون النظر إلى عدد الحزم الواردة ووقت استجابة الخوادم. هذا يفسر وقت

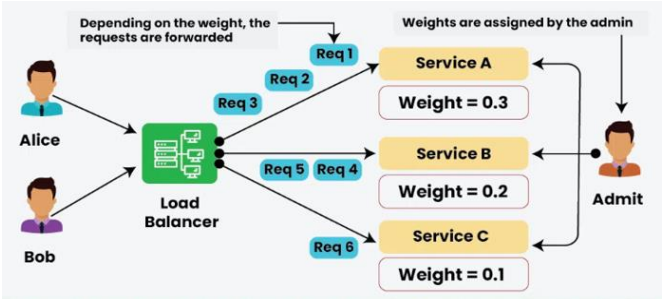
الانتظار الكبير لهذه الطريقة الذي يؤثر على إجمالي وقت التأخير. هذه الخوارزمية مناسبة لضمان توزيع الحمل في شبكة مكونة من خوادم ذات سعة متشابهة.



الشكل 6 خوارزمية توزيع الحمل بالأسلوب الدائري

- خوارزمية توزيع الحمل الدائري الموزن(Weighted Round Robin load balancing) [11]:

هذه الخوارزمية هي ترقية لخوارزمية التوزيع الدائري(Round Robin) . يتم تعيين وزن لكل مورد في البداية، والذي يحدد الحمل الذي سيتم تعيينه لكل منهم وعدد الإدخالات التي سيحصل عليها كل مورد في قائمة الدوران الدائرية. بحال تم وضع أوزان ثابتة تكون خوارزمية ثابتة ولا تأخذ في الاعتبار تطور حركة المرور في الشبكة أو اختلاف الموارد المخصصة للمخدمات. من حيث وقت الاستجابة والتأخير end-to-end، هذه الخوارزمية الأصلية غير محسّنة. من الممكن أن يتم توجيه حركة المرور الواردة إلى الخادم الأبعد مما يتطلب المرور عبر عدد أكبر من المبدلات. وبحال تم وضع أوزان ديناميكية بحيث تتغير هذه الأوزان بناء على حالة الشبكة تكون هذه الخوارزمية ديناميكية وتتطلب مراقبة للشبكة لمعرفة حالة المخدمات وقياس أدائها.



الشكل 7خوارزمية توزيع الحمل بالأسلوب الدائري الموزون

- خوارزمية توزيع الحمل بأقل عدد من الاتصالات(Least connection load-balancing)[10]:

هي طريقة ديناميكية لتوزيع الحمل، حيث يتولى موزع الحمل مراقبة عدد الاتصالات لكل مخدم. قد يزيد أو

ينقص العدد بناءً على إضافة اتصال جديد أو تحريره. يختار موزع الحمل الخادم الذي يحتوي على أقل عدد من الاتصالات النشطة حاليًا. تقوم طريقة أقل عدد من الاتصالات بتوزيع الاتصالات ديناميكيًا بناءً على أداء الخادم في الوقت الفعلي ولكنها غير مدركة لحجم حركة المرور الواردة لكل اتصال، مما يؤثر على أداء الشبكة العام

-موازنة حمل الروابط (Link Load Balancing) هي تقنية تُستخدم لتوزيع حركة المرور عبر مسارات متعددة في الشبكة لضمان استغلال أمثل للروابط المتاحة وتحسين أداء الشبكة. تهدف هذه التقنية إلى تجنب ازدحام الروابط الفردية وضمان نقل البيانات بكفاءة عالية. تُستخدم موازنة حمل الروابط بشكل واسع في شبكات المؤسسات ومراكز البيانات لتحسين الإنتاجية والموثوقية. يمكن تحقيق موازنة حمل الروابط باستخدام العديد من الطرق منها Metaheuristic algorithms وmachine learning، وهي خوارزميات من النوع الديناميكي حيث يتعلق الحل الموضوع بالحالة الحالية لازدحام الروابط والحمل الفعلي. يوجد العديد من الخوارزميات مثل Dijkstra وغيرها الكثير من خوارزميات اختيار أفضل مسار بناء على محددات معينة.

ميزاتها:

- قادرة على التكيف مع التغيرات في حركة المرور بشكل فعال.
- تحسن استجابة النظام والأداء العام.

عيوبها:

- أكثر تعقيدًا في التنفيذ والإدارة.
- تحتاج إلى موارد أكبر للمراقبة والتحكم المستمر.

في طبقة التحكم (Control Plane):

- التنظيم المركزي المنطقي/الموزع ماديا (Logically Centralized/Physically Distributed): في هذا النموذج، يتم توزيع وحدات التحكم ماديًا عبر الشبكة ولكنها تعمل بطريقة منطقية مركزية حيث تتواصل وحدات التحكم مع بعضها البعض لتبادل المعلومات وضمان رؤية موحدة لحالة الشبكة.

ميزاتها:

- زيادة التوافر والموثوقية: حيث يمكن لوحدة تحكم واحدة أن تحل محل أخرى في حالة فشلها.
- تحسين كفاءة إدارة الشبكة من خلال رؤية موحدة لحالة الشبكة.

عيوبها:

- قد يؤدي تبادل المعلومات المستمر بين وحدات التحكم إلى زيادة الحمل على الشبكة.
- تتطلب تنسيقًا عاليًا بين وحدات التحكم لضمان الاتساق.

خطوات عملية موازنة الحمل المركزية في طبقة التحكم:

1. **مراقبة الشبكة:**
   - تقوم وحدة التحكم المركزية بمراقبة حالة الشبكة بشكل مستمر، بما في ذلك حركة المرور، استخدام الموارد، وأداء الشبكة.
2. **تحليل البيانات:**
   - تحليل البيانات المجمعة لتحديد نمط حركة المرور والموارد التي تعاني من التحميل الزائد.
3. **تطبيق الخوارزميات:**
   - استخدام الخوارزميات المحددة لتحديد أفضل طريقة لتوزيع الأحمال. يمكن أن يشمل ذلك توجيه حركة المرور إلى روابط أقل تحميلًا أو إعادة توزيع الأحمال بين الموارد المتاحة.
4. **تحديث قواعد التوجيه:**
   - تحديث قواعد التوجيه في الأجهزة الشبكية بناءً على التحليل والخوارزميات لتوجيه حركة المرور بالطريقة المثلى.
5. **التنظيم والتنسيق:**
   - في حالة التنظيم المركزي المنطقي/الموزع ماديًا، تقوم وحدات التحكم بتبادل المعلومات والتنسيق فيما بينها لضمان اتساق الشبكة وأداءها الأمثل.

موازنة الحمل المركزية في طبقة التحكم تعتمد على خوارزميات محددة ونماذج تنظيمية لضمان توزيع الأحمال بكفاءة وفعالية. تستخدم هذه الآليات لتحسين استخدام الموارد، تقليل زمن الاستجابة، وزيادة موثوقية الشبكة. على الرغم من أنها تتطلب

موارد حسابية أكبر وقد تؤدي إلى زيادة الحمل على الشبكة بسبب تبادل المعلومات المستمر، إلا أنها تعتبر حلاً فعالًا لضمان أداء الشبكات الكبيرة والمعقدة

2.4.1. موازنة الحمل الموزعة (Distributed Load–Balancing):

وحدات تحكم موزعة (Distributed Controllers): تعتمد موازنة الحمل الموزعة على وجود وحدات تحكم متعددة تتعاون فيما بينها لتوزيع الأحمال عبر الشبكة. هذه الطريقة تضمن تحسين استخدام الموارد وزيادة كفاءة الشبكة من خلال توزيع المهام وتحمل الأعباء بشكل متساوٍ ولها نوعين التنظيم الهرمي والتنظيم الافقي.

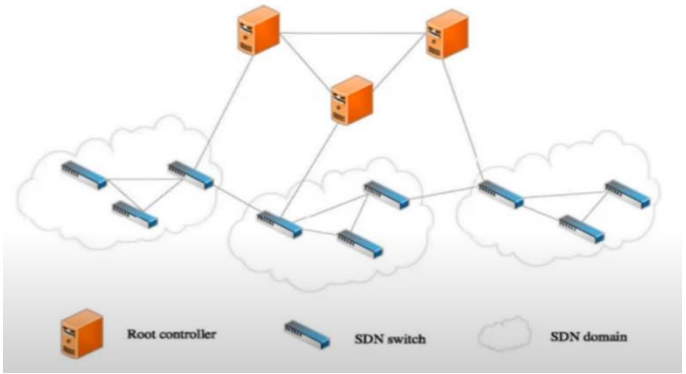
- تنظيم أفقي (Horizontal): في هذا النموذج، تتوزع وحدات التحكم بشكل متساوٍ عبر الشبكة، حيث تتحكم كل وحدة تحكم في جزء محدد من الشبكة دون تداخل كبير مع وحدات التحكم الأخرى.

الميزات:

- تقليل الحمل على كل وحدة تحكم حيث يتم توزيع المهام بشكل متساوٍ.
- تحسين كفاءة الشبكة من خلال تقليل نقاط الفشل الواحدة.(SPF)

العيوب:

- تتطلب تنسيقًا دقيقًا لضمان عدم وجود تعارض أو تداخل في المهام بين وحدات التحكم.
- قد يكون هناك تحديات في ضمان الاتساق الكامل عبر الشبكة.



الشكل 8التوزيع الأفقي لوحدات التحكم

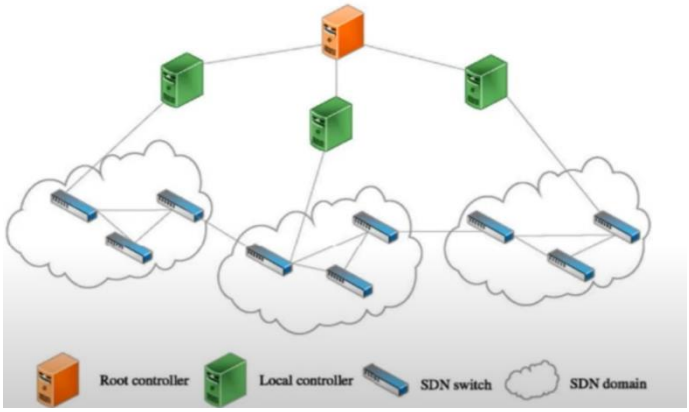
- التوزيع الهرمي (Hierarchical Distribution): في هذا النموذج، يتم تنظيم وحدات التحكم في هيكل هرمي، حيث تكون هناك وحدات تحكم رئيسية تتولى إدارة وحدات تحكم ثانوية. تتولى الوحدات الرئيسية التنسيق العام بين الوحدات الثانوية.

الميزات:

- تحسين التنسيق والإدارة من خلال وجود وحدات تحكم رئيسية تشرف على العملية الكلية.
- تسهيل إدارة الشبكة الكبيرة والمعقدة من خلال توزيع المهام بين الوحدات الرئيسية والثانوية.

العيوب:

- قد تزيد تعقيد النظام نتيجة للتنظيم الهرمي.
- وجود نقاط فشل في الوحدات الرئيسية قد يؤثر على أداء الشبكة بشكل عام.



الشكل 9 التوزع الهرمي لوحدات التحكم

خطوات عملية موازنة الحمل الموزعة:

1. **مراقبة الشبكة:**
   - تقوم كل وحدة تحكم بمراقبة حالة الجزء المحدد من الشبكة الذي تتحكم فيه، بما في ذلك حركة المرور واستخدام الموارد وأداء الشبكة.
2. **تحليل البيانات:**

- تحليل البيانات المجمعة من كل وحدة تحكم لتحديد نمط حركة المرور والموارد التي تعاني من التحميل الزائد في الجزء المحدد من الشبكة.

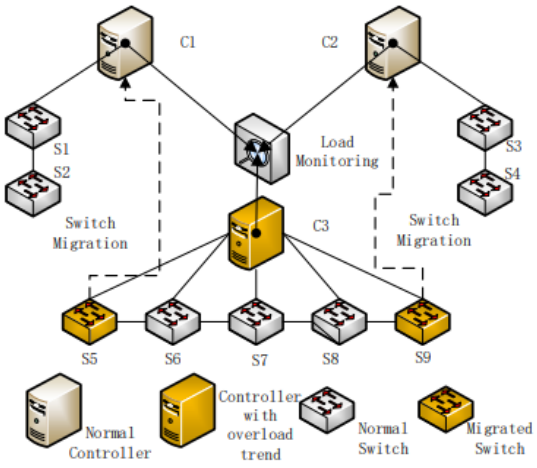
3. **تطبيق الخوارزميات:**
   - استخدام الخوارزميات الموزعة لتحديد أفضل طريقة لتوزيع الأحمال في الجزء المحدد من الشبكة. يمكن أن يشمل ذلك توجيه حركة المرور إلى روابط أقل تحميلًا أو إعادة توزيع الأحمال بين الموارد المتاحة.
4. **تحديث قواعد التوجيه:**
   - تحديث قواعد التوجيه في الأجهزة الشبكية بناءً على التحليل والخوارزميات لتوجيه حركة المرور بالطريقة المثلى.
5. **التنظيم والتنسيق:**
   - في حالة التوزيع الهرمي، تتولى الوحدات الرئيسية التنسيق بين الوحدات الثانوية لضمان اتساق الشبكة وأدائها الأمثل. في حالة التوزيع الأفقي، تتواصل الوحدات فيما بينها لتبادل المعلومات وضمان توزيع الأحمال بشكل متساوٍ.

موازنة الحمل الموزعة تعتمد على التعاون بين وحدات التحكم المتعددة لتحسين استخدام الموارد وزيادة كفاءة الشبكة. يمكن أن تتم هذه الموازنة من خلال التوزيع الأفقي، حيث تتحكم كل وحدة في جزء محدد من الشبكة، أو من خلال التوزيع الهرمي، حيث تتولى وحدات تحكم رئيسية إدارة الوحدات الثانوية. كلا النموذجين يهدفان إلى تحسين أداء الشبكة وتقليل نقاط الفشل، مع بعض التحديات المرتبطة بالتنسيق وضمان الاتساق الكامل.

3.4.1. جوانب أخرى من موازنة الحمل في طبقة المعطيات والتحكم

Switch Migration (1)

يتم نقل المبدلات بين وحدات التحكم لتوزيع الحمل بشكل متوازن، مما يقلل من خطر وجود نقطة فشل واحدة ويحسن من استجابة الشبكة بشكل عام. عندما تولد وحدة التحكم حملًا زائدًا، يكون ذلك عادةً لأن عددًا كبيرًا من المبدلات لا يمكنها العثور على مدخل تدفق مطابق(flow entry) للحزمة الجديدة التي وصلت للتو، مما يؤدي إلى إرسال رسالة "packet-in" إلى وحدة التحكم الافتراضية. تشارك كل وحدة تحكم في طبقة التحكم نفس الرؤية للشبكة ويحقق توازن الحمل من خلال النقل الديناميكي للمبدلات. كما هو موضح في الشكل 10 التوازن الديناميكي للحمل. عندما يكون لدى وحدة التحكم في طبقة التحكم (وحدة التحكم C3) حمل زائد، ستدخل مرحلة switch migration [16]، وسيتم نقل بعض من مبدلات وحدات التحكم الذي يكون عليه حمل زائد (المبدل S5 والمبدل S9) إلى وحدات التحكم الأقل ازدحامًا نسبيًا (وحدة التحكم C2 ووحدة التحكم C3).



الشكل 10هجرة المبدلات switch migration

5.1. موازنة الحمل في وحدات تحكم SDN

تم اقتراح وحدات تحكم SDN متعددة من قبل المطورين، ولكل منها تطبيقها الخاص ومنصتها الخاصة ودرجة قابليتها للتوسع. بعضها خاص ومُسوق، والبعض الآخر مفتوح المصدر.

ولكن جميعها تهدف إلى إنشاء سياسة معيارية مركزية للشبكة لتحقيق إدارة حركة مرور مثالية.

يلخص الشكل 8 وحدات التحكم هذه وما إذا كانت تحقق أهداف موازنة الحمل المذكورة سابقًا بينما يصنفها الشكل 9 وفقًا لبنيتها ولغة البرمجة الخاصة بها.

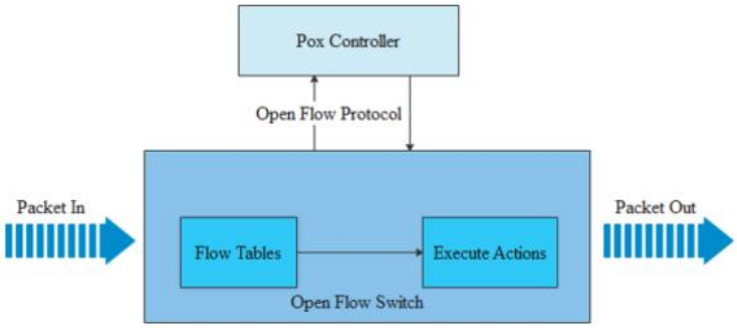
Controller	Scalability	Resilience	Rssource utilization	QoS
Beacon	✓	✓	×	×
Floodlight	✓	✓	✓	✓
Hyperflow	✓	✓	✓	✓
Onix	×	×	×	×
NOX	×	×	×	×
OpenDayLight	✓	✓	✓	✓
ONOS	×	×	×	×
POX	✓	×	✓	×

الشكل 11 أداء وحدات التحكم في موازنة التحميل

Controller	Architecture	Program-mation language
Beacon	Physically centralized	Java
Floodlight	Physically centralized	Java
Hyperflow	Logically centralized, physically distributed	C++
Onix	Logically centralized, physically distributed	C, Python
OpenDayLight	Logically centralized, physically distributed	Java
ONOS	Logically centralized, physically distributed	Java
POX	Physically centralized	Python

الشكل 12 تصنيف وحدات التحكم من حيث البنية ولغة البرمجة المنجزة بها

POX: هذا المتحكم هو متحكم SDN منجز بلغة البرمجة Python [15]. يضمن POX Loadbalancer.py إدارة الحمل ويوازن حركة المرور الواردة بين عناصر الشبكة. يتم اختيار المخدم بناءً على الخوارزمية المراد تنفيذها او المسار الذي سيتم نقل البيانات بناء على خوارزمية مختارة. فيما يلي يوضح الشكل 10 المتحكم POX.



الشكل 13المتحكم POX

6.1. مقاييس موازنة الحمل

لتقييم نجاح خوارزمية التوازن بين الأحمال، يمكن أخذ العديد من المقاييس في الاعتبار. تستخدم الأوراق البحثية مجموعة متنوعة من المقاييس للإشارة إلى مزايا وعيوب الأساليب المختلفة الموجودة. أدناه ملخص أكثر المقاييس استخدامًا فيما يتعلق بالأهداف الأربعة المذكورة في القسم السابق[7].

- قابلية التوسع: في الشبكات الواسعة النطاق، مع زيادة حجم الشبكة، لا يمكن أن تلبي البنية المركزية الحاجة المتعلقة بالقابلية للتوسع، ومن ثم فإن البنى الموزعة التي تستخدم مجالات متعددة ووحدات تحكم متعددة تكون أكثر ملاءمة. تقاس قابلية التوسع لنظام ما بقدرته على الحفاظ على إنتاجيته عندما يتغير حجم الشبكة. أحد المقاييس المستخدمة يعتمد على الإنتاجية. بالنسبة لشبكة تحتوي على N host، يتم تعريفه بالمعادلة $\mathrm{F(N)} = \Phi(\mathrm{N}) \times \frac{\mathrm{T(N)}}{\mathrm{C(N)}}$ حيث $\Phi(\mathrm{N})$ هو معدل نقل البيانات في مستوى التحكم؛ $\mathrm{T(N)}$ هو متوسط وقت استجابة طلبات الشبكة، و$\mathrm{C(N)}$ هو تكلفة نشر مستوى التحكم.
- المرونة: بالنسبة لهذا الهدف، تعتبر فشل وحدات التحكم وتعطل الشبكة من أهم المقاييس. في حالة حدوث فشل في وحدة التحكم، من الضروري ضمان إعادة تعيين عقدها إلى وحدة تحكم احتياطية. يمكن أن يتسبب تعطل الشبكة أيضًا في حدوث مشاكل نتيجة لفشل الروابط أو العقد. يضمن توازن الحمل تعيين العقد بشكل عادل ومتوازن إلى وحدات التحكم في هذه الحالات لتجنب الازدحام وعدم توازن الأحمال بين وحدات التحكم. تم اقتراح استخدام مقياس π_{fail} كمعيار لإدارة فشل وحدات التحكم، والذي يأخذ بعين الاعتبار المسافة إلى وحدات التحكم الاحتياطية.
- استخدام الموارد: لقياس تأثير توازن الأحمال على تحسين استخدام الموارد، تم اقتراح عدة مقاييس:

نسبة استخدام النطاق الترددي: يقيم هذا المقياس نقل البيانات في الشبكة ويعكس حالة الحمل على الروابط. يقوم متحكم SDN بحساب نسبة النطاق الترددي للرابط بناءً على إجمالي البيانات المرسلة عبر منافذ المبدلات المقابلة في فترتين متتاليتين. الفرق بين الفترتين هو حجم البيانات المرسلة أو استخدام النطاق الترددي خلال هذه الفترة. ثم نقسم الأخير على أقصى نسبة للنطاق الترددي للحصول على نسبة استخدام النطاق الترددي.

مقياس استخدام الموارد: هو نسبة استخدام الموارد بشكل عام.

التكاليف الزائدة: هي الذاكرة أو النطاق الترددي أو الموارد الأخرى المستخدمة لنقل معلومات الاتصال، إحصائيات التدفق، أو بيانات التزامن.

مدخلات التوجيه: كلما كان عدد مدخلات التوجيه أقل، والتي تحدد قواعد إدارة الحزم، كان استخدام موارد الذاكرة أكثر فعالية.

- جودة الخدمة: تُستخدم العديد من المقاييس لقياس جودة الخدمة (QoS) في الشبكة. أكثرها أهمية بالنسبة لتوازن الحمل في الشبكات المعرفة برمجياً (SDN) موضحة أدناه:

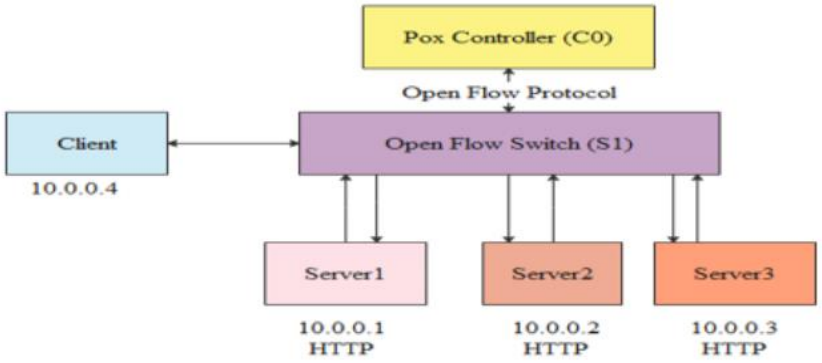
- التأخير (Latency): هو الوقت الذي يستغرقه إرسال حزمة البيانات من المصدر إلى الوجهة عبر الشبكة، ويعتمد على تأخير التبديل وازدحام الشبكة وحجم البيانات المرسلة. لحساب زمن الاستجابة لمسار معين، يُجمع تأخير الإرسال لكل رابط في هذا المسار، مما يعكس درجة ازدحام الشبكة. بالإضافة إلى ذلك، يُستخدم وقت الإكمال (Completion Time) كمقياس لقياس زمن تنفيذ خوارزمية توازن الحمل، حيث يشمل وقت الهجرة للمبدلات (Switch Migration) ووقت التوجيه في الشبكة.
- زمن الاستجابة (Response Time): هو مقياس يشير إلى قدرة الشبكة على تلبية متطلبات جودة الخدمة من حيث التوفر والاستجابة والتأخير. يُعرَّف على أنه الفترة الزمنية بين استلام الطلب وقبوله حتى استجابة الطلب. عندما تكون الشبكة مزدحمة وتعاني من توازن حمل ضعيف، فإن معدل توليد الحزم قد يتجاوز في بعض الأحيان قدرة موارد الشبكة، مما يؤدي إلى زيادة وقت الانتظار. يسمح تقييم وقت الاستجابة في الشبكة بقياس فعالية نظام توازن الحمل بشكل فعال.
- معدل فقدان الحزم (Packet Loss Rate): يشير هذا المعدل إلى نسبة الحزم المفقودة خلال فترة نقلها. يعكس هذا المقياس حالة ازدحام التبديل ومسارات الشبكة، حيث قد تكون المبدلات مشغولة للغاية وقد تصبح الواجهات مثقلة وتبدأ في إسقاط الحزم لمعالجة الحزم الواردة مما يؤدي إلى فقدانها. معدل فقدان الحزم في مسار معين هو نسبة عدد الحزم المفقودة المحسوبة بطرح عدد الحزم المستلمة من عدد الحزم المرسلة.
- معدل النقل (Throughput): هو سرعة معالجة عقد الشبكة وأدائها. يعبر عن كمية البيانات التي تمت معالجتها ونقلها خلال فترة زمنية من المصدر إلى الوجهة (throughput = 1/ responsetime). يحقق توازن الأحمال الناجح الذي يخصص الأحمال إلى العقد ذات السعة المناسبة أقصى معدل نقل.

7.1. موازنة الحمل على مستوى المخدمات (في طبقة المعطيات)

قام (S.Al-Hasnawi,M.Ibrahem)، بتنفيذ خوارزميتي موازنة الحمل على مستوى المخدمات random selection وRound-Robin.

يقوم مبدل OpenFlow بتلقي الحزم ليتم توجيهها الى أحد المخدمات. يعتمد القرار بشأن المخدم الذي سيتم تحديده على القواعد التي يقوم المتحكم بوضعها، التي تعرف من قبل مسؤول الشبكة (Administrator).

لتنجيز موازن الحمل، تم استخدام متحكم من نوع POX controller. حيث يعرض الشكل 14 الطوبولوجيا المستخدمة في الاختبار.



الشكل 14 الطوبولوجيا المستخدمة في اختبار الخوارزميات المطبقة على المخدمات[17]

1.7.1. خوارزمية Random Selection

تم تطبيق خوارزمية الاختيار العشوائي في تنجيز موازن الحمل وهي خوارزمية ثابتة (static) لا تتعلق بوضع الشبكة الحالي بحيث تقوم بتوزيع الحمل على المخدمات بطريقة عشوائية بغض النظر عن حالة المخدم. الرماز الافتراضي للمتحكم POX يأتي مع نموذج افترضي ip_loadbalncer يقوم بتنفيذ موازن حمل بسيط مبنى بخوارزمية الاختيار العشوائي. يضم هذا النموذج صفين وظيفتهما تجريد الدفق ضمن المبدلات وتنجيز خوارزمية اختيار المخدم.

يقوم الأخير بتخزين عناوين IP للمخدمات، للمتحكم والتدفقات النشطة وطرق اختيار المخدمات والتعامل مع الحزم التي تصل الى وحدة التحكم.

تضم الشبكة زبون واحد وثلاثة مخدمات HTTP بسيطة. ومن ثم تم تطبيق موازن الحمل حيث يقوم الزبون بإرسال عدد من الطلبات ويقوم موازن الحمل بتوزيع هذه الطلبات على المخدمات بشكل عشوائي. وتم تقييم أداء هذه الخوارزمية بالنسبة لزمن الاستجابة لكل طلب حيث تم ارسال 500 طلب وتم حساب زمن الاستجابة الوسطي وكان قدره 20 ms [17].

ولكن سنتطرق في الفصل الرابع لدراسة زمن الاستجابة الوسطي ومعدل النقل(Throughput) لأعداد مختلفة من الطلبات.

2.7.1. خوارزمية Round-Robin

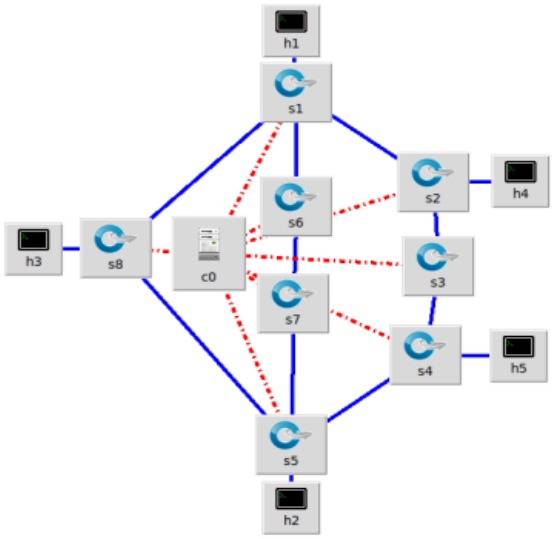
تم تطبيق هذه الخوارزمية في موازنة الحمل حيث تقوم بتوزيع الطلبات بشكل دوري على المخدمات بالترتيب. تم تطبيق هذه الخوارزمية على نفس الشبكة المعرفة أعلاه وبنفس الظروف حيث تم ارسال 500 طلب وتم حساب زمن الاستجابة الوسطي وكان 22 ms حيث كانت اعلى من زمن الاستجابة الوسطي باستخدام خوارزمية random selection [17].

ولكن سنتطرق في الفصل الرابع لدراسة زمن الاستجابة الوسطي والفعالية(Throughput) لأعداد مختلفة من الطلبات.

8.1. موازنة الحمل على مستوى الوصلات (طبقة المعطيات)

تعد موازنة الحمل على مستوى الوصلات في بيئة الشبكات المعرفة برمجياً (SDN) من الأساليب الأساسية التي تهدف إلى تحسين توزيع حركة البيانات عبر الشبكة. من بين الخوارزميات الشائعة التي تم تطبيقها لتحقيق هذا الهدف هي خوارزمية البحث في العرض BFS وخوارزمية البحث بالعمق DFS، حيث تسمح هذه الخوارزميات بتحديد المسارات المثلى بناءً على معايير مختلفة مثل عرض النطاق الترددي والتأخير وغيرها. تعتبر هذه الخوارزميات من الأدوات الفعّالة في تحسين توزيع الحمل وتقليل زمن الاستجابة في الشبكات المعرفة برمجياً. وقد أظهرت هذه الدراسات أن مثل هذه النهج يمكن أن يحقق تحسينات ملحوظة في أداء الشبكة. قام (S.Hossen وآخرون) بتطبيق خوارزميات DFS وBFS لموازنة الحمل على مستوى الوصلات حيث تم اختيار أقصر طريق بناء على قيمة عرض المجال الترددي Band-Width للطريق والمقارنة بينهما من حيث (RTT) حيث ان خوارزمية DFS تعيد لائحة من الوصلات تكون غير مثقلة ومن اجل حساب أقصر مسافة للوصلة، تم استخدام طريقة Open Shortest Path First (OSPF) لتقدير هذه المسافة. بينما تقوم خوارزمية BFS بتفقد اوزان الوصلات اثناء البحث وبالنهاية يتم اختيار انسب طريق بناء على محددات معرفة مسبقا ك التأخير أو عرض المجال الترددي وغيرها.

استنادًا إلى النتائج التي تم التوصل إليها في هذه الدراسة تبين ان الـ RTT كان أفضل عند استخدام خوارزمية DFS مما يحسن أداء الشبكة من ناحية معدل النقل والتأخير [18]. في هذه الدراسة، تم استخدام الشبكة الموضحة بالشكل 15 :



الشكل 15 الطوبولوجيا المستخدمة لتنفيذ خوارزميتي BFS,DFS [18]

سنتطرق بالفصل الرابع لدراسة وسطي زمن الاستجابة لأعداد مختلفة من الطلبات بحيث تم حساب الأوزان بناء على عرض المجال الترددي مرة وعلى التأخير مرة أخرى.

2. الفصل الثاني

هدف المشروع والمتطلبات الوظيفية والغير وظيفية

(Software Requirements Specifications)

نقدم في هذا الفصل المتطلبات الوظيفية وغير الوظيفية والهدف من المشروع.

1.2. أهداف المشروع

يهدف المشروع إلى دراسة مفهوم موازنة الحمل في الشبكات التقليدية والمعرفة برمجيا SDN، حيث سنتمكن من فهم الأساليب الحالية المستخدمة لموازنة الحمل في الشبكات التقليدية والاختلافات في تنفيذ هذه الأساليب في بيئة SDN. كما سيعمل على تطوير خوارزمية موازنة حمل مخصصة تتناسب مع بيئة SDN. سيتم تنفيذ خوارزميات موازنة الحمل على مستوى المخدمات وعلى مستوى الوصلات باستخدام أدوات محاكاة الشبكات مثل Mininet لمحاكاة بيئة SDN لاختبار فعالية هذه الخوارزمية، وأخيراً سيقوم بتحليل الأداء لتقييم أداء الخوارزمية من خلال تجارب المحاكاة.

2.2. المتطلبات الوظيفية

يجب أن يوفر النظام آلية لاختيار المخدم باستخدام خوارزميات موازنة الحمل في طبقة تمرير المعطيات.

يجب أن يوفر النظام آلية لاختيار المسارات باستخدام خوارزميات موازنة الحمل في طبقة تمرير المعطيات.

يجب أن يوفر النظام القدرة على مقارنة أداء الخوارزميات المختلفة من خلال قياس وتحليل الأداء.

3.2. المتطلبات غير الوظيفية

الأداء:

- يجب أن يكون توزيع الحمل له أثر ملحوظ على تحسين أداء الشبكة.
- يجب أن يكون النظام قابلاً للتطبيق على شبكات SDN واقعية.

ديناميكية الطوبولوجيا:

- يجب أن يدعم النظام إضافة المزيد من العقد أو الأجهزة إلى الشبكة دون الحاجة إلى إعادة تصميم الخوارزمية أو بنية الشبكة.
- ينبغي أن تتكيف الخوارزمية بشكل فعال مع التغيرات في حجم الشبكة أو توزيع الحمل، سواء كان ذلك ناتجاً عن زيادة في عدد المستخدمين أو الأجهزة.

قابلية الصيانة:

- يجب أن تكون الخوارزمية ونظام المحاكاة قابلين للتعديل والتطوير بسهولة.
- يجب أن يكون النظام مرناً بما يكفي لدعم تحسينات وتعديلات مستقبلية بدون الحاجة إلى تغييرات كبيرة في البنية.

قابلية الاختبار:

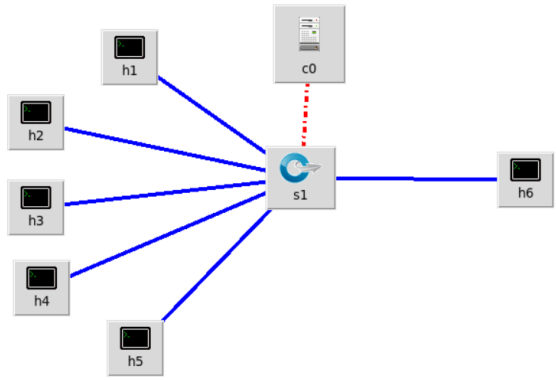
- يجب أن يكون النظام قابلاً للاختبار في سيناريوهات مختلفة من حيث حجم حركة المرور، وتوزيع الحمل بمختلف حالاته كحالات التحميل العالي والتغيّرات المفاجئة في ظروف الشبكة (مثل فشل الروابط أو زيادة الحمل على أحد المسارات).

3. الفصل الثالث: تصميم الحلول

سنقدم في هذا الفصل كيفية تصميم الحلول.

تعتبر خوارزميات التوجيه والتوزيع من العناصر الأساسية في تصميم الشبكات الحديثة، حيث تلعب دورًا محوريًا في تحسين الأداء وزيادة الكفاءة. في هذا السياق، سنستعرض أربع خوارزميات رائدة تُستخدم في توجيه الطلبات إلى المخدمات المتاحة، وهي: خوارزمية الاختيار العشوائي، خوارزمية التوزيع الدائري، خوارزمية التوزيع الدائري الموزون، وخوارزمية أقل عدد من الاتصالات. كل من هذه الخوارزميات تعتمد على مدخلات مختلفة وتقدم مخرجات محددة تهدف إلى تحسين عملية توزيع الحمل على المخدمات.

بالنسبة لموازنة الحمل على مستوى المخدمات، سنقوم بتطبيق الخوارزميات التالية على الطوبولوجيا المبينة بالشكل 16 بحيث هذه الطوبولوجيا تناسب عمل هذه الخوارزميات.



الشكل 16 الطوبولوجيا المستخدمة لاختبار خوارزميات اختيار المخدمات

حيث سيتم تشغيل كل من h1,h2,h3,h4,h5 كمخدمات http بسيطة حيث سيتم عرض كيفية تنفيذهم في الفصل الرابع.

1.3. تصميم خوارزمية الاختيار العشوائي (Random Selection)

المدخلات:

- قائمة المخدمات المتاحة (Servers List): تحتوي على عناوين IP أو معرفات المخدمات التي يمكن توجيه الطلبات إليها.
- الطلب الوارد (Incoming Request): الطلب الذي يحتاج إلى توجيه إلى أحد المخدمات.

المخرجات:

- المخدم المختار (Selected Server): عنوان IP أو معرف المخدم الذي تم اختياره عشوائيًا لخدمة الطلب الوارد.

2.3. تصميم خوارزمية التوزيع الدائري (Round-Robin)

المدخلات:

- قائمة المخدمات المتاحة (Servers List): تحتوي على عناوين IP أو معرفات المخدمات.
- الطلب الوارد (Incoming Request): الطلب الذي يحتاج إلى توجيه إلى أحد المخدمات.
- المؤشر الحالي (Current Index): مؤشر يدل على آخر مخدم تم استخدامه.

المخرجات:

- المخدم المختار (Selected Server): عنوان IP أو معرف المخدم الذي سيتم توجيه الطلب إليه بالتتابع.
- المؤشر المحدّث (Updated Index): المؤشر بعد تحديثه ليشير إلى المخدم التالي في الدورة.

3.3. تصميم خوارزمية التوزيع الدائري الموزون (Weighted Round-Robin)

المدخلات:

- قائمة المخدمات المتاحة مع الأوزان (Servers List with Weights): قائمة تحتوي على عناوين IP أو معرفات المخدمات مع الأوزان المرتبطة بكل مخدم.
- الطلب الوارد (Incoming Request): الطلب الذي يحتاج إلى توجيه إلى أحد المخدمات.
- المؤشر الحالي (Current Index): مؤشر يدل على آخر مخدم تم استخدامه.

المخرجات:

- المخدم المختار (Selected Server): عنوان IP أو معرف المخدم الذي سيتم توجيه الطلب إليه وفقًا للأوزان.
- المؤشر المحدّث (Updated Index): المؤشر بعد تحديثه ليشير إلى المخدم التالي.

4.3. تصميم خوارزمية أقل عدد من الاتصالات (Least Connection)

المدخلات:

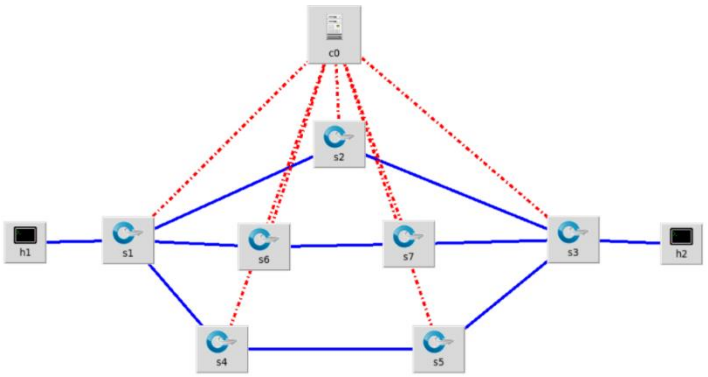
- قائمة المخدمات المتاحة (Servers List): تحتوي على عناوين IP أو معرفات المخدمات.
- عدد الاتصالات الحالية لكل مخدم (Current Connections for Each Server): عدد الاتصالات النشطة لكل مخدم.

- الطلب الوارد (Incoming Request): الطلب الذي يحتاج إلى توجيه إلى أحد المخدمات.

المخرجات:

- المخدم المختار (Selected Server): عنوان IP أو معرف المخدم الذي يحتوي على أقل عدد من الاتصالات النشطة ويكون مستعدًا لخدمة الطلب.

سنقوم بتطبيق خوارزمية Dijkstra على طوبولوجيا مختلفة عن الطوبولوجيا المستخدمة في الخوارزميات الأربعة السابقة كما هو موضح بالشكل 17. والتي تُعتبر من الخوارزميات الشهيرة في إيجاد المسارات المثلى بين العقد في الشبكات. وهذه الطوبولوجيا تناسب عمل هذه الخوارزمية.



الشكل 17 الطوبولوجيا المستخدمة لتنفيذ خوارزمية Dijkstra

5.3. تصميم خوارزمية Dijkstra:

المدخلات:

- قائمة العقد والوصلات (Nodes and Links List): تشمل جميع العقد (مثل المبدلات والموجهات) والوصلات بينها.
- العقدة المصدر (Source Node): العقدة التي يبدأ منها مسار نقل البيانات.
- العقدة الوجهة (Destination Node): العقدة التي سيتم توجيه البيانات إليها.

المخرجات:

- أفضل مسار (Optimal Path): سلسلة من العقد التي تمثل أفضل مسار بين العقدة المصدر والعقدة الوجهة بناءً على المعايير المختارة (عرض النطاق الترددي أو التأخير).
- تكلفة المسار (Path Cost): القيمة الكلية لتكلفة المسار المختار، سواء كانت تعتمد على عرض النطاق الترددي أو التأخير.

4. الفصل الرابع: الأدوات المستخدمة والتنفيذ

Implementation&Tools

نقدم في هذا الفصل بيئة العمل والأدوات المستخدمة التي ساعدت في التنجيزو كيفية تنفيذ الخوارزميات.

1.4. بيئة العمل وأدوات التنفيذ

1.1.4. Visual Studio Code (VS Code)

هو محرر نصوص مفتوح المصدر من تطوير مايكروسوفت. يُستخدم على نطاق واسع من قبل المطورين بسبب مرونته وقوته. يدعم العديد من لغات البرمجة مثل JavaScript، Python، Java، C++، وغيرها.

2.1.4. VirtualBox

هو برنامج مفتوح المصدر من تطوير Oracle يتيح إنشاء وتشغيل الآلات الافتراضية على أجهزة الكمبيوتر الشخصية. يتيح هذا البرنامج للمستخدمين تشغيل أنظمة تشغيل متعددة في وقت واحد على جهاز مادي واحد. يدعم VirtualBox أنظمة أساسية متنوعة مثل Windows، MacOS، Linux، وSolaris، ويقدم واجهة مستخدم بسيطة وسهلة لإنشاء وتكوين الآلات الافتراضية. يمكن للمستخدمين تخصيص موارد النظام مثل وحدة المعالجة المركزية، الذاكرة، والتخزين لكل آلة افتراضية وضبط الإعدادات الخاصة بها. يمكن استخدام VirtualBox لإنشاء بيئة افتراضية لاستخدام Mininet من اجل اختبارات المحاكاة.

3.1.4. الأداة Mininet

هي أداة برمجية مجانية لمحاكاة الشبكات الحاسوبية الحقيقية باستخدام البرمجيات نسخة مبسطة من الشبكة الحقيقية. تسمح للمستخدمين بإنشاء شبكات افتراضية، وتشغيل تطبيقات أو بروتوكولات الشبكة، وتطوير البرمجيات عالية السرعة. يقوم ببناء شبكة افتراضية تعمل على جهاز واحد باستخدام روابط شبكية افتراضية وخوادم افتراضية يمكن التحكم فيها برمجيًا عبر مكتبات إدارة الشبكات القياسية والتقاط الحزم. حيث يوفر أدوات للتجربة ومحاكاة بروتوكولات الشبكات الواقعية في بيئة اختبار افتراضية معزولة. يتميز Mininet بخفة وزنه، وقابليته للنقل، ومرونته. يتم التحكم فيه بواسطة سكريبت مكتوب بلغة Python ويمكنه تنفيذ عمليات محاكاة معقدة يصعب أو يستحيل القيام بها على العتاد المادي. يتمثل الاستخدام الرئيسي لـ Mininet في تمكين تطوير الشبكات المعرفة برمجيا (SDN) أو وظائف الشبكة الافتراضية(NFV).

تعتمد على لغة Python لإنشاء شبكات برمجية. يمكنه تشغيل الآلات الافتراضية التي يمكنها التواصل عبر مبدلات شبكية افتراضية متعددة، مما يسمح بمحاكاة شبكة لأغراض التطوير أو التدريب. يمكن للمستخدمين نشر العديد من الطوبولوجيا الشبكية المختلفة، بما في ذلك L2 وL3، على جهاز واحد. في Mininet، كل عقدة فردية في الشبكة هي نظام Linux كامل مع عتاد افتراضي. يمكن تغيير سلوك هذه الأجهزة الافتراضية أثناء التشغيل. من خلال التحكم في التفاعل بين الأجهزة الافتراضية المختلفة بشكل معزول، يمكن إنشاء بيئة بروتوكولات شبكية مخصصة للتجارب.

تم تصميمه لنمذجة سلوك نظام حقيقي، حيث يستخدم البرمجيات لمحاكاة الطبقة الفيزيائية للشبكة (MAC وPHY) وبروتوكولات طبقة الربط (Link layer) (ARP، IPv4/IPv6) وطبقة النقل(Transport layer)(SCTP).

يسهل Mininet على الباحثين إنشاء بيئات اختبار معزولة شبكيًا — سواء كانت محاكاة كاملة الحجم أو حتى محاكاة جزئية تتكون من أجزاء محددة مأخوذة من العالم الحقيقي — باستخدام Mininet كجهاز تحكم وLinux كمبدل/مضيف(switch/host). Mininet هو الأداة المفضلة لبناء بيئة بحث مفتوحة قابلة لإعادة الإنتاج، مما يمكّن الباحث من إعادة إنشاء أي تجربة بجهد قليل. Mininet هو بيئة افتراضية خفيفة الوزن لإنشاء ونشر الآلات الافتراضية. يمكن استخدامه لإعداد شبكات لمحاكاة، أو بيئات اختبار، أو مناهج تعليمية، أو محاكاة واسعة النطاق.

يمكن إنشاء شبكات افتراضية خاصة وتشغيل تطبيقات شبكية حقيقية بسرعات عالية. بحيث تأتي مع مكتبة شاملة من خدمات الشبكة والتطبيقات الجاهزة، بما في ذلك خادم DNS، NAT، خادم وعميل DHCP، وحدة تحكم ومبدل OpenFlow، مولد حركة المرور، برنامج لنقل البيانات، وأوامر ping وtraceroute.

Mininet تستخدم تقنية OpenFlow (OF) -based. Mininet هو محاكي شبكة عالي الأداء، حيث تعمل الآلة الافتراضية (VM) على جهاز مادي واحد كجهاز شبكة افتراضي. التطبيقات التي تُنفذ داخل الآلات الافتراضية يمكنها التواصل مع بعضها البعض و/أو مباشرة مع الشبكات الخارجية عبر هذا الجهاز الشبكي الافتراضي.

2.4. التنفيذ

فيما يلي تم تنفيذ الخوارزميات باستخدام VirtualBox-6.1 و Mininet-2.3.0 والالة الافتراضية Ubuntu-20.04.1 بذاكرة 5.71 GB و Python 3 و المتحكم POX 0.7.0 و المتحكم Ryu 4.34.

1.2.4. تنفيذ خوارزمية Random Selection

تم انشاء الشبكة الموضحة بالشكل 16 من الأداة التي يقدمها الـ **Mininet** وهي الأداة MiniEdit، هي واجهة رسومية (GUI) تقدمها منصة **Mininet**، وهي أداة قوية ومفتوحة المصدر تُستخدم لإنشاء شبكات افتراضية صغيرة من أجهزة الشبكة مثل المبدلات (switches) وأجهزة التوجيه (routers) والعقد (hosts) في بيئة افتراضية. تسهل على المستخدمين بناء وإدارة هذه الشبكات الافتراضية دون الحاجة إلى كتابة نصوص برمجية معقدة.

بحيث تم تهيئة كل من العقد h1,h2,h3,h4,h5 للعمل كمخدمات HTTP بسيطة وذلك عن طريق استخدام التعليمة python3 -m http.server 80 لكل منها.

وكان h6 بمثابة زبون يقوم بإرسال طلبات http ليتم تخديمها. ومن ثم تم تخصيص عناوين لكل منها بحيث تم تخصيص العنوانين للعقد h1,h2,h3,h4,h5,h6 على الترتيب

10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5,10.0.0.6 ومن ثم تم تشغيل رماز موازن الحمل المنجز بخوارزمية random selection وهو بمثابة تطبيق يعمل وفقه المتحكم بحيث يضع قواعد الدفق وفقا للطريقة التي تم بها تنجيز برنامج موازن الحمل وذلك عن طريق تنفيذ التعليمة التالية:

~/pox/pox.py log.level --DEBUG misc.ip_loadbalancer --ip=10.0.1.1 --servers=10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5

كما ذكرنا فان النموذج ip_loadbalancer.py الذي يحتوي منطق موازن الحمل الذي سوف نستخدمه يستخدم بشكل افتراضي خوارزمية اختيار المخدم بشكل عشوائي، كما هو موضح بالشكل 18

```
def _pick_server (self, key, inport):
  """
  Pick a server for a (hopefully) new connection
  """
  return random.choice(list(self.live_servers.keys()))
```

الشكل 18 اختيار المخدم بشكل عشوائي

2.2.4. تنفيذ خوارزمية Round-Robin

تم اتباع نفس الخطوات السابقة ولكن هنا تم التعديل بآلية عمل موازن الحمل كي يعمل بهذه الخوارزمية حيث تم تعديل آلية اختيار المخدم بحيث تم التعديل بجسم التابع التالي:

```
def _pick_server (self, key, inport):
      """
      Pick a server using round-robin
      """
      server = list(self.live_servers.keys())[self.current_server]
      self.current_server = (self.current_server + 1) % len(self.live_servers)
      return server
```

الشكل 19 اختيار المخدم باستخدام خوارزمية التوزيع الدائري

بحيث يتم توجيه الطلبات بشكل دوار حيث يتم توجيه طلب معين الى مخدم والطلب اللاحق الى المخدم التالي بحيث عندما يتم الوصول للمخدم الأخير يتم توجيه الطلب التالي الى المخدم الأول وفق ترتيب للمخدمات موضوع مسبقا ويمكن تنفيذ هذه الخوارزمية من خلال تشغيل برنامج موازن الحمل على المحددات المرادة وفق التعليمة التالية:

~/pox/pox.py log.level --DEBUG misc.roundCode --ip=10.0.1.1 --servers=10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5

بعد تشغيل موازن الحمل في طبقة التحكم كما ذكرنا سابقا وذلك بتخصيص عمل كل من **h1**,**h2**,**h3**,**h4**,**h5** كمخدمات **http** بسيطة، من واجهة تنفيذ التعليمات لـ**h6** نقوم بتنفيذ الرمازين السابقين (حساب زمن الاستجابة الوسطي و معدل النقل).

3.2.4. تنفيذ خوارزمية Weighted Round-Robin

```
def _pick_server(self, key, inport):
    """
    Pick a server using weighted round-robin
    """
    if not self.live_servers:
        return None  # Handle case when there are no live servers

    # Select server index based on expanded weights
    server_index = self.server_weights[self.current_server_index]
    server = list(self.live_servers.keys())[server_index]

    # Move to the next server for the next connection
    self.current_server_index = (self.current_server_index + 1) % len(self.server_weights)

    return server
```

الشكل 20 اختيار المخدم بخوارزمية التوزيع الدائري الموزون

كما فعلنا في تنفيذ خوارزمية Round-Robin، وعلى نفس الشبكة وبنفس الظروف، قمنا بالتعديل بجسم التابع الذي يقوم باختيار المخدم كما هو موضح بالشكل 20، وفق ما يلي:

يقوم التابع أعلاه أولا بالتحقق من حالة المخدم إذا كان متاح بحيث إذا لم يكن متاح ببساطة لا يقوم بإرجاع هذا المخدم. اما إذا كان متاح يقوم بإعادته بناء على تابع معرف سابقا هو عبارة تابع يعيد لائحة من المخدمات بناء على الاوزان بحيث يتم انشاء هذه اللائحة كما هو موضح بالشكل 21:

```
def _expand_weights(self, weights):
    """
    Expand the weights into a list of server indices based on their weights
    """
    expanded_weights = []
    for index, weight in enumerate(weights):
        expanded_weights.extend([index] * weight)
    return expanded_weights
```

الشكل 21 إعادة المخدمات على شكل لائحة بناء على الأوزان

حيث هذه اللائحة تحوي على المخدمات بناء على الاوزان حيث يتم مضاعفة المخدم وفق الوزن أي انه يتم تكرار المخدم وفق وزنه (هنا يكون الوزن عبارة عن عدد طبيعي) حيث تكون المخدمات المتكررة متتالية.

تم تشغيل موازن الحمل بناء على اوزان قمنا باختيارها حيث تم اخذ الأوزان التالية 2,1,1,1,1 للمخدمات 5,4,3,2,1 على الترتيب وهذه الأوزان هي على سبيل التجريب وأنها توضع وفق موارد المخدمات أو عندما يتم مراقبة أدائها. وتم تشغيل موازن الحمل من خلال تنفيذ التعليمة التالية:

~/pox/pox.py log.level --DEBUG misc.WroundCode --ip=10.0.1.1 --servers=10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5 --weights=2,1,1,1,1

بعد تنفيذ التعليمة السابقة وتشغيل موازن الحمل بالخوارزمية السابقة تم أيضا تنفيذ الرمازين المسؤولان عن حساب زمن الاستجابة الوسطي ومعدل النقل.

4.2.4. تنفيذ خوارزمية Least Connection

تم تنفيذ هذه الخوارزمية أيضا على نفس الشبكة السابقة وتم تنفيذ هذه الخوارزمية وذلك من خلال التعديل على التابع الذي يقوم باختيار المخدم كما هو موضح بالشكل 22:

```
def _pick_server (self, key, inport):
    """
    Pick a server for a (hopefully) new connection
    """
    if not self.live_servers:
        self.log.warn("No servers!")
        return None

    # Choose the server with the least number of connections
    min_connections = float('inf')
    selected_server = None

    for server in self.live_servers:
        count = self.connection_counts.get(server, 0)
        if count < min_connections:
            min_connections = count
            selected_server = server

    return selected_server
```

الشكل 22 اختيار المخدم باستخدام خوارزمية أقل عدد من الاتصالات

يتم بداية التحقق من المخدمات المتاحة (التي تعمل) ومن ثم يتم تحديد القيمة min_connection بحيث تكون قيمة كبيرة جدا لضمان ان أي عدد من الاتصالات سيعتبر اقل من هذه القيمة بالبداية.

يتم التحقق من عدد الاتصالات الحالية المرتبطة بالمخدم الحالي من خلال self.connection_counts.get(server,0) بحيث يخزن عدد الاتصالات لكل خادم ويتم مقارنة عدد الاتصالات count بقيمة min_connections إذا كان عدد الاتصالات الحالي اقل من min_connection يتم تحديث الأخيرة الى هذا العدد ويتم تعيين selected_server ليكون هذا المخدم ومن ثم يتم إعادة هذا المخدم الذي يملك اقل عدد من الاتصالات.

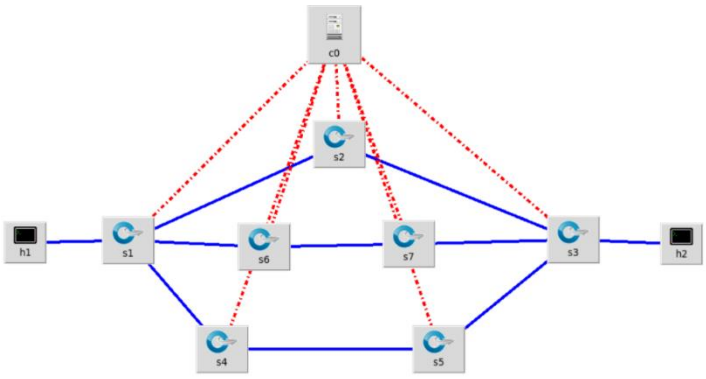
تم تشغيل موازن الحمل باستخدام هذه الخوارزمية من خلال تنفيذ التعليمة:

```
~/pox/pox.py log.level --DEBUG misc.LeastConn --ip=10.0.1.1 --servers=10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5
```

بعد تنفيذ التعليمة السابقة وتشغيل موازن الحمل بالخوارزمية السابقة تم أيضا تنفيذ الرمازين المسؤولان عن حساب زمن الاستجابة الوسطي ومعدل النقل.

5.2.4. تنفيذ خوارزمية Dijkstra بناء على عرض المجال الترددي للوصلات

تم استخدام هذه الخوارزمية على الشبكة الموضحة التالية:



الشكل 23 الطوبولوجيا المستخدمة لاختبار خوارزمية Dijstra

تم انشاء هذه الشبكة بواسطة الأداة MiniEdit كما فعلنا لإنشاء الشبكة السابقة. ولكن هنا تم استخدام المتحكم Ryu وذلك لانه يدعم بروتوكولات لا يدعمها المتحكم ك بروتوكول STP الذي يقوم بمنع الحلقات (Loops). في الشبكات التي تحتوي على مسارات متعددة بين أي جهازين، يمكن أن تتسبب الحلقات في حدوث "عواصف بث" (broadcast storms) وتكرار البيانات داخل الشبكة بشكل مستمر، مما يؤدي إلى تعطيل الشبكة بالكامل. يعمل STP على منع حدوث هذه الحلقات من خلال تحديد مسار واحد فقط للبيانات، وإيقاف أو تعطيل المسارات البديلة مؤقتًا. وبما اننا نقوم بموازنة الحمل عن طريق اختيار أفضل طريق بناء على قيمة الـ Band-Width فيهمنا عدم وجود حلقات في الشبكة. بداية تم استخدام خوارزمية DFS كما في الشكل 24، بحيث تقوم الخوارزمية باستكشاف الرسم البياني بالاتجاه نحو العمق، وتعتمد على

```
def find_paths_and_costs(self, src, dst):
    '''
    Implementation of Depth-First Search Algorithm (DFS)
    Output of this function returns an list on class Paths objects
    '''
    if src == dst:
        return [Paths(src,0)]
    stack = [(src, [src])]
    possible_paths = list()
    while stack:
        (edge, path) = stack.pop()
        for vertex in set(self.neigh[edge]) - set(path):
            if vertex == dst:
                path_to_dst = path + [vertex]
                cost_of_path = self.find_path_cost(path_to_dst)
                possible_paths.append(Paths(path_to_dst, cost_of_path))
            else:
                stack.append((vertex, path + [vertex]))
    return possible_paths
```

الشكل 24آلية إيجاد المسارات والكلف باستخدام DFS

المكدس للاحتفاظ بالقمة التالية التي ستبدأ منها عملية البحث عند الوصول إلى طريق مسدود في أي مرحلة من التكرار. والتابع التالي بهذه العملية:

اثناء عملية استكشاف الوصلات يتم حساب كلفة هذا الطريق عن طريق جمع كافة الكلف لكل الوصلات من عقدة الى عقدة أخرى ويتم ذلك عن طريق التابع التالي:

```
def find_path_cost(self, path):
    ''' arg path is a list with all nodes in our route '''
    path_cost = []
    i = 0
    while(i < len(path) - 1):
        port1 = self.neigh[path[i]][path[i + 1]]
        bandwidth_between_two_nodes = self.get_bandwidth(path, port1, i)
        path_cost.append(bandwidth_between_two_nodes)
        i += 1
    return sum(path_cost)
```

الشكل 25 آلية حساب كلفة مسار بناء على قيمة عرض المجال الترددي

تم حساب الكلفة بناء على قيمة عرض المجال الترددي لهذه الوصلة، وذلك باستخدام التابع التالي:

```
def get_bandwidth(self, path, port, index):
  return self.bw[path[index]][port]
```

الشكل 26 التابع الذي يعيد قيمة عرض المجال الترددي لمسار

بداية لحساب عرض المجال الترددي يقوم التابع الموضح في الشكل 27 بما يلي :

```
def run_check(self, ofp_parser, dp):
    threading.Timer(1.0, self.run_check, args=(ofp_parser, dp)).start()

    req = ofp_parser.OFPPortStatsRequest(dp)
    dp.send_msg(req)
```

الشكل 27مراقبة إحصاءات المنافذ

يستدعي طلب الإحصاءات OFPPortStatsRequest و يستقبل الرد في التابع _port_stats_reply_handler

حيث يتم استخدام threading.Timer لإنشاء مؤقت زمني هذا المؤقت سيستدعي التابع run_check بعد 1 ثانية حيث start(). تبدأ تشغيل المؤقت مما يؤدي الى استدعاء التابع run_check بشكل دوري كل ثانية. ومن ثم يتم انشاء طلب

```
def _port_stats_reply_handler(self, ev):
    '''Reply to the OFPPortStatsRequest, visible beneath'''
    switch_dpid = ev.msg.datapath.id
    for p in ev.msg.body:
        self.bw[switch_dpid][p.port_no] = (p.tx_bytes - self.prev_bytes[switch_dpid][p.port_no])*8.0/1000000
        self.prev_bytes[switch_dpid][p.port_no] = p.tx_bytes
```

الشكل 28 آلية حساب عرض المجال الترددي

لاحصاءات المنفذ. وهكذا فان هذا التابع هو جزء من عملية مستمرة لمراقبة إحصاءات المنافذ في المبدل ثم تستخدم هذه المعلومات لاحقا في حساب عرض المجال الترددي في التابع _port_stats_reply_handler الموضح بالشكل 28.

حيث يتم حساب عرض المجال الترددي وذلك بحساب الفرق بين البتات المرسلة من المنفذ والبتات التي تم ارسالها في المرة السابقة ومن ثم تحويل هذه القيمة الى ميغابايت ومنه أخيرا نحصل على قيمة عرض المجال الترددي بواحدة الـ Mbps.

ومن ثم يتم اختيار افضل طريق وذلك كما في الشكل 29:

```
def find_n_optimal_paths(self, paths, number_of_optimal_paths = MAX_PATHS):
    '''arg paths is an list containing lists of possible paths'''
    costs = [path.cost for path in paths]
    optimal_paths_indexes = list(map(costs.index, heapq.nsmallest(number_of_optimal_paths,costs)))
    optimal_paths = [paths[op_index] for op_index in optimal_paths_indexes]
    return optimal_paths
```

الشكل 29 اختيار أفضل طريق

حيث يقوم التابع باختيار افضل طريق من ناحية عرض المجال الترددي.

تم تشغيل موازن الحمل باستخدام هذه الخوارزمية من خلال تنفيذ التعليمة:

sudo ryu-manager --observe-links multipath.py

6.2.4. تنفيذ خوارزمية Dijkstra بناء على التأخير للوصلات

تم تنفيذ هذه الخوارزمية بنفس ظروف الخوارزمية السابقة وعلى نفس الشبكة، حيث تم تعديل الخوارزمية السابقة ليتم حساب كلفة الوصلة بالنسبة للتأخير وليس عرض الحزمة الترددي. حيث تم التعديل في جسم التابع التالي:

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
    switch_dpid = ev.msg.datapath.id

    # Calculate the current time (reply time)
    reply_time = time.time()

    # Retrieve the request time for this switch
    request_time = self.request_timestamps.get(switch_dpid, None)

    if request_time:
        # Calculate latency
        latency = reply_time - request_time

        for p in ev.msg.body:
            # Store the latency for each port
            self.latency[switch_dpid][p.port_no] = latency
    else:
        # Handle the case where there is no request timestamp (unexpected)
        self.logger.error(f"No request timestamp found for switch {switch_dpid}")
```

الشكل 30 آلية حساب التأخير

كما تم تعريف التابع run_check الذي يقوم بمراقبة إحصاءات المنافذ في المبدل ثم تستخدم هذه المعلومات لاحقا في حساب التأخير في التابع _port_stats_reply_handler الموضح أعلاه حيث يتم حساب التأخير عن طريق حساب الوقت الذي يتم به ارسال الطلب من المنفذ ومن ثم حساب وقت الرد ومنه يتم حساب التاخير من منفذ معين والمنفذ المجاور (أي التاخير الحاصل على الوصلة بين مبدلين) وتخزين هذه القيم لهذه الوصلات ومنه يصبح لدينا كلفة هذه الوصلات بالنسبة لكل الوصلات في كل لحظة وبعدها يتم حساب كلفة كافة الطرق عن طريق نفس التابع المعرف في الخوارزمية السابقة find_path_costs مع التعديل البسيط التالي:

```
def find_path_cost(self, path):
    ''' arg path is a list with all nodes in our route '''
    path_cost = []
    i = 0
    while(i < len(path) - 1):
        port1 = self.neigh[path[i]][path[i + 1]]
        latency_between_two_nodes = self.get_latency(path, port1, i)
        path_cost.append(latency_between_two_nodes)
        i += 1
    return sum(path_cost)
```

الشكل 31 آلية حساب كلفة مسار بناء على قيمة التأخير

تم تشغيل موازن الحمل باستخدام هذه الخوارزمية من خلال تنفيذ التعليمة:

sudo ryu-manager --observe-links multipathWithLatency.py

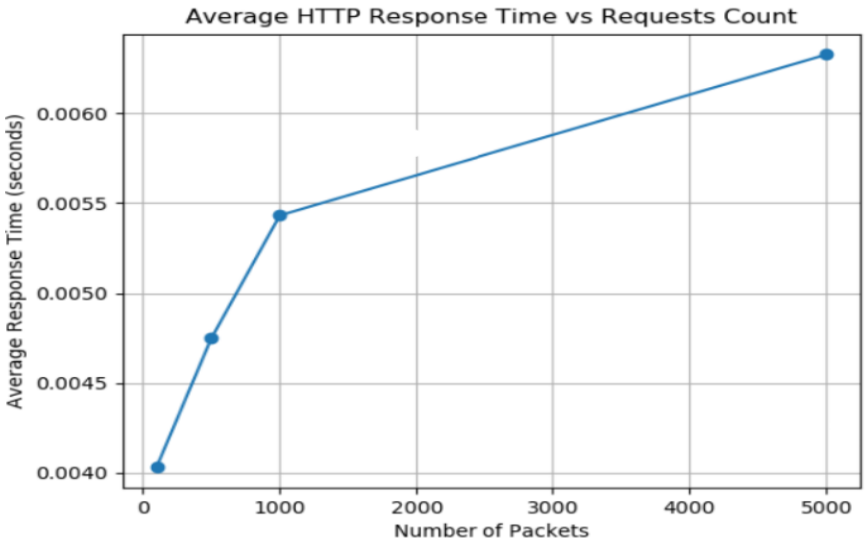
5. الفصل الخامس: الاختبارات والنتائج

Testing&Results

سنقدم في هذا الفصل الاختبارات ومناقشة النتائج.

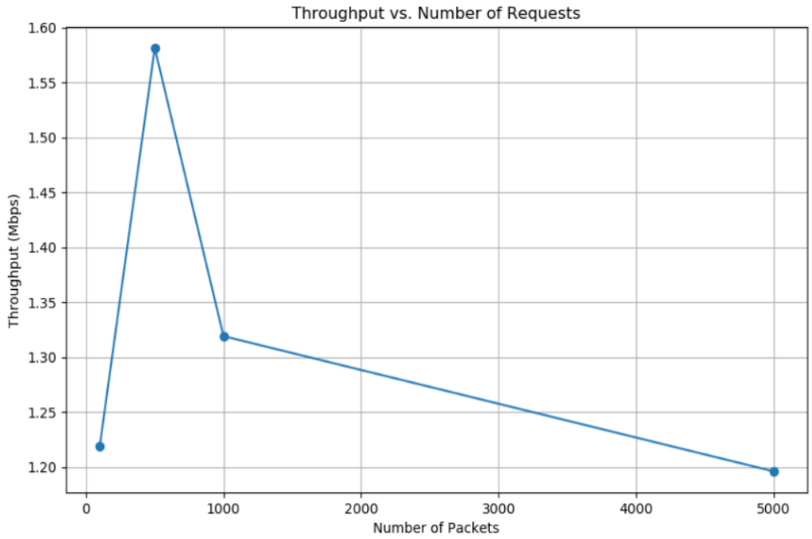
1.5. اختبار خوارزمية Random Selection

من اجل الاختبار تم تنجيز رماز بلغة Python يقوم بتوليد اعداد مختلفة من الطلبات حيث يرسل بداية طلبات لها نفس الحجم وذلك من اجل التبسيط حيث يتم ارسال 100 طلب وذلك من اجل تهيئة الشبكة (يجب ان يكون عدد صغير نسبيا) ومن ثم 500 طلب (حمل قليل) ثم 1000 طلب (حمل وسطي) و5000 طلب (حمل عالي نسبيا) وهذه الاعداد هي بهدف التجريب. يتم تنفيذ هذا الرماز من واجهة تنفيذ الأوامر الخاصة بـ h6 حيث يقوم بحساب زمن الاستجابة الوسطي للطلبات التي تتم معالجتها من قبل المخدمات وذلك من خلال تعيين الوقت الذي تم ارسال فيه الطلب والوقت الذي تم الرد من قبل المخدم (وقت الاستجابة) ومن خلال حساب الفرق بين الوقتين يكون قد أصبح لدينا وقت الاستجابة لكل طلب وبالتالي يمكن حساب زمن الاستجابة الوسطي لكل الطلبات المرسلة [100,500,1000,5000] وكانت النتائج على النحو التالي:



الشكل 32 زمن الاستجابة الوسطي لخوارزمية الاختيار العشوائي

ومن ثم تم تنجيز رماز اخر بلغة Python يقوم بحساب معدل النقل Throughput حيث يتم حساب الوقت الذي تم ارسال الطلبات فيه ومن ثم حساب حجم البيانات المستلمة لكل طلب (طول محتوى الاستجابة بالبايت) ومن ثم حساب حجم البيانات المستلمة من جميع الطلبات وأخيرا يتم حساب وقت انتهاء هذه المرحلة، حيث أصبح لدينا الزمن الكلي الذي استغرقته العملية ولحساب معدل النقل نقوم بحساب كمية البيانات المنقولة ولحساب معدل النقل يتم تقسيم حجم البيانات على الزمن الكلي في الثانية الواحدة بوحدة الميغابت. يتم تطبيق ذلك من اجل اعداد مختلفة من الطلبات [100,500,1000,5000] ونحصل على النتيجة التالية:

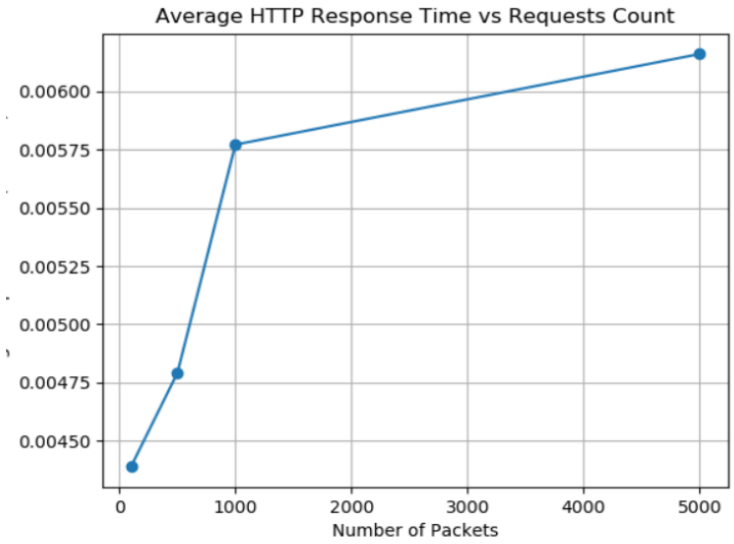


الشكل 33 معدل النقل لخوارزمية الاختيار العشوائي

2.5. اختبار خوارزمية Round-Robin

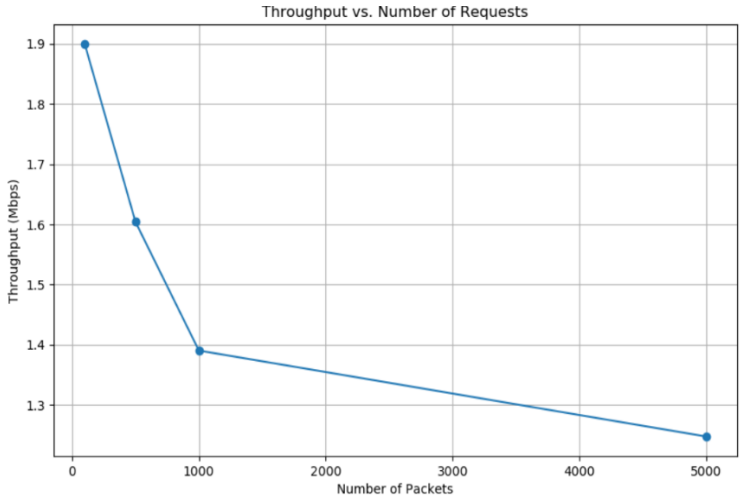
بعد تشغيل موازن الحمل في طبقة التحكم كما ذكرنا سابقا وذلك بتخصيص عمل كل من **h1**,**h2**,**h3**,**h4**,**h5** كمخدمات **http** بسيطة، من واجهة تنفيذ التعليمات لـ**h6** نقوم بتنفيذ الرمازين السابقين (حساب زمن الاستجابة الوسطي و معدل النقل) وكانت النتائج على الشكل التالي:

حساب زمن الاستجابة الوسطي كما في الشكل:



الشكل 34 زمن الاستجابة الوسطي لخوارزمية التوزيع الدائري

وكانت نتائج معدل النقل كما في الشكل:

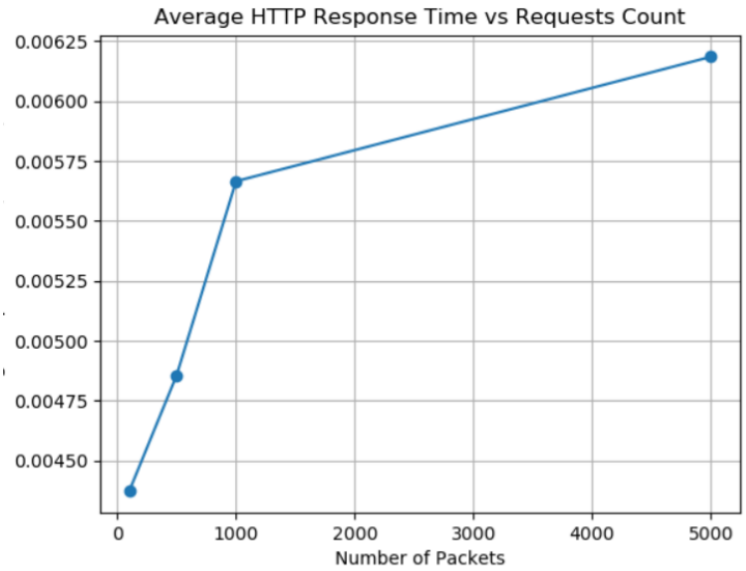


الشكل 35 معدل النقل لخوارزمية التوزيع الدائري

3.5. اختبار خوارزمية Weighted Round-Robin

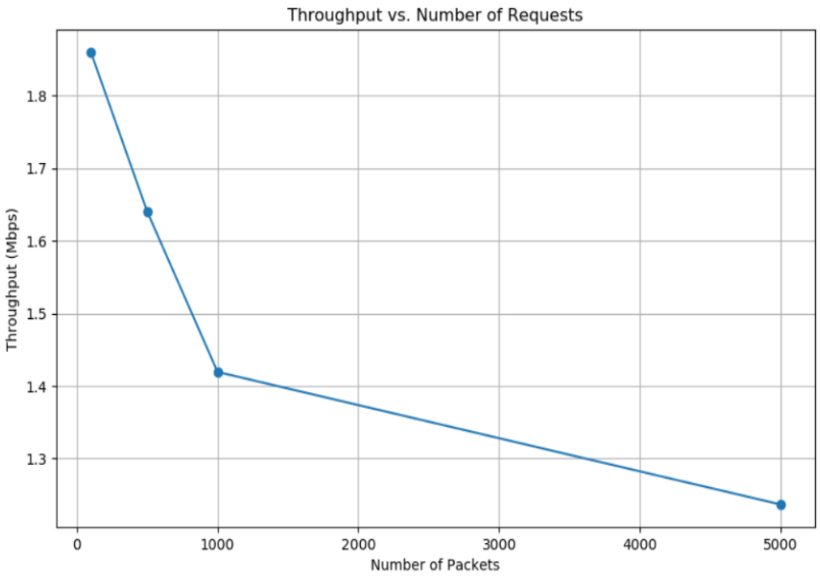
بعد تشغيل موازن الحمل في طبقة التحكم كما ذكرنا سابقا وذلك بتخصيص عمل كل من **h1**,**h2**,**h3**,**h4**,**h5** كمخدمات **http** بسيطة، من واجهة تنفيذ التعليمات لـ**h6** نقوم بتنفيذ الرمازين السابقين (حساب زمن الاستجابة الوسطي و معدل النقل) كانت النتائج على الشكل التالي:

حساب زمن الاستجابة الوسطي كما في الشكل 36:



الشكل 36 زمن الاستجابة الوسطي لخوارزمية التوزيع الدائري الموزون

كانت نتائج معدل النقل كما في الشكل:

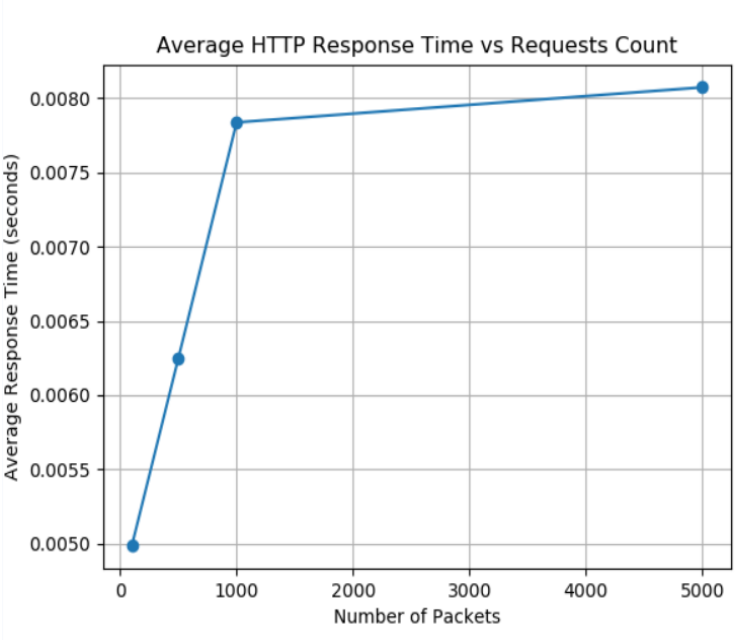


الشكل 37 معدل النقل لخوارزمية التوزيع الدائري الموزون

4.5. اختبار خوارزمية Least Connection

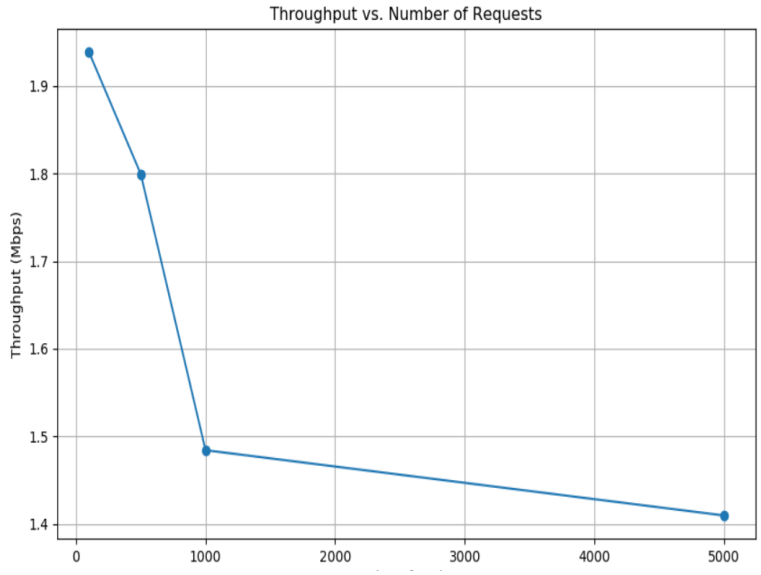
بعد تشغيل موازن الحمل في طبقة التحكم كما ذكرنا سابقا وذلك بتخصيص عمل كل من **h1**,**h2**,**h3**,**h4**,**h5** كمخدمات **http** بسيطة، من واجهة تنفيذ التعليمات لـ**h6** نقوم بتنفيذ الرمازين السابقين (حساب زمن الاستجابة الوسطي و معدل النقل) وكانت النتائج على الشكل التالي:

حساب زمن الاستجابة الوسطي كما في الشكل:



الشكل 38زمن الاستجابة الوسطي لخوارزمية أقل عدد من الاتصالات

كانت نتائج معدل النقل كما في الشكل:

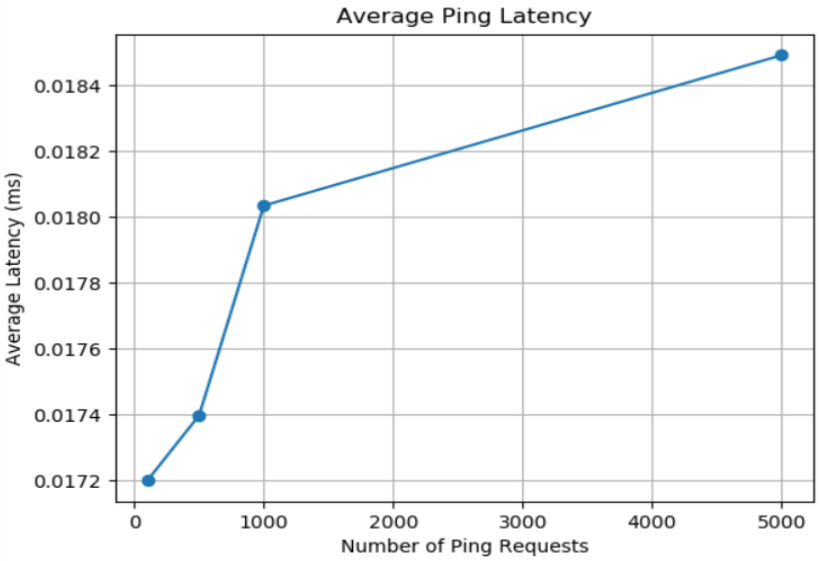


الشكل 39معدل النقل لخوارزمية أقل عدد من الاتصالات

5.5. اختبار خوارزمية Dijkstra بناء على عرض المجال الترددي للوصلات

ومن اجل الاختبار تم تنجيز رماز بلغة Python (مرفق بالملحق) يقوم بتوليد اعداد مختلفة من الطلبات حيث يرسل بداية طلبات لها نفس الحجم وذلك من اجل التبسيط حيث يتم ارسال 100 طلب وذلك من اجل تهيئة الشبكة (يجب ان يكون

عدد صغير نسبيا) ومن ثم 500 طلب (حمل قليل) ثم 1000 طلب (حمل وسطي) و5000 طلب (حمل عالي نسبيا) وهذه الاعداد هي بهدف التجريب. حيث يتم ذلك عن استخدام تعليمة ping حيث تقوم بارسال طلب من h1 و h2 ومن ثم إعادته من h2 الى h1 ومنه نحسب زمن RTT وبعدها نحسب زمن الاستجابة الوسطي لهذه الطلبات من اجل كل حالة. يظهر الشكل زمن الاستجابة الوسطي:

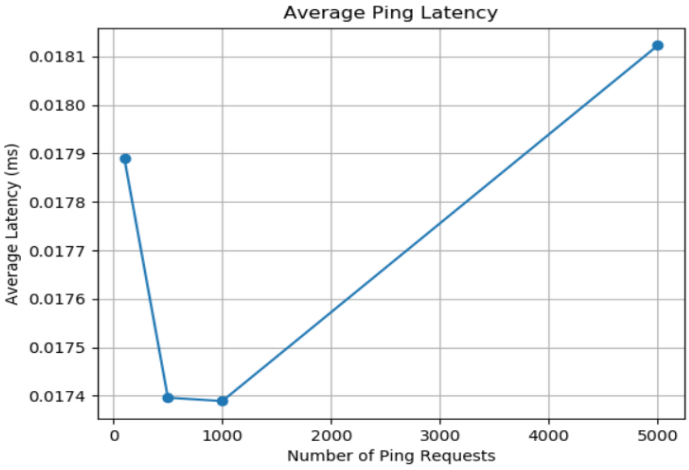


الشكل 40 زمن الاستجابة الوسطي لخوارزمية Dijkstra بناء على كلفة عرض المجال الترددي

6.5. اختبار خوارزمية Dijkstra بناء على التأخير للوصلات

ومن اجل الاختبار تم استخدام الرماز السابق لحساب وسطي زمن الاستجابة.

يظهر الشكل زمن الاستجابة الوسطي:



الشكل 41 زمن الاستجابة الوسطي لخوارزمية Dijkstra بناء على كلفة التأخير

7.5. مناقشة النتائج

بعد إجراء الاختبارات عدة مرات لزيادة دقة النتائج، أظهرت الخوارزميات المستخدمة على مستوى الخوادم نتائج متقاربة من حيث زمن الاستجابة الوسطي. يعود سبب هذا التقارب إلى تماثل مواصفات المخدمات، حيث كان الاختلاف بسيطًا. على سبيل المثال، عند استخدام خوارزمية Weighted Round-Robin مع تخصيص أوزان مختلفة للمخدمات، لم يظهر فرق واضح في الأداء بسبب تماثل مواصفات هذه المخدمات. كما أظهرت النتائج تقاربًا من ناحية معدل النقل بين الخوارزميات المختلفة، مع ملاحظة أن خوارزمية Least Connection قدمت أداءً أفضل قليلاً فيما يتعلق بمعدل النقل. يمكن تحقيق نتائج أكثر وضوحًا عند استخدام مخدمات ذات مواصفات متنوعة، إلا أن الاختبارات التي أجريناها اعتمدت على مخدمات HTTP بسيطة تعمل كآلات افتراضية، حيث تتشارك في نفس الموارد التي تم تخصيصها للبيئة الافتراضية Mininet.

أما بالنسبة للخوارزميات المستخدمة على مستوى الوصلات، فقد كانت النتائج متوقعة. عند حساب أوزان الوصلات بناءً على زمن التأخير واختيار المسار الذي يتمتع بأقل زمن تأخير، أظهر الحساب الوسطي لزمن الاستجابة تحسنًا مقارنةً بالحالة التي تم فيها حساب أوزان الوصلات بناءً على عرض النطاق الترددي. ومع ذلك، يبقى اختيار الخوارزمية الأنسب مرتبطًا بطوبولوجيا الشبكة المستخدمة والسيناريو المطبق عليها، حيث يتم تحديد الخوارزمية بناءً على متطلبات محددة مثل زمن التأخير أو معدل النقل وغيرها من العوامل.

6. الفصل السادس

الخاتمة والافاق المستقبلية

سنقدم في هذا الفصل الخاتمة ومناقشة النتائج.

1.6. الخاتمة

في ختام هذا المشروع، تمكنا من تسليط الضوء على أهمية موازنة الحمل في الشبكات المعرفة برمجياً (SDN) ودورها الحيوي في تحسين أداء الشبكة واستقرارها، حيث تكمن المشكلة الأساسية في كيفية توزيع الحمل بشكل فعّال على الخوادم والوصلات داخل شبكة SDN، وذلك بهدف تحسين استجابة الشبكة وتقليل زمن التأخير مع زيادة معدل النقل. حيث تم اتباع نهج شامل لحل هذه المشكلة من خلال تطوير وتنفيذ مجموعة من الخوارزميات لموازنة الحمل، بما في ذلك خوارزميات اختيار الخادم مثل Round-Robin وLeast Connection، وكذلك خوارزميات اختيار المسار بناءً على عرض النطاق الترددي وزمن التأخير. تم اختبار هذه الخوارزميات في بيئة محاكاة باستخدام Mininet، حيث تم قياس الأداء بناءً على معايير مختلفة مثل زمن الاستجابة ومعدل النقل.

2.6. الافاق المستقبلية

لتحقيق تحسينات أكبر في أداء الشبكة، من الضروري النظر في توسيع نطاق الاختبارات ليشمل بيئات أكثر تعقيدًا مع تنوع أكبر في مواصفات الخوادم والوصلات. كما يمكن الاستفادة من تقنيات الذكاء الاصطناعي والتعلم الآلي لتحليل البيانات المستمرة من الشبكة واقتراح تعديلات فورية على توزيع الحمل. إضافةً إلى ذلك، يمكن العمل على تطوير خوارزميات جديدة تأخذ في الاعتبار استهلاك الطاقة وموثوقية الشبكة بشكل أكبر.

في المستقبل، يمكن توسيع نطاق تطبيق SDN ليشمل شبكات أكثر تعقيدًا مثل شبكات إنترنت الأشياء (IoT) والشبكات السحابية، مما يتطلب حلولًا أكثر تطورًا وقابلية للتكيف مع البيئات المتغيرة باستمرار. بذلك، نكون قد وضعنا الأسس لمزيد من البحث والتطوير في هذا المجال الذي لا يزال يحمل الكثير من الإمكانات لتحسين الأداء الشبكي وتقليل التكلفة التشغيلية.

المراجع

[**1**] R. Presuhn, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)," RFC 3416 (INTERNET STANDARD), Internet Engineering Task Force, Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3416.txt

[**2**] K. Greene, "MIT Tech Review 10 Breakthrough Technologies: Software-defined Networking," http://www2.technologyreview.com/ article/412194/tr10-software-defined-networking/, 2009.

[**3**] P. Newman, G. Minshall, and T. L. Lyon, "Ip switching—atm under ip," IEEE/ACM Trans. Netw., vol. 6, no. 2, pp. 117–129, Apr. 1998

[**4**] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008.

[**5**] H. Alkhatib, P. Faraboschi, E. Frachtenberg, H. Kasahara, D. Lange, P. Laplante, A. Merchant, D. Milojicic, and K. Schwan, "IEEE CS 2022 report (draft)," IEEE Computer Society, Tech. Rep., February 2014.

[**6**] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," Commun. ACM, vol. 57, no. 10, pp. 86–95, Sep. 2014.

[**7**] Chahlaoui, Farah, and Hamza Dahmouni. 2020. "A Taxonomy of Load Balancing Mechanisms in Centralized and Distributed SDN Architectures." *SN Computer Science* 1, no. 268. https://doi.org/10.1007/s42979-020-00288-8

[**8**] Hanamakkanavar AS, Handur VS. Load balancing in distributed systems: a survey. Int J Emerg Technol Comput Sci Electron (IJETCSE). 2

[**9**] Vashistha J, Jayswal AK. Comparative study of load balancing algorithms. IOSR J Eng. 2013;3:45–50.

[**10**] Sidhu AK, Kinger S, Sahib F. Analysis of load balancing techniques in cloud computing. Int J Comput Technol 2013;4.

[**11**] MacVittie D. Intro to load balancing for developers the algorithms. DevCentral. [Online] 31 March 2009. https://devcentral .f5.com/articles/intro-to-load-balancing-for-developers-ndash –the-algorithms.

[**12**] Bala K, Vashist S, Singh R, Singh G. A review of the load balancing techniques at cloud server. Int J Adv Comput Sci Commun Eng. 2014;2:2347–6788.

[**13**] Li Y, Pan D. OpenFlow based load balancing for fat-tree networks with multipath support. In: IEEE international conference on communication (ICC). 2013.

[**14**] Moore J, Nettles S. Towards practical programmable packets. In: Proceeding of IEEE computer and communication societies (INFOCOM) Anchorage. 2001.

[**15**] POX. https://www.noxrepo.org/pox/about-pox/.

[**16**] Zhuo, L., Zhou, X., Gao, J., & Qin, Y. (2021). *SDN controller load balancing based on reinforcement learning*. arXiv preprint arXiv:2103.06579.

[**17**] Aguilar, L. C. F. P., and D. M. Batista. Effectiveness of Implementing Load Balancing via SDN. University of Sao Paulo (USP), 2019

[**18**] Doe, John. "Enhancing Quality of Service in SDN based on Multi-path Routing Optimization with DFS." IEEE Access, 2023.

[**19**] Adams, K., & Agesen, O. (Year). A Comparison of Software and Hardware Techniques for x86 Virtualization.

[**20**] "Advantages and Disadvantages of Virtualization" - Vittana. Available at: https://vittana.org/14-advantages-and-disadvantages-of-virtualization

[**21**] "What is a Virtual Machine?" - Citrix. Available at: https://www.citrix.com/solutions/vdi-and-daas/what-is-a-virtual-machine.html

[**22**] Tuncer D, Charalambides M, Clayman S, Pavlou G. Adaptive resource management and control in software defned networks. In: IEEE transactions on network and service management, vol 12, no 1. 2015