مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (41) العدد (1) 2019
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (41) No. (1) 2019

A Study of OpenFlow Protocol and POX Controller in Software Defined Networks(SDN) Using Mininet

Dr. Ahmad Saker Ahmad [*]
Afraa Mohammad[**]

(Received 29 / 11 / 2018. Accepted 24 / 2 / 2019)

□ ABSTRACT □

In spite of the tremendous development in the field of information technology and the emergence of different mechanisms to improve the overall performance of the network, it did not study the development of the network infrastructure, prompting researchers to find a technology to develop the network architecture and the result was emergence of Software Defined Networks (SDN) which is a quantum leap in networking for its advantages in improving network performance and scalability.

SDN is the next generation of infrastructure in the world of networks, because it can accomplish what can't be done by traditional networks which contain a switch or a router responsible for making decisions and implementing these decisions, while SDN provided a split between the decision making (Control Plane) and (Data Plane), and this division is the great development provided by Software Defined Networks.

Communication between devices in the Data Plane and controller which is the most important element and represents the brain of network is driven through OpenFlow protocol which is the most popular protocol in SDN

This research provides a study of OpenFlow protocol, the mechanism of communication between switch and the controller and what are the exchanged messages. In addition to study of POX controller that is one of important and famous controllers in SDN and how to achieve the intelligence of the network and the possibility of programming to implement many applications in order to realize the concept of SDN, then we measured throughput and latency of this controller, and thus we have a full knowledge of the components of SDN technology which helps us to propose and implement many different applications.

Key Words: Software Defined Networks, OpenFlow Protocol, Controller, Throughput, Latency.

[*] **Professor, Department of Computer Networks & System, Faculty of Information Technology, University of Tishreen, Lattakia, Syria.**
[**] **PhD Student, Department of Computer Networks & System, Faculty of Information Technology, University of Tishreen, Lattakia, Syria.**

مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (41) العدد (1) 2019
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (41) No. (1) 2019

دراسة البروتوكول OpenFlow والمتحكم POX في الشبكات المعرفة بالبرمجة SDN باستخدام Mininet

د. أحمد صقر أحمد*
عفراء محمد**

(تاريخ الإيداع 29 / 11 / 2018. قُبِل للنشر في 24/ 2 / 2019)

□ ملخّص □

على الرغم من التطور الهائل في مجال تقانة المعلومات، وظهور آليات مختلفة لتحسين الأداء العام للشبكة، إلا أنها لم تتناول تطوير البنية التحتية للشبكة، مما دفع الباحثين إلى إيجاد تقنية بهدف تطوير معمارية الشبكة ونتج عن ذلك ظهور الشبكات المعرفة بالبرمجة ((Software Defined Networks (SDN) التي تعتبر نقلة نوعية في مجال الشبكات، لما قدمته من مزايا في تحسين الأداء وإمكانية التطوير اللامحدود.

تعتبر SDN الجيل القادم للبنية التحتية في عالم الشبكات، لكونها تستطيع إنجاز ما لا يمكن أن تقدمه الشبكات التقليدية التي تحتوي على مبدل أو موجه مسؤول عن اتخاذ قرارات التوجيه وتنفيذ هذه القرارات، بينما قدمت SDN فصلاً بين اتخاذ قرارات التوجيه (Control Plane) وتنفيذ هذه القرارات (Data Plane) ، ويعتبر هذا الفصل هو التطور الكبير الذي قدمته الشبكات المعرفة بالبرمجة.

يتم تنظيم التواصل بين الأجهزة الموجودة في الـ (Data Plane) والمتحكم (Controller)، الذي يعتبر العنصر الأكثر أهميةً، ويمثل عقل الشبكة من خلال بروتوكول التدفق المفتوح (OpenFlow Protocol) الأكثر شهرةً في تقنية SDN.

يقدم هذا البحث دراسة البروتوكول OpenFlow والتعرف على آلية الاتصال بين المبدل والمتحكم وماهي الرسائل المتبادلة بينهما، إضافةً إلى دراسة المتحكم POX الذي يعد من المتحكمات الهامة والشهيرة في SDN وكيف يحقق ذكاء الشبكة وإمكانية برمجته لينفذ أكثر من تطبيق مما يحقق مفهوم SDN ، ومن ثم قياس إنتاجية وتأخير هذا المتحكم، وبالتالي تتشكل لدينا معرفة كاملة حول مكونات تقنية SDN مما يساعدنا في اقتراح وتنفيذ عدة تطبيقات مختلفة.

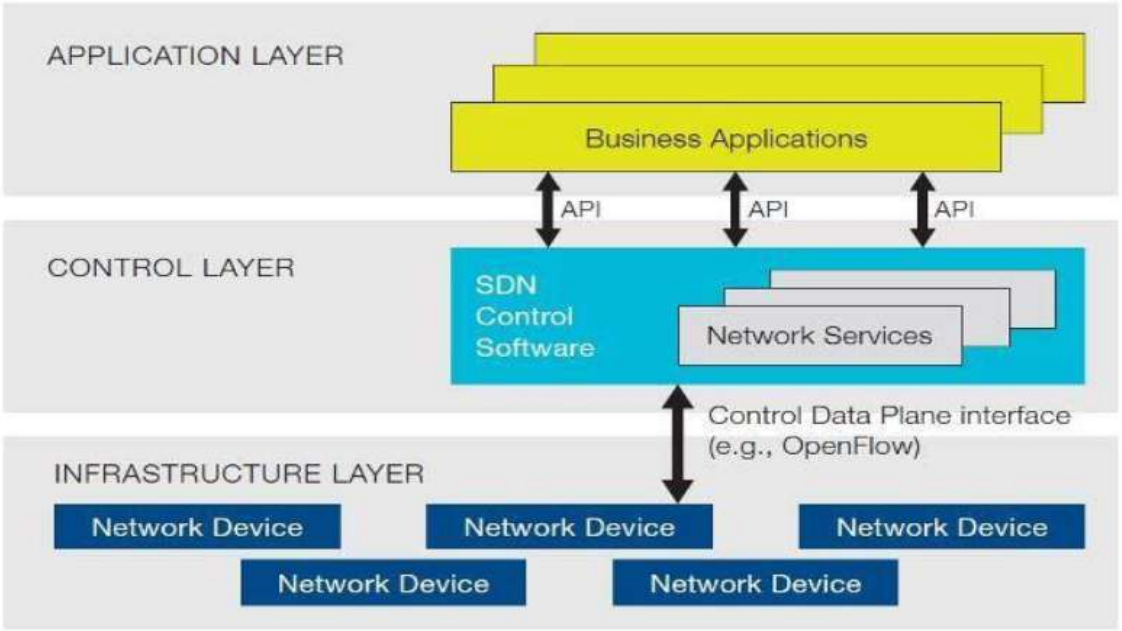
الكلمات المفتاحية: الشبكات المعرفة بالبرمجة، بروتوكول التدفق المفتوح، المتحكم، الإنتاجية، التأخير.

* أستاذ – قسم النظم والشبكات الحاسوبية – كلية الهندسة المعلوماتية – جامعة تشرين – اللاذقية – سورية.
** طالبة دكتوراه – قسم النظم والشبكات الحاسوبية – كلية الهندسة المعلوماتية – جامعة تشرين – اللاذقية – سورية.

مقدمة:

قدمت الشبكات المعرفة بالبرمجة ((Software Defined Networks(SDN) تطوراً كبيراً في معماريّة الشبكات بما يتناسب مع متطلبات التطوير المستمر في عالم الشبكات والسرعة بالأداء، حيث تقوم على مبدأ الفصل بين طبقة توجيه البيانات (Data Palne) وطبقة التحكم (Control Plane) ليصبح دور الأجهزة مقتصراً على تمرير البيانات وتنفيذ أوامر التحكم الصادرة عن المتحكم (Controller)، الذي يعتبر العقل الرئيسي في عمل SDN .

نتيجةً لهذا الفصل أصبحت معمارية الشبكة أفضل، وتم تقسيمها إلى 3 طبقات يوضحها الشكل (1).



الشكل (1): طبقات الـ SDN

وهذه الطبقات هي [1]:

1. **طبقة التطبيقات (Application Layer):** تتكون من الخدمات والتطبيقات التي تقدمها الشبكة للمستخدم.
2. **طبقة التحكم (Control Layer):** تحتوي على المتحكم (Controller) الذي يعتبر أساس هذه التقنية، فهو المسؤول عن قرارات التوجيه والإدارة والتحكم بكامل وظائف الشبكة. تتصل هذه الطبقة مع طبقة التطبيقات من خلال واجهة برمجة التطبيقات ((API) Application Programming Interface).

توجد أنواع مختلفة للمتحكمات من أهمها: OpenDayLight- POX - Beacon–NOX FloodLight.

3. **طبقة البنية التحتية (Infrastructure Layer):** والتي تتكون من أجهزة الشبكة (Routers-Switches) ، تتلقى الأوامر من طبقة التحكم وتقوم بتنفيذها، حيث تتصل هذه الطبقة مع المتحكم عن طريق البروتوكول (OpenFlow) وهو البروتوكول الأساسي في عمل تقنية SDN حيث ينظم التواصل بين المتحكم والبنية التحتية.

إذاً SDN تحقق الفصل بين (Control Plane) و (Data Plane)، مما ينتج عنه العديد من **المزايا** التي تتمتع بها وأهمها [2]:

1- تعتبر SDN بنية شبكية قابلة للبرمجة ويمكن التحكم بها عن طريق المتحكم، مما يؤمن المرونة في الإدارة والتصميم والتنفيذ.
2- توفير في الكلفة المادية للعتاد.
3- تقدم أداءً أفضل من الشبكات التقليدية نظراً لسهولة التحكم بالشبكة من خلال وجود المتحكم.
4- تعتبر التقنية الأفضل في مراكز البيانات لكونها تدعم الحوسبة السحابية، وتؤمن إدارة هذه المراكز بسهولة.

أهمية البحث وأهدافه:

تأتي أهمية البحث من ضرورة فهم مكونات SDN وآلية عملها، كونها تعتبر مستقبل عالم الشبكات الذي يتطور بسرعة ويتجه نحو جعل كل مكونات الشبكة قابلة للبرمجة والإدارة بسهولة.

يهدف هذا البحث إلى تحليل ودراسة كلاً من البروتوكول(OpenFlow) والمتحكم POX ومعرفة الرسائل التي يتم تبادلها بين المتحكم والأجهزة الموجودة في الشبكة، وخاصةً أن هذا البروتوكول مفتوح المصدر وأن المتحكم يمكن برمجته لينفذ التطبيق الذي نريده، مما يسهل إجراء التحسينات الممكنة بهدف رفع كفاءة استخدام SDN في عالم الشبكات.

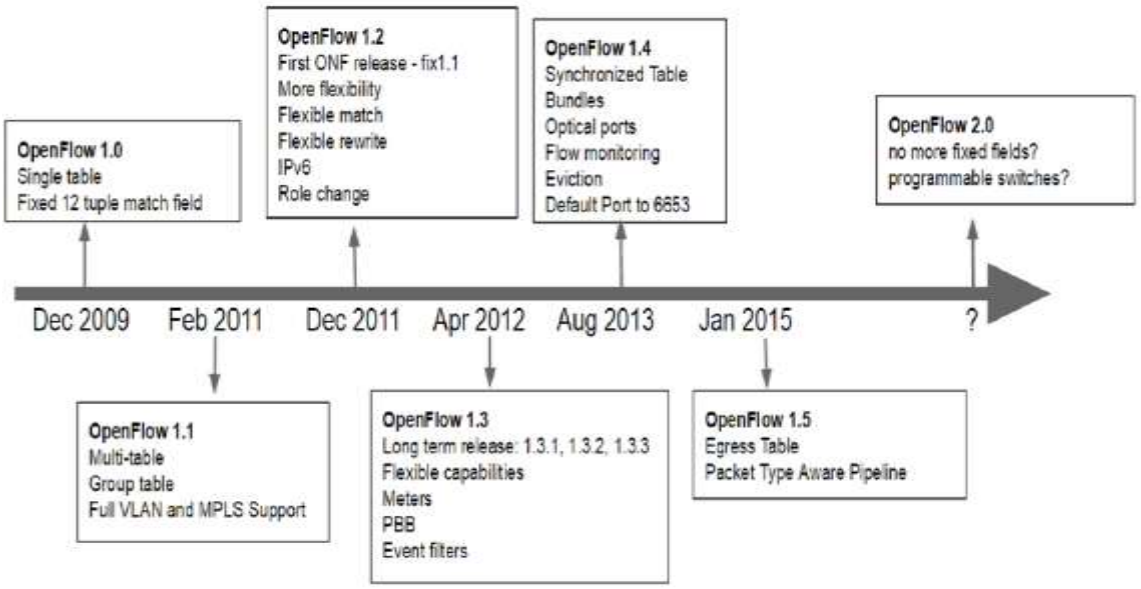
طرائق البحث ومواده:

تم إنجاز البحث بالاعتماد على المحاكي (Mininet) وهو المحاكي الأكثر استخداماً في بناء شبكات SDN، بالإضافة إلى استخدام الأداة (Miniedit) لرسم طوبولوجيا الشبكة، والأداة (Wireshark) لتحليل رسائل البروتوكول OpenFlow، والأداة (Cbench) لقياس أداء المتحكم POX من ناحيتي الإنتاجية والتأخير.

بروتوكول التدفق المفتوح (OpenFlow Protocol):

يعد البروتوكول ((OpenFlow(OF) أو ما يعرف بالجسر الجنوبي (SouthBound Interface) الناظم لكل عمليات التواصل بين أجهزة الشبكة والمتحكم في SDN. وهو بروتوكول مفتوح المصدر، وكانت بداية ظهوره في العام 2009 من قبل منظمة ((Open Network Foundation (ONF) وفق الإصدار OpenFlow v1.0 ، واستمرت إجراء تحديثات عليه حتى وصلنا إلى الإصدار الحالي OpenFlow v1.5 وقريباً سيتم إطلاق النسخة v2.0 .

يوضح الشكل (2) إصدارات البروتوكول OF [3].

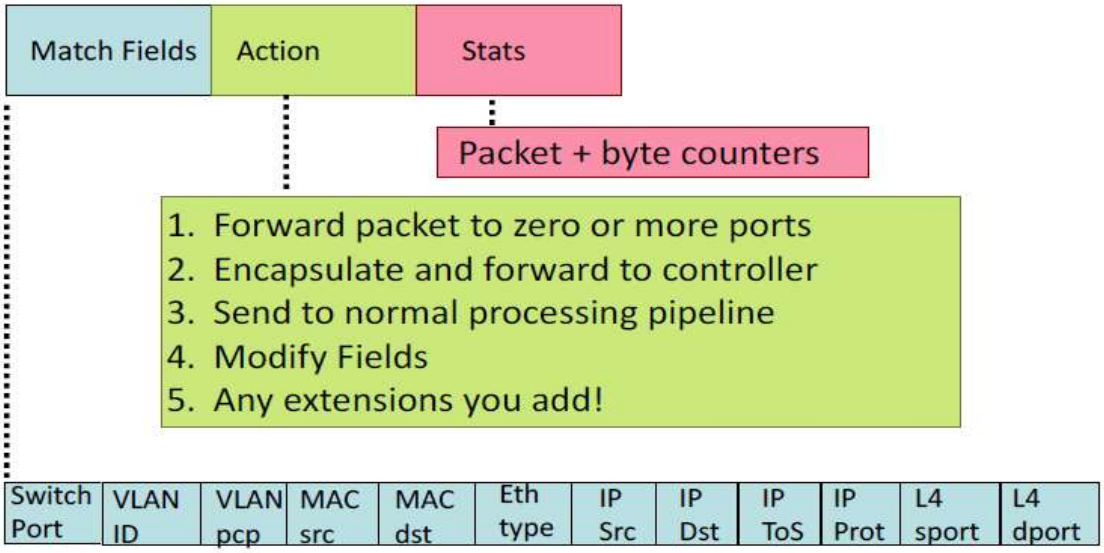


الشكل (2): إصدارات البروتوكول OF

يعمل البروتوكول OF وفق TCP ، ويقوم بتخزين قواعد التوجيه لدى أجهزة الشبكة في جداول التدفق (Flow Tables) المؤلفة من مداخل التدفق (Flow Entries) .يعرّف التدفق (Flow) بأنه الرزم (Packets) الواردة إلى منفذ معين، ويعرف مدخل التدفق(Flow Entry) بأنه الحدث أو التعليمة الواجب تنفيذها على هذه الرزم الواردة وتشمل إما إضافة أو تعديل أو حذف مسار موجود وذلك وفقاً لما يقرره المتحكم [4] .

1 محتويات مدخل التدفق (Flow Entry) :

يوضح الشكل (3) محتويات مدخل التدفق [5].



الشكل (3): محتويات مدخل التدفق

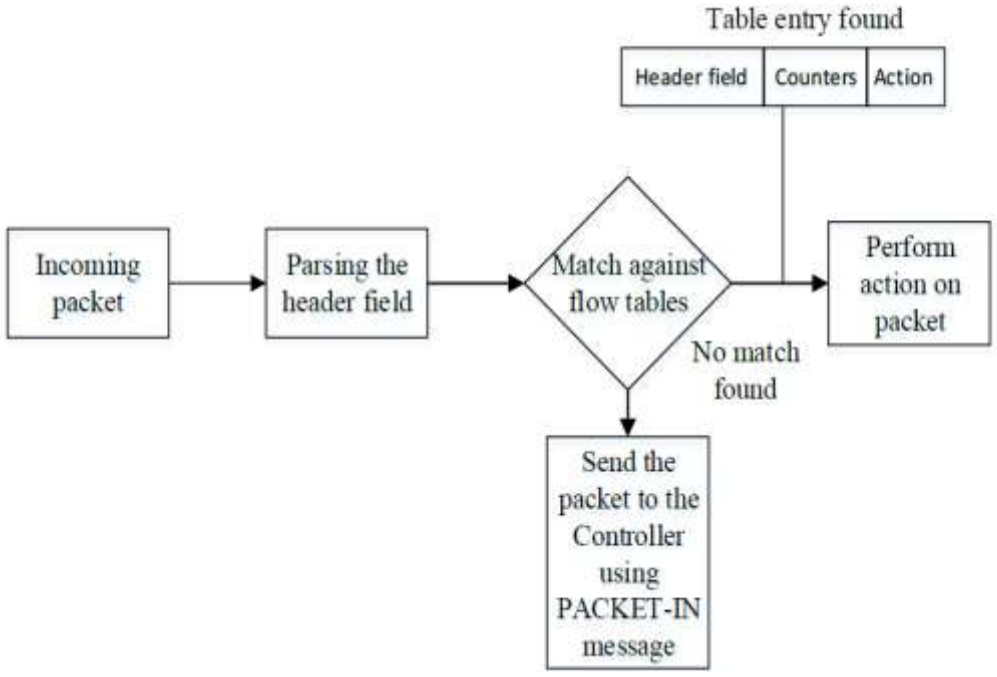
ويمكن تلخيص هذه المحتويات كما يلي:

1. الحقول المتطابقة (Match Fields): تتألف من قسمين وهما الـ (Layer 2) والـ (Layer 3) ومن خلالها نبني المسارات ونحدد من أين وإلى أين سوف تتم عملية التوجيه.
2. الحدث (Action): يمثل الأمر الذي يجب تنفيذه كتمرير الرزمة أو تجاهلها أو إرسالها إلى المتحكم من خلال الأجهزة الموجودة في طبقة (Data Palne) .
3. الإحصائيات (Stats): تمثل عدد البايتات وهي مهمة لمتابعة آخر المستجدات ووضع الـ (Dead Line) للمدخلات الموجودة في الجدول، أي لو في حال لم يتم إيجاد أي مطابقة للرزمة منذ فترة طويلة فسوف يتم حذفها من جدول التدفق.

2 آلية عمل البروتوكول (OpenFlow) :

تقوم الفكرة الأساسية من عمل البروتوكول OF كما يوضح الشكل (4) أنه عندما يصل تدفّق جديد إلى المبدّل تتم مطابقة هذا التدفّق مع مجموعة القواعد أو المداخل الموجودة في جدول التدفّق، ومن ثم يتم التعامل مع هذا التدفق وفق الحالات التالية [6] :

- في حال حدوث مطابقة يتم التصرّف بالرّزمة وفقاً للقاعدة الموجودة وتعديل البيانات الموجودة في عمود الإحصائيات.
- في حال عدم وجود مطابقة يتم إرسال هذه الرزمة بشكل كامل أو ترويستها إلى المتحكّم وذلك لاتخاذ القرار المناسب.
- بعد أن يتم اتّخاذ القرار، يقوم المتحكّم بإرسال هذا القرار إلى المبدّل ويقوم بتحديث جدول التدفق الخاص به.
- تتم هذه الآلية وفق تبادل عدة رسائل بين المتحكم والمبدل، والتي سيتم توضيحها في الفقرة (3-4).



الشكل (4): آلية عمل البروتوكول OpenFlow

3 رسائل البروتوكول (OpenFlow) :

يتم تأسيس الاتصال بين المتحكم والمبدل وفق تبادل عدة رسائل أهمها [7]:

1- OFPT_HELLO : وهي رسالة ترحيبية عند بدء تأسيس الاتصال بين المتحكم والمبدل، ويتم تبادلها بكلا الاتجاهين، وتتضمن نسخة البروتوكول (OpenFlow Protocol Version).

2- OFPT_ECHO_REQUEST : يتم تبادلها بكلا الاتجاهين لمعرفة مدى صلاحية الاتصال.

3- OFPT_FEATURES_REQUEST : يتم إرسالها من المتحكم إلى المبدل، ومن خلالها يقوم المتحكم بالاستعلام عن حالة المبدل وميزاته.

4- OFPT_FEATURES_REPLY : تمثل الرد من المبدل إلى المتحكم للطلب OFPT_FEATURES_REQUEST و تتضمن خصائص المبدل التالية:

1- الإعدادات (Configurations) : للتعريف بإعدادات المبدل.

2- تعديل الحالة (Modify State) : وتمثل إضافة أو حذف قواعد للتدفقات في جدول التدفق.

3- قراءة الحالة (Read State) : تمثل الإحصائيات الخاصة لكل تدفق حيث يقوم المبدل بحسابها وإرسال قيمتها إلى المتحكم.

بعد أن يتم تأسيس الاتصال بين المبدل والمتحكم يتم إرسال عدة رسائل بهدف تنظيم التدفقات والتعامل معها مما يحقق إدارة الشبكة من قبل المتحكم بشكل فعال، ومن أهم هذه الرسائل:

1- OFPT_PACKET_IN : في حال تم استقبال رزمة معينة على أحد منافذ المبدل، إلا أنه لا توجد قاعدة مطابقة لهذا التدفق في أحد القواعد المخزنة في جدول التدفق، عندها يقوم المبدل بإرسال هذه الرزمة إلى المتحكم الذي يتخذ القرار المناسب بشان التعامل معها.

2- ADD-MODIFY-DELETE : تمثل (حذف-تعديل-إضافة) للقواعد ضمن جدول التدفق، ويتم إرسالها من المتحكم إلى المبدل.

3- STATE_REQUEST : يرسلها المتحكم إلى المبدل لمعرفة حالة منافذ المبدل، فمثلاً في حال تعطل أحد منافذ المبدل يكون الرد من قبل المبدل برسالة STATE_REPLY .

4- BARRIER_REQUEST : يرسلها المتحكم إلى المبدل لكي يتأكد من أن المبدل قد قام بالتعامل مع كافة الطلبات الواردة قبل هذه الرسالة.

5- OFPT_ERROR : يتم إرسالها من قبل المبدل في حال حدوث خطأ ما ليعلم المتحكم به.

6- SET_CONFIG : يرسلها المتحكم إلى المبدل ليتأكد من تاريخ انتهاء صلاحية القواعد المخزنة في جدول التدفق.

5- المتحكم POX في SDN :

يعتبر المتحكم العنصر الأكثر أهمية في SDN حيث يمثل عقل الشبكة، ويقوم بإصدار القواعد التي تنفذها الأجهزة الموجودة في طبقة (Data Plane) ويتواصل مع هذه الطبقة بواسطة البروتوكول (OpenFlow) ، بينما يتواصل مع طبقة التطبيقات عن طريق ما يسمى بالجسر الشمالي (NorthBound Interface) . تستخدم SDN أنواعاً مختلفةً من المتحكمات ومن أهمها[8] : (POX- NOX - FloodLight - OpenDayLight – Ryu - Beacon - Maestro -Trema).

تختلف هذه المتحكمات عن بعضها بلغة البرمجة التي تم برمجتها من خلالها، وإصدار البروتوكول الذي تدعمه، بالإضافة إلى مزايا تقنية أخرى.

يعتبر المتحكم POX من المتحكمات الهامة والشهيرة في SDN، وقد تمت برمجته بلغة (Python)، كما أنه يوفر منصة برمجية سهلة ومرنة لكتابة التطبيقات التي يمكن تنفيذها في الشبكة [9] .

تتناول أغلب الأبحاث في مجال SDN المتحكم POX لتنفيذ عدة تطبيقات مثل موازنة الحمل وتطبيقات الوسائط المتعددة ونقل الملفات ومخدم ويب وغير ذلك من التطبيقات، مما يتيح إمكانية التطوير المستمر لأداء الشبكة.

النتائج والمناقشة:

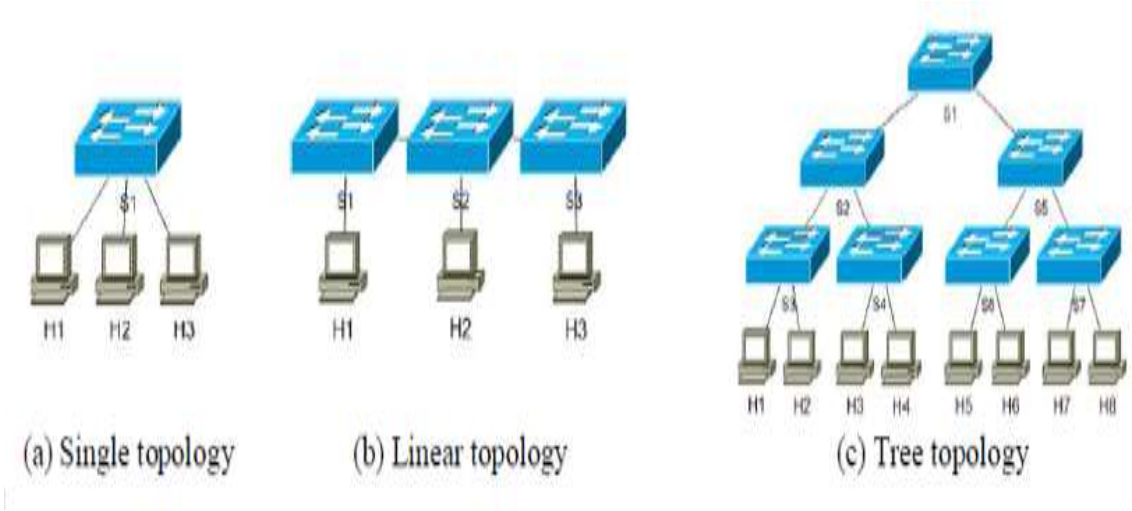
تم إجراء التطبيق العملي لشبكة وفق تقنية SDN باستخدام المحاكي Mininet، وبناء طوبولوجيا الشبكة باستخدام Miniedit، وتحليل رسائل البروتوكول OpenFlow باستخدام Wireshark، ومن ثم تشغيل تطبيقين مختلفين على المتحكم POX، والخطوة الأخيرة هي قياس إنتاجية وتأخير المتحكم POX باستخدام الأداة Cbench.

اعتمدنا في هذا البحث على المحاكي Mininet الذي يفضله أغلب الباحثون في تقنية SDN [10]، لكونه مفتوح المصدر ويؤمن نمذجة العقد والوصلات والمتحكمات وكافة مكوّنات الشبكة بسهولة، حيث يسمح بإنشاء شبكات وفق أحد الطوبولوجيات الموضحة في الشكل (5) وهي [11] :

1- مفرد (Single) : وتتألف من متحكم واحد يتصل مع مبدل واحد الذي بدوره يتصل مع عدد لانهائي من المضيفات.

2- خطي (Linear) : وتتألف من متحكم واحد أو أكثر يتصل مع عدة مبدلات تتصل مع بعضها بشكل متسلسل، وكل منها يتصل مع مضيف واحد.

3- نجمي (Star) : وفيها نلاحظ وجود عدة مستويات تتصل من خلالها المبدلات مع بعضها البعض، والتي بدورها تتصل مع عدد لا نهائي من المضيفات تتواجد في المستوى الأخير.



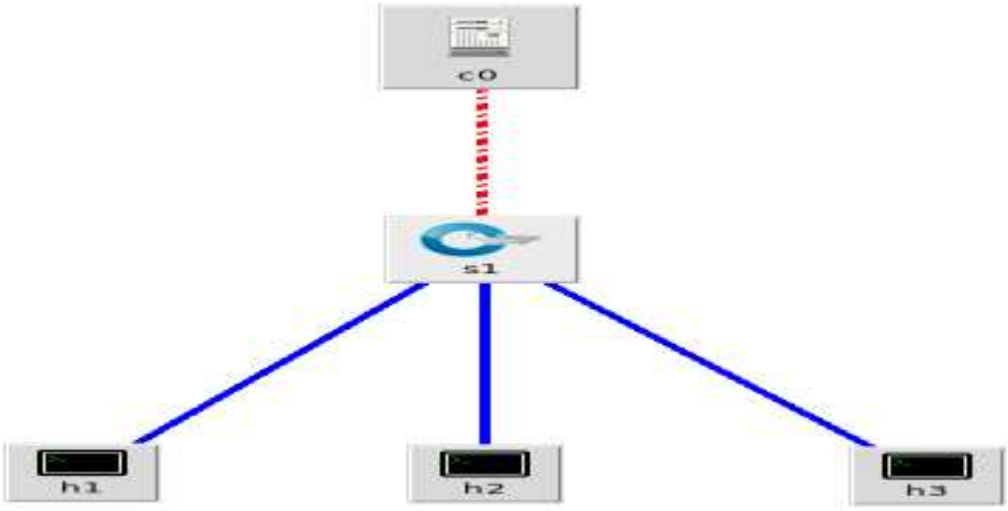
الشكل (5): طوبولوجيا الشبكة في Mininet

كما أن المحاكي Mininet يتضمن المتحكم POX والبروتوكول OF1.0 ، ولتأمين عمل Mininet بما يحقق رسم الشبكة باستخدام الأداة Miniedit، وتحليل رسائل البروتوكول Openflow باستخدام Wireshark ، وقياس أداء المتحكم POX باستخدام Cbench نحتاج المتطلبات التالية:

1. Vmware WorkStation or VirtualBox
2. Mininet package with Ubuntu Server
3. Putty
4. Xming
5. WIRESHARK
6. Miniedit
7. Cbench

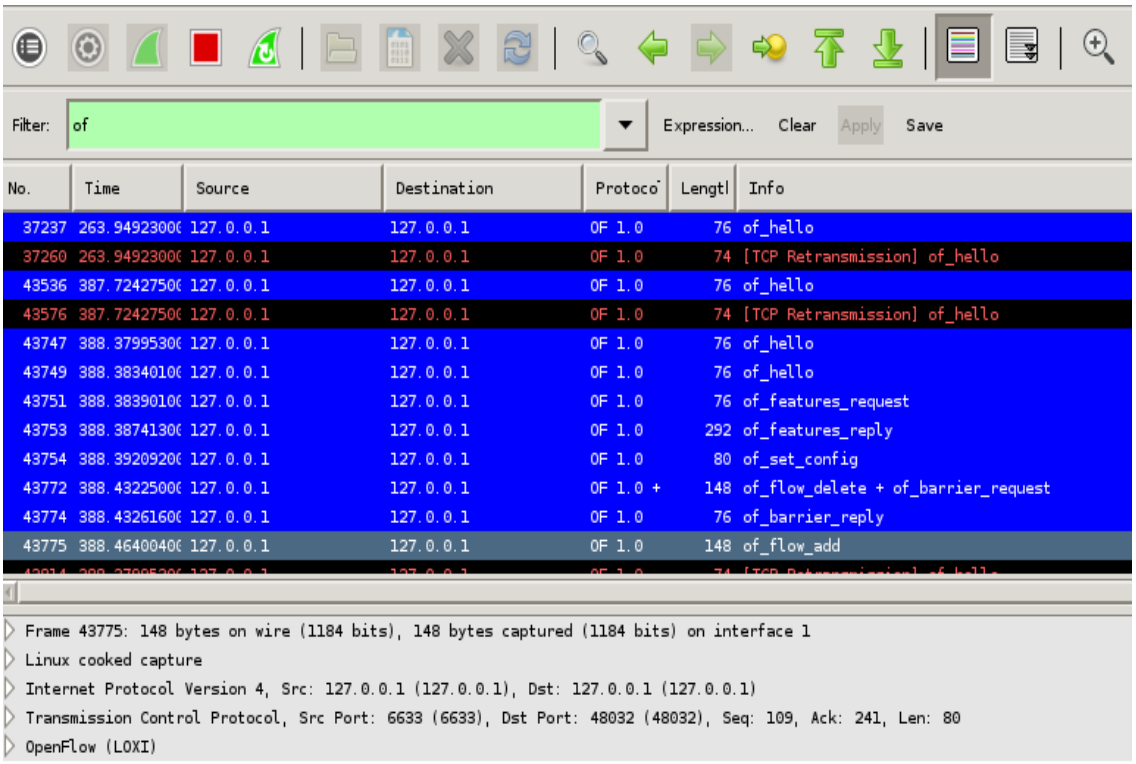
1 سيناريوهات العمل:

1-1 السيناريو الأول: يهدف السيناريو الأول إلى مراقبة رسائل البروتوكول OpenFlow ، حيث قمنا ببناء شبكة باستخدام الأداة Miniedit مؤلفةً من متحكم POX واحد ومبدل واحد (Switch) و3 مضيفات (hosts)، وتتوزع وفق طوبولوجيا من النوع (Single) موضحةً في الشكل (6).



الشكل (6): طوبولوجيا الشبكة من نوع Single

يتم تشغيل الشبكة بهدف مراقبة رسائل البروتوكول OpenFlow المتبادلة بين المتحكم والمبدل بواسطة الأداة Wireshark وحصلنا على الخرج الموضح في الشكل (7).



الشكل (7): رسائل البروتوكول OpenFlow باستخدام Wireshark

يوضح الشكل (7) آلية الاتصال بين المتحكم والمبدل وفق البروتوكول OpenFlow 1.0 وتتضمن كلاً من الخطوات التالية:

- تبدأ آلية مصافحة TCP بين المبدل والمتحكم POX ذو العنوان (loopback)127.0.0.01 على المنفذ Port :6633 ، وبالتالي يتم تبادل رسائل of_hello من كلا الطرفين والتي تتضمن إصدار البروتوكول OpenFlow وهي OF 1.0 وفقاً للشكل (7) .
- يرسل المتحكم of_features_request إلى المبدل لكي يستعلم عن ميزات المبدل الذي بدوره يستجيب برسالة of_features_reply تتضمن قائمة بالمنافذ وسرعة كل منفذ بالإضافة إلى قواعد التدفق والإحصائيات الخاصة بها.
- يرسل المتحكم رسالة of_set_config إلى المبدل ليتأكد من تاريخ انتهاء صلاحية القواعد المخزنة في جدول التدفق.
- يرسل المتحكم أيضاً الرسالة of_barrier_request ليتاكد من أن المبدل قد قام بالتعامل مع كافة الطلبات الواردة قبل هذه الرسالة، ويكون الجواب من المبدل بإرسال رسالة of_barrier_reply .
- يرسل المتحكم of_flow_add لإضافة قاعدة جديدة إلى جدول التدفق.
- ومن ثم يستمر التواصل بين المتحكم والمبدل من خلال packet_in و packet_out و رسائل echo للتحقق من استمرار الاتصال بينهما.

2-1 السيناريو الثاني:

يتناول هذا السيناريو برمجة المتحكم POX ليقوم بدور HUB ، ونستخدم من أجل ذلك نفس الشبكة الموضحة في الشكل (6) والتي تم إنشاؤها باستخدام الأداة Miniedit أو يمكننا الحصول عليها أيضاً بكتابة التعليمة التالية في سطر الأوامر:

mininet@mininet-vm:~$ sudo mn --topo single,4 --controller remote

- mn : تعني إنشاء شبكة.
- Topo single,4: تعني طوبولوجيا من النوع single ومؤلفة من 4 مضيفات تتصل مع المبدل الذي بدوره يتصل مع المتحكم (controller remote) .

نقوم بتشغيل التطبيق الذي يجعل المتحكم POX يقوم بدور HUB، أي أن الرسالة التي يرسلها أحد الأجهزة إلى جهاز آخر، سوف تصل إلى جميع الأجهزة الأخرى عدا الجهاز الذي قام بإرسالها، وبالتالي تحدث حالة غمر (Flooding) في الشبكة، وهذا يمثل عمل الـ HUB .

لتشغيل تطبيق الـ HUB على المتحكم نكتب في الـ Terminal التعليمة:

mininet@mininet-vm:~$ /home/mininet/pox/pox.py log.level –DEBUG forwarding.hub

تمثل هذه التعليمة الانتقال إلى المجلد POX ومن ثم تنفيذ الـ forwarding.hub Script الذي يجعل المتحكم يقوم بدور HUB ويتضمن التعليمات التالية:

```
from pox.core import core
import pox.openflow.libopenfloew_01 as of
from pox.lib.util import dpidToStr
log = core.getLogger()

def_handle_ConnectionUp (event) :
msg = of.ofp_flow_mod()
msg.actions.append (of.ofp_action_output(port = of.OFPP_FLOOD))
event.connection.send(msg)
log.info("Hubifying %s", dpidToStr(event.dpid))

def launch () :
core.openflow.addListenerByName("ConnectionUp",_handle_ConnectionUp)
```

- يبدأ الكود بدايةً باستيراد مكتبة المتحكم POX ومكتبة البروتوكول OpenFlow ليتم تنفيذ التطبيق HUB.
- يبدأ الاتصال بين المتحكم والمبدل كما يوضح التابع Connectionup.
- يعيد التابع() ofp_Flow_mod رسالة msg تعبر عن الرسالة التي يرسلها المتحكم إلى المبدل ليقوم بإضافة مدخل إلى جدول التدفق flow table entry .
- يحدد الصف ofp_action_output منفذ المبدل الذي سيتم إرسال الرزم إليه، وفي هذا التطبيق تم تحديد المنفذ بالتعليمة port=of.OFPP_FLOOD ، والتي تعني توجيه الرزم إلى جميع المنافذ الأخرى عدا المنفذ الذي أتت منه.
- يرسل المتحكم من خلال التابع connection.send(msg) رسائل OpenFlow إلى المبدل.

- لكل مبدل يعمل وفق البروتوكول OpenFlow رقماً خاصاً ((DataPath ID(DPID) بحجم 64 bits، حيث يجب إعلام المتحكم بقيمته خلال عملية المصافحة بينه وبين المبدل، وهذا يتم من خلال التابع ()dpidToStr.
- ومن ثم يتم استدعاء التابع ()launch الذي نستدعي من خلاله التوابع الأخرى لكي تبدأ بالعمل.

إذاً أصبح المتحكم الآن يقوم بدور HUB لنتأكد من ذلك نقوم بإجراء PING من المضيف h1 إلى h2 ونراقب هل الرسائل وصلت أيضاً إلى h3 ؟

ندخل للمضيف h3 وننفذ التعليمة tcpdump لمراقبة ترويسات الرزم المرسلة إليه على الواجهة المحددة وفق التعليمة وفق الآتي:

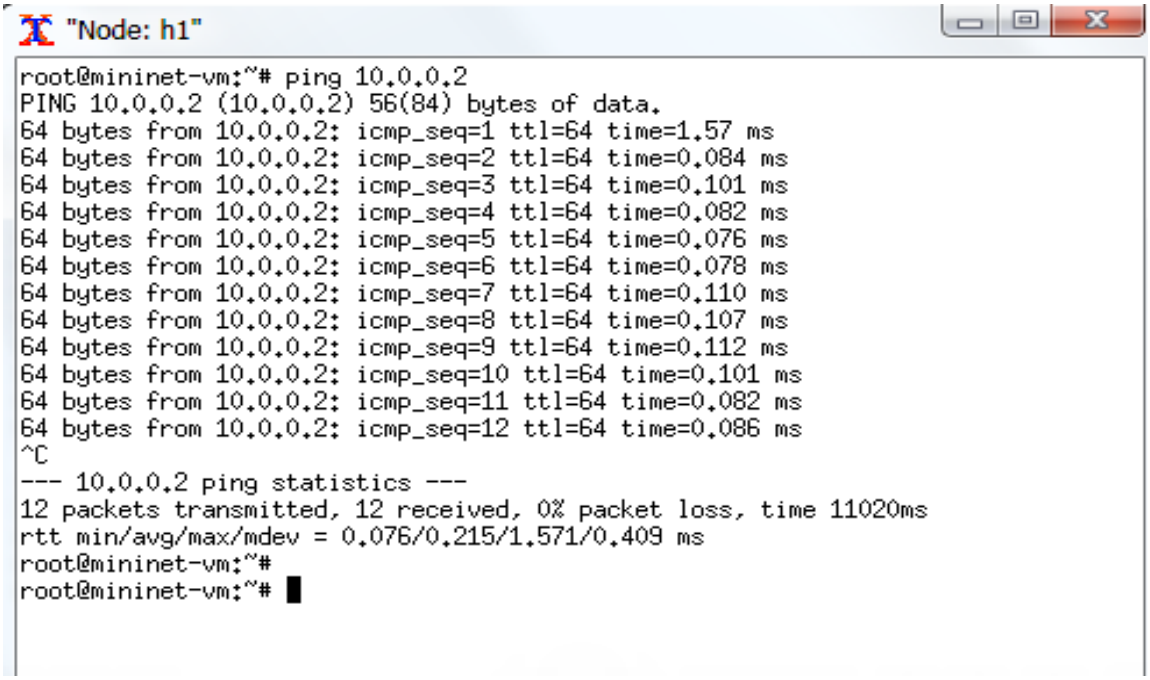
```
root@mininet-vm:~# tcpdump –n –i h3-eth0
```

ومن ثم نقوم بإجراء PING من h1 إلى h2 وفق التعليمة:

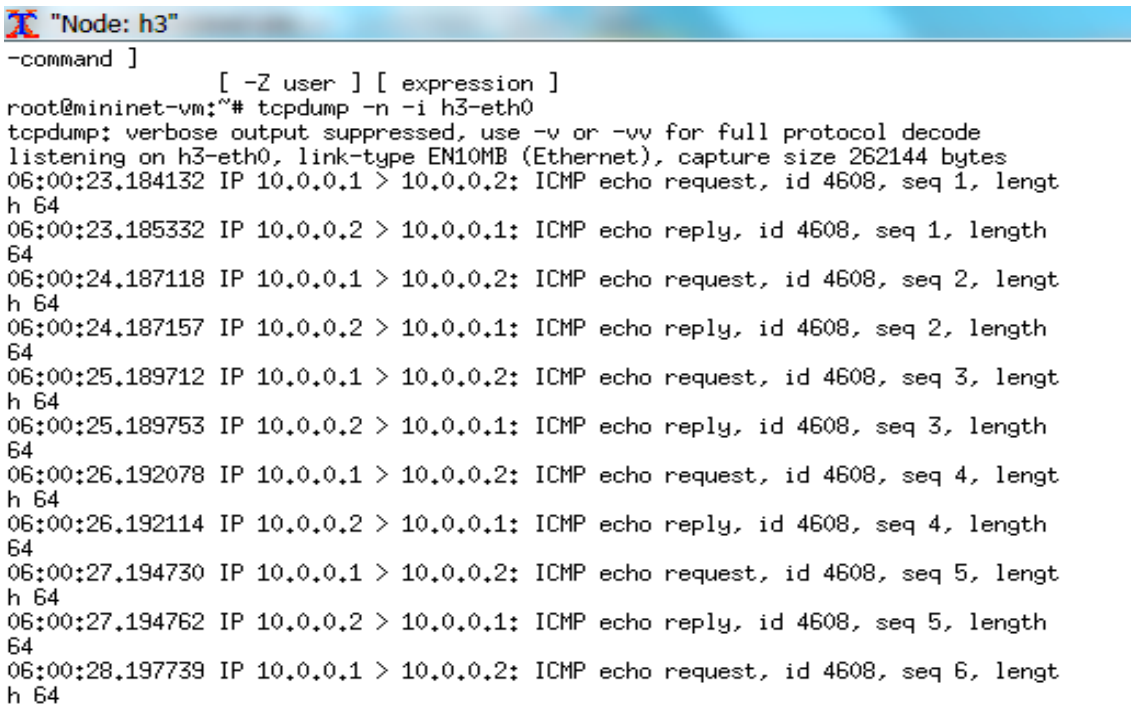
```
root@mininet-vm:~# ping 10.0.0.2
```

حيث 10.0.0.2 هي الـ ip للمضيف h2.

وعندها تتم عملية ping بين h1 و h2 كما يوضح الشكل (8) ، وفي نفس الوقت يستقبل المضيف h3 أيضاً هذه الرسائل كما يوضح الشكل (9) مما يؤكد أن المتحكم قام بدور HUB وأغرق الشبكة بالرسائل .



الشكل (8): إجراء PING بين h1 و h2.



الشكل (9): استقبال المضيف h3 لجميع الرسائل المرسلة في الشبكة.

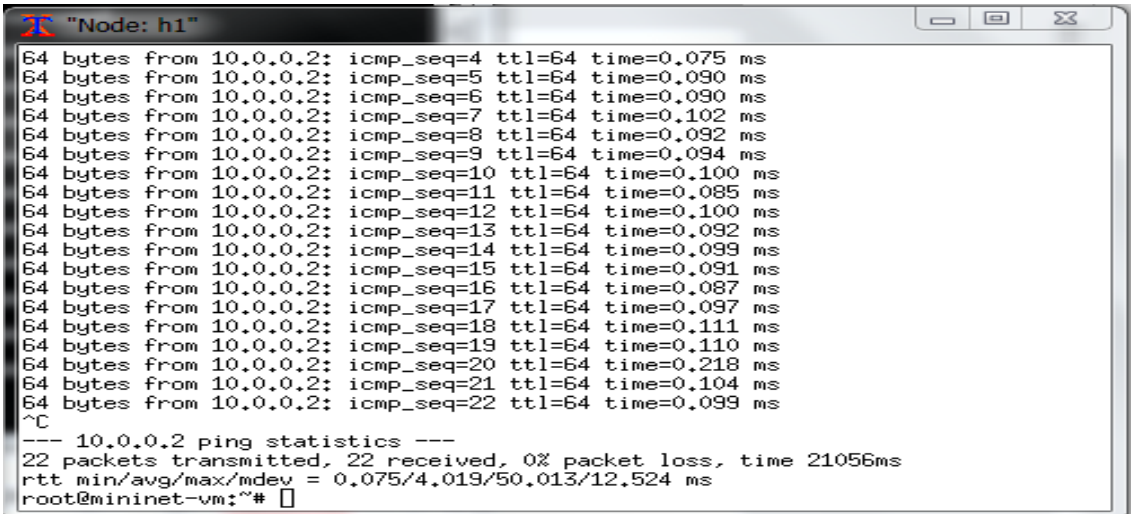
3-1 السيناريو الثالث:

يتناول هذا السيناريو برمجة المتحكم POX ليقوم بدور(layer2 Switching)، وبالتالي عند إرسال رسالة من جهاز إلى آخر لن يتم معرفة الرسالة المرسلة من قبل الأجهزة الأخرى الموجودة في الشبكة.

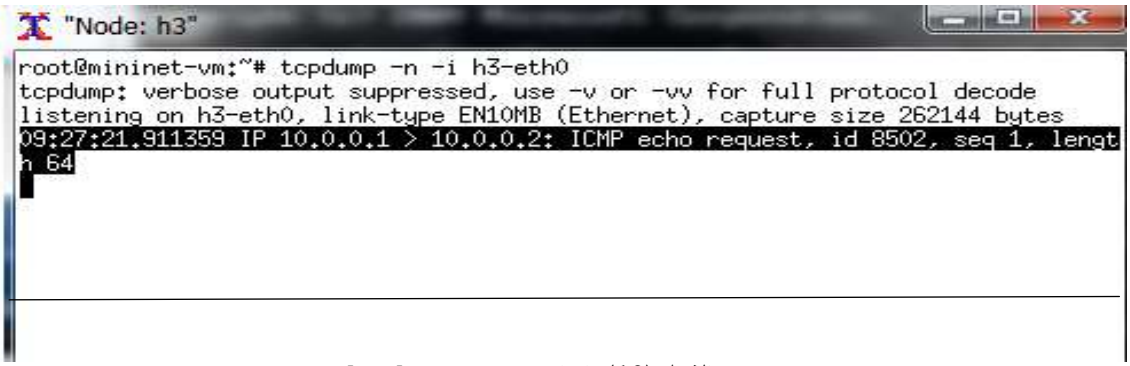
قمنا بالتنفيذ على نفس الشبكة الموضحة في الشكل(6) لملاحظة الفرق بين التطبيقين، ثم ننفذ التعليمة التالية التي تقوم بتشغيل التطبيق forwarding.l2_learning على المتحكم.

mininet@mininet-vm:~$ /home/mininet/pox/pox.py forwarding.l2_learning

وبعد تنفيذ tcpdump على المضيف h3 ، نقوم بإجراء ping بين h1 و h2 كما في الشكل(10)، فنلاحظ أن المتحكم في البداية ليس لديه معرفة مسبقة بالمضيف h3 لذلك تم إرسال الرزمة الاولى لكل من h3 و h2 كما يوضح الشكل (11) ، ولكن بما أنه يعمل وفق التطبيق الذي تم تشغيله forwarding.l2_learning بالتالي لن يتابع إرسال الرزمة إلا للمضيف h2 ويتوقف عن إرسالها لباقي الأجهزة، وهذا يمثل الفرق بين هذا التطبيق والتطبيق الذي تم تنفيذه في السيناريو الأول والذي كان يقوم فيه المتحكم بدور HUB.



الشكل (11): استقبال h3 للرزمة الأولى فقط



الشكل (10): إجراء PING بين h1 وh2

نستنتج من السيناريو الثاني والسيناريو الثالث أنه يمكن برمجة المتحكم وفق التطبيق الذي نريده، وهذا يمثل مفهوم تقنية SDN التي جعلت الشبكة أسهل إدارة وقابلة للبرمجة وتنفيذ التطبيقات التي نريد دون الحاجة إلى تغيير العتاد.

4-1 السيناريو الرابع:

يتناول هذا السيناريو قياس أداء المتحكم POX باستخدام الأداة Cbench من ناحيتي الإنتاجية (Throughput) والتأخير(Latency) من خلال إرسال رسائل إلى المتحكم وفق البروتوكول OpenFlow وانتظار رسالة Flow_mod.

تعمل هذه الأداة وفق الخوارزمية التالية [12]:

```
cbench is a benchmarking tool for controllers

Algorithm:
    pretend to be n switches (n=16 is default)
    create n openflow sessions to the controller
    if latency mode (default):
        for each session:
            1) send up a packet in
            2) wait for a matching flow mod to come back
            3) repeat
            4) count how many times #1-3 happen per sec
    else in throughtput mode (i.e., with '-t'):
        for each session:
            while buffer not full:
                queue packet_in's
                count flow_mod's as they come back
```

وتقوم الخوارزمية بحساب كل من الإنتاجية والتأخير كما يلي:

يرسل Cbench عدة رزم إلى المتحكم (Packet_in) وينتظر استجابة المتحكم من خلال إرسال (Flow_mod) ويتم تكرار ذلك عدة مرات ومنها يحسب قيمة التأخير.

أما الإنتاجية فيتم حسابها من خلال عدد رسائل (Flow_mod) التي يرسلها المتحكم.

تمّ في السيناريو الرابع تشغيل التطبيق (forwarding.l2_learning) الذي تم تنفيذه في السيناريو الثالث على المتحكم POX ، ومن ثم قياس كل من الإنتاجية والتأخير في حال تغيير عدد المبدلات وفق القيم (1,4,16,64).

ننفذ التعليمة:

mininet@mininet-vm:~/oflops$ cbench –c localhost –s 1 –l 2

والتي تتضمن الانتقال إلى المسار oflops وتشغيل الأداة cbench لقياس تأخير المتحكم POX الذي له (IP:127.0.0.1(localhost)) والمنفذ الافتراضي (Port:6633) ، وتنفيذ الاختبار في حال استخدام مبدل واحد (s=1) وتكرار الاختبار مرتين(l=2).

```
mininet@mininet-vm:~/oflops$ cbench -c localhost -s 1 -l 2
cbench: controller benchmarking tool
   running in mode 'latency'
   connecting to controller at localhost:6633
   faking 1 switches offset 1 :: 2 tests each; 1000 ms per test
   with 100000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
10:58:23.419 1   switches: flows/sec:  0   total = 0.000000 per ms
10:58:24.524 1   switches: flows/sec:  193   total = 0.193000 per ms
RESULT: 1 switches 1 tests min/max/avg/stdev = 193.00/193.00/193.00/0.00 responses/s
mininet@mininet-vm:~/oflops$ cbench -c localhost -s 4 -l 2
cbench: controller benchmarking tool
   running in mode 'latency'
   connecting to controller at localhost:6633
   faking 4 switches offset 1 :: 2 tests each; 1000 ms per test
   with 100000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
10:59:04.550 4   switches: flows/sec:  0  0  0  0   total = 0.000000 per ms
10:59:05.654 4   switches: flows/sec:  369  369  369  368   total = 1.474257 per ms
RESULT: 4 switches 1 tests min/max/avg/stdev = 1474.26/1474.26/1474.26/0.00 responses/s
```

الشكل (12): قيمة التأخير في حال استخدام مبدل واحد ثم 4 مبدلات

تمّ تكرار تنفيذ هذه التعليمة مع تغيير عدد المبدلات إلى 4 ثم 16 ثم 64 مبدل. وحصلنا على النتيجة الموضحة في الشكل (12) التي تبين قيمة التأخير في حال استخدام مبدل واحد ثم 4 مبدلات.

يوضح الجدول (1) قيم متوسط التأخير وفقاً لتنفيذ السيناريو الرابع.

الجدول (1): قيم متوسط التأخير مع اختلاف عدد المبدلات

Number of Switches	Latency Average (ms)
1	0.193
4	1.47426
16	2.086
64	2.15764

يبين الجدول (1) أن زيادة عدد المبدلات وفق التطبيق المستخدم (forwarding.l2_learning) يسبب زيادة في التأخير عند المتحكم لأن أي رزمة واردة إلى أي مبدل ولن تجد قاعدة لها في المبدل سيتم إرسالها إلى المتحكم، وبالتالي يزداد الفاصل الزمني بين إرسال الرزمة (packet_in) وانتظار الاستجابة (Flow_mod) المرسلة من قبل المتحكم.

أما بالنسبة للإنتاجية، نكتب التعليمة:

mininet@mininet-vm:~/oflos$ cbench –c localhost –s 1 –l 2 -t

والتي تعني قياس الإنتاجية (t -) من خلال الانتقال إلى المسار oflops وتشغيل الأداة cbench لقياس إنتاجية المتحكم POX الذي له ((IP:127.0.0.1(localhost والمنفذ الافتراضي (Port:6633) ، وتنفيذ الاختبار في حال استخدام مبدل واحد (s=1) وتكرار الاختبار مرتين(l=2) ، وتمّ تكرار قياس الإنتاجية مع اختلاف عدد المبدلات.
يظهر الشكل (13) قيمة الإنتاجية عند تغيير عدد المبدلات بين 1 و 4 مبدلات.

```
mininet@mininet-vm:~$ cbench -c localhost -s 1 -l 2 -t
cbench: controller benchmarking tool
   running in mode 'throughput'
   connecting to controller at localhost:6633
   faking 1 switches offset 1 :: 2 tests each; 1000 ms per test
   with 100000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
11:05:13.150 1   switches: flows/sec:  3407   total = 3.314767 per ms
11:05:14.259 1   switches: flows/sec:  4329   total = 4.327992 per ms
RESULT: 1 switches 1 tests min/max/avg/stdev = 4327.99/4327.99/4327.99/0.00 resp
onses/s
mininet@mininet-vm:~$ cbench -c localhost -s 4 -l 2 -t
cbench: controller benchmarking tool
   running in mode 'throughput'
   connecting to controller at localhost:6633
   faking 4 switches offset 1 :: 2 tests each; 1000 ms per test
   with 100000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
11:05:32.458 4   switches: flows/sec:  335  1120  1050  1050   total = 3.506049 per ms
11:05:33.561 4   switches: flows/sec:  699  698  675  675   total = 2.746986 per ms
RESULT: 4 switches 1 tests min/max/avg/stdev = 2746.99/2746.99/2746.99/0.00 responses/s
```

الشكل (13): قيمة الإنتاجية في حال استخدام مبدل واحد ثم 4 مبدلات

وكانت النتائج بالنسبة للباقي موضحة في الجدول (2).

الجدول (2): قيم متوسط الإنتاجية مع اختلاف عدد المبدلات

Number of Switches	Throughput Average (flows/sec)
1	4327.99
4	2746.99
16	1836.16
64	0

توضح قيم الجدول (2) انخفاض قيمة الإنتاجية للمتحكم مع ازدياد عدد المبدلات في الشبكة، وهذا يعود لأن عدد رسائل (flow_mod) التي يرسلها المتحكم سوف يتناقص بسبب امتلاء الرتل بالرسائل (packet_in) والتي لم تتم معالجتها، حتى تصبح قيمة الانتاجية معدومة عندما نستخدم 64 مبدل بسبب وجود حالة من غمر الشبكة برسائل (packet_in) التي لم ترسل إلى المتحكم.

الاستنتاجات والتوصيات:

تناولنا في هذا البحث محاكاة شبكة SDN باستخدام المحاكي Mininet ، وقمنا بتحليل البروتوكول OpenFlow الذي ينظم الاتصال بين المتحكم والمبدل وفق إرسال عدة رسائل تمت مراقبتها بواسطة الأداة Wireshark.

وطالما أن تقنية SDN تتميز بوجود المتحكم الذي ينظم كل العمليات في الشبكة، لذلك اخترنا المتحكم POX الأكثر شهرة في SDN من أجل إجراء أكثر من تطبيق لكي نحقق مفهوم الشبكة المعرفة بالبرمجة، وقياس إنتاجية وتأخير هذا المتحكم باستخدام الأداة Cbench ، وأظهرت النتائج أن زيادة عدد المبدلات يسبب زيادة في التاخير وانخفاض قيمة الإنتاجية، ومنه نستنتج أن عدد المبدلات يؤثر على أداء المتحكم وهذا التأثير يختلف أيضاً وفق التطبيق المستخدم وطوبولوجيا الشبكة.

يساعد هذا البحث في فهم مكونات SDN وبالتالي كيفية دمج SDN مع تقنيات أخرى مثل WiFi و 5G وغيرها من التقنيات الأخرى، كما أنه يفتح المجال لإمكانية تنفيذ تطبيقات مختلفة مثل موازنة الحمل وVoIP و Firewall ، ومن الممكن مقارنة أداء المتحكم POX مع متحكمات أخرى مثل FloodLight و NOX و OpenDayLight وكل ذلك يهدف إلى تحقيق الاستخدام الأفضل لتقنية SDN .

المراجع:

[1] SOUAD, F.; MOUGHIT,M. ; IDBOUFKER, N. *OpenFlow Controllers Performance Evaluation.* International Journal of Emerging Research in Management &Technology, May 2016.

[2] OMINIKE, A. ; ADEBAYO, A. ; OSISANWO, F. *Introduction to Software Defined Networks (SDN).* International Journal of Applied Information Systems (IJAIS), Volume 11 – No. 7, December 2016.

[3] HAO, C.;DAR LIN, Y. *OpenFlow Version Roadmap*. 2015, 1 September. 2018. < http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf >

[4] CHHIKARA,P.; MATHARU,G.; DEEP,V. *Towards OpenFlow Based Software Defined Networks,* IEEE, 2014.

[5] GENG LI. Basic network flows; *OpenFlow as a datapath programming standard*. 2017, 9 October.2018. <http://zoo.cs.yale.edu/classes/cs434/cs434-2017-spring/lectures/02-prognet-openflow.pdf>

[6] BRAUN, W.; MENTH,M. *Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices*. Future Internet, 2014.

[7] SUZUKI, K.; SONODA, K.; TONOUCHI,T.;SHIMONISHI,H. *A Survey on OpenFlow Technologies*. IEICE TRANS. COMMUN., VOL.E97–B, NO.2 FEBRUARY 2014.

[8] SUDARSANA RAJU, V R. *SDN CONTROLLERS COMPARISON*. Science Globe International Conference, Bengaluru, India, 10th June, 2018.

[9] Fancy,C. ; Pushpalatha,M. *Performance Evaluation of SDN controllers POX and Floodlight in Mininet Emulation Environment.* IEEE, 2017.

[10] AL-SOMAIDAI,M. ; YAHYA,E. *Survey of software components to emulate OpenFlow protocol as an SDN implementation*. American Journal of Software Engineering and Applications,2014**.**

[11] ROWSHANRAD,S. ; ABDI, V. ; KESHTGARI,M. *PERFORMANCE EVALUATION OF SDN CONTROLLERS: FLOODLIGHT AND OPENDAYLIGHT*. IIUM Engineering Journal, Vol. 17, No. 2, 2016.

[12] oflops ,github, 20 August. 2018.
https://github.com/mininet/oflops/tree/master/cbench