مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (40) العدد (5) 2018
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (40) No. (5) 2018

تَقييمُ أداءِ المُتحكّمات في الشّبكات المُعرَّفة بالبَرمجيات

د. محمد صبيح*
الحسن أبو عبيد**

(تاريخ الإيداع 27 / 6 / 2018. قُبِل للنشر في 21/ 10 / 2018)

□ ملخّص □

تُعد عملية فَصل اتّخاذ قراراتِ التّوجيه عن عَمليةِ توجيهِ البيانات جوهرُ تقنيةِ الشّبكات المُعرَّفة بالبرمجيّات. أحدُ أهمِّ مكوّنات هذه التقنية هو المُتحكّم والذي يُعتبر المكوّن الأذكى في الشبكة. لقد تم تطويرُ العديدِ منَ المتحكمات منذ أن نشأت هذه التقنية، وتطرقت الكثير من الأبحاث إلى مقارنةِ أداءِ العديد منها بالنسبةِ للإنتاجية والتّأخير والحماية. ونظراً لأهمية اختيار المتحكم المناسب حسب البارامترات والظروف المختلفة للشبكة قُمنا في هذا البحث بدراسةِ أداءِ أربعةِ متحكماتٍ وهي Floodlight, Beacon, NOX, RYU من حيث الإنتاجية وزمَنُ الرّحلة الانكفائيّة RTT بالإضافةِ إلى زَمنِ تأسيسِ الاتّصال مع مُبدِّل الشبكة وزَمن إضافة مَدخل إلى جَدول التّدفُّق Flow Table للمبدل. النتائجُ أظهَرت تفوُّق المتحكّم Beacon من حيث الإنتاجية عندما يكون عدد المبدلات في الشبكة مساوياً لعدد أنوية المعالج للجهازِ الّذي يَعمَلُ عليهِ المُتحكّم، أمّا ما يتعلّقُ بزمنِ الرّحلةِ الانكفائيّةِ وزَمنِ إضافةِ مدخَلٍ إلى جَدولِ التّدفُّق فقد حَقق المتحكم NOX أقلَّ زمن، وأخيراً كان المتحكم Floodlight هو الأفضل زمنياً من حيثُ تأسيسِ الاتّصال مع المُبدّل.

الكلمات المفتاحية: المتحكم، طبقة التحكم، طبقة البيانات، جدول التدفق، الشبكات المُعرَّفة بالبرمجيات

*أستاذ مساعد في قسم النظم والشبكات الحاسوبية – كلية الهندسة المعلوماتية – جامعة تشرين – اللاذقية – سورية
**طالب دراسات عليا (ماجستير) في قسم النظم والشبكات الحاسوبية –كلية الهندسة المعلوماتية–جامعة تشرين–اللاذقية–سورية

مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (40) العدد (5) 2018
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (40) No. (5) 2018

Performance evaluation of controllers in software-defined networks

Dr. Mohammed Sobih*
Alhasan Abo Obaid**

(Received 27 / 6 / 2018. Accepted 21 / 10 / 2018)

□ ABSTRACT □

Decoupling the decision-making process from the data forwarding process is the heart of software-defined networks technology. One of the most important components of this technology is the controller, which is the smartest component in the network. Many of the controllers have been developed since the technology originated, and many researches have been done to compare the performance of these controllers for productivity, delay and protection.

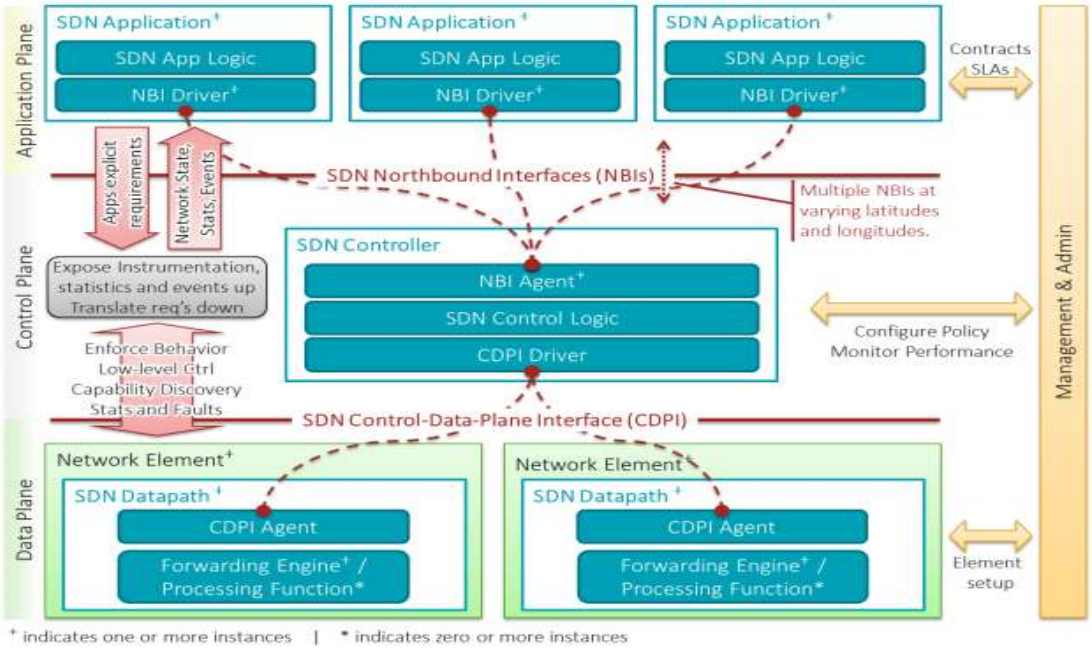
And due to the importance of selecting the appropriate controller according to different parameters and network states, we studied the performance of four controllers: Floodlight, Beacon, Nox, RYU in terms of productivity, RTT, time of establishing connection with an OpenFlow switch and the time for adding an input to the switch flow table. The results showed that the Beacon control was superior in performance when the number of switches in the network was equal to the number of processor cores used by the controller. For RTT and the time needed to add an input to the flow table, the NOX controller achieved less time. Finally, the Floodlight controller was the best in terms of establishing connection with the switch because it needed less time.

Keywords: controller, control plane, data plane, flow table, software-defined networks

* **Professor, Department of Networks and Operating Systems. Faculty of Information Technology, Tishreen University, Lattakia, Syria**
** **Postgraduate Student (MSc), Department of Networks and Operating Systems. Faculty of Information Technology, Tishreen University, Lattakia, Syria**

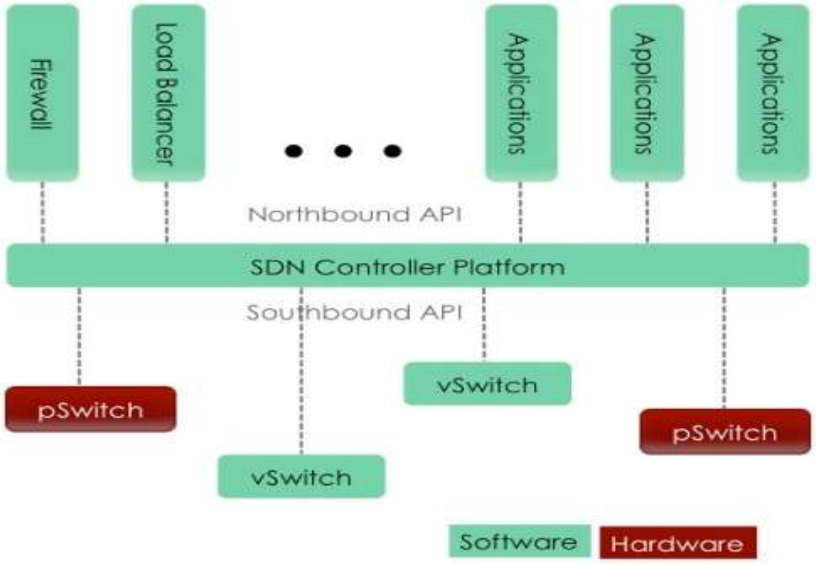
مقدمة

الشَبكات المعرَّفة بالبرمجيات هي أكثر المواضيع المطروحة للنقاش في السنوات الأخيرة [1]. فقد قدَّمت العديد من الميزات وساهمت في حل العديد من المشاكل التي عانت منها الشبكات التقليدية. تعتمد الشبكات المعرفة بالبرمجيات على فصل عمليات توجيه البيانات عن عمليات التحكم واتخاذ قرارات التوجيه لجعل التحكم بالشبكة مركزياً [2] كما يظهر في الشكل (1) وبالتالي تقليل الأخطاء التي من الممكن أن تتعرض لها الشبكات [3]. يُعتبر بروتوكول OpenFlow [3] [4] هو الأكثر شيوعاً واستخداماً في متحكمات الشبكات المعرفة بالبرمجيات للتواصل والتحكم بالمبدلات. باستخدام هذا البروتوكول، يتعلم المبدل معلومات التوجيه من المتحكم ومن ثم تمرير حزم البيانات بالاعتماد على هذه المعلومات.



الشكل (1) بنية شبكة SDN [2]

عرّفت منظمة ONF (Open Networking Foundation) البنية المعمارية لتقنية الشبكات المعرفة بالبرمجيات بنموذج مكون من ثلاث طبقات: طبقة التطبيقات، طبقة التحكم، وطبقة البنية التحتية [2]. تتكون طبقة التطبيقات من الخدمات والتطبيقات التي تقدمها الشبكة للمستخدم مثل التوجيه – ACL – QOS ويتواصل المتحكم مع هذه الطبقة عن طريق الواجهة الشمالية Northbound API كما يتواصل مع طبقة البنية التحتية عن طريق واجهة التخاطب الجنوبية Southbound API [2] كما في الشكل (2).



الشكل (2) واجهة التخاطب الشمالية والجنوبية [2]

عندما يتم إضافة مبدل إلى الشبكة لأول مرة، أول ما يقوم به هو تأسيس اتصال TCP أو TLS مع المتحكم، حيث يرسل رسالة HELLO إلى المتحكم متضمنة أعلى اصدار من بروتوكول OpenFlow يدعمه المبدل، ليرد المتحكم برسالة HELLO ويُتبعها برسالة FEATURE_REQUEST طالباً من المبدل إرسال الميزات والإمكانيات التي يدعمها. بمجرد استلام المتحكم رسالة FEATURE_REPLY من المبدل ينهي عملية تأسيس الاتصال برسالة SET_CONFIG يرسلها إلى المبدل متضمنةً مجموعة من الإعدادات [4].

عند وصول رزمة بيانات إلى المبدل عبر منفذ معين، يقوم المبدل بالبحث عن تطابق في جدول التدفق Flow Table حيث يتم الأخذ بعين الاعتبار التطابق ذو الأولوية العليا ليقوم بعدها المبدل بتوجيه الرزمة إلى المنفذ المناسب. يُعتبر المتحكم عقل الشبكة والمكوّن الأكثر ذكاءً، فهو يملك نظرة شاملة عن الشبكة والوصلات فيها ويقوم بإدارتها ومراقبتها واتخاذ قرارات التوجيه المناسبة ثم ارسالها إلى المبدلات. وبناءً على تغيُّر حالة الشبكة يقوم بتعديل قواعد التوجيه وتغيير جداول التدفق وغيرها من الوظائف [5].

يُشار إلى المتحكّم عادةً بأنّه نظام تشغيل الشبكة [6]. هناك العديد من أجهزة المُتحكمات بَعضها مفتوح المصدَر والبعض الآخر تجاري وهي تختلف أيضا باختلاف لُغات البَرمجة الّتي تم تطويرها بها. أول مُتحكم تم تطويره كان NOXعام 2008 وهو متحكم لا يدعم تعدد النّياسب [7] وقد تم تطويره بلغة c++ وكان مُنطلقاً وأساساً لتطوير متحكمات أخرى.

معظم المتحكمات تستخدم بروتوكول LLDP لاكتشاف المسارات في الشبكة ومن ثم تطبق خوارزمية المسار الأقصر عند اتخاذ قرارات التوجيه مع الأخذ بعين الاعتبار سياسات جودة الخدمة والسياسات الأخرى المطبَّقة [8].

تختلف عملية إدارة الشبكة في SDN عن الشبكات التقليدية IP networks بأن الشبكات المعرفة بالبرمجيات تجعل الإدارة مركزية في المتحكم وبالتالي فإن المبدلات لا تعرف شيئاً عن بقية الشبكة، فقط تتلقى الأوامر من المتحكم. لذلك عند تغير الطوبولوجيا أو عند وجود الأخطاء سيكون هناك تأخيراً إضافياً في معالجة الخطأ لأن المبدلات ستنتظر أن

يقوم المتحكم بحساب الطوبولوجيا الجديدة ومن ثم اختبار الخصائص وتوليد القواعد المناسبة التي سيقوم فيما بعد بإرسالها إلى المبدلات.

أهمية البحث وأهدافه

العديد من المتحكمات قد تم تطويرها منذ أن نشأت فكرة SDN، إن اختيار المتحكم المناسب والذي يلبي متطلبات الشبكة هو تحدي بذاته حيث يجب أن يتميز المتحكم بمعدل إنتاجية Throughput عالٍ وزمن استجابة منخفض بالإضافة إلى تأمين الحماية من طرود المتطفلين وتأمين استقرار الشبكة.

تكمن أهمية البحث في اختيار متحكمات مناسبة للعمل في الشبكات المعرفة بالبرمجيات وبالتالي زيادة كفاءة هذه الشبكات وزيادة فعاليتها.

إن الهدف من هذا البحث هو دراسة المتحكمات NOX, Floodlight, Beacon, RYU والتي تختلف عن بعضها من حيث دعم تعدد النياسب ومن حيث لغة البرمجة التي طُوِّرَ بها المتحكم. من ثم تقييم أدائها بالنسبة لعدة بارامترات أهمها الإنتاجية Throughput وزمن الرحلة الانكفائية RTT الذي يحتاجه أول طرد يتم ارساله في الشبكة بالإضافة إلى زمن تأسيس الاتصال بين المتحكم والمبدل وأخيراً زمن إضافة مدخل إلى جدول التدفق الخاص بالمبدل. وذلك للوصول إلى اختيار المتحكم الأفضل بحسب متطلبات الشبكة.

طرائق البحث ومواده:

لتحقيقِ هدفِ البحث تمّ اتباعُ المنهَجيّة الآتية:

- ❖ دراسة نظرية تناولت الجانبين الآتيين:
  - o التعريف بالشبكات المعرفة بالبرمجيات وأهميتها ومزايا الانتقال إليها
  - o التعريف بالمتحكمات المدروسة و استعراض مزاياها و المكونات و الخدمات التي يقدمها كل متحكم
- ❖ الدراسة العملية وذلك من خلال استخدام المحاكي Mininet لبناء طوبولوجيا الشبكة وتنفيذ السيناريوهات المختلفة لمقارنة أداء المتحكمات من حيث زمن الرحلة الانكفائية RTT وزمن تأسيس الاتصال مع المبدل والزمن اللازم لإضافة مدخل إلى جدول التدفق الخاص بالمبدل. كما قمنا باستخدام الأداة CBench لقياس إنتاجية كل متحكم والتي يمكن التعبير عنها بعدد طرود packet_in التي يمكن للمتحكم أن يعالجها في الثانية.

1 الدراسة النظرية:

الشبكات المعرفة بالبرمجيات هي تقنية ناشئة ومنهج جديد في إدارة شبكات الحاسب، حيث يستطيع مسؤول الشبكة إدارة الشبكة بطريقة مجرّدة بعيداً عن معرفة تفاصيل الشبكة في الطبقات السفلى. وهي تقنية ديناميكية قابلة للبرمجة وذات هيكلية منظمة بشكل جيد. وهي تفصل الشبكة إلى مستويين:

- مستوى التحكم Control Plane
- مستوى البيانات Data Plane

حيث تقوم الشبكات المعرفة بالبرمجيات باستخلاص الوظائف الموجودة في مستوى التحكم من المبدلات Switches وتحقيقها بشكل برمجي ضمن تطبيقات مخصصة لهذه الغاية تسمى متحكمات Controllers

التوفير في تكاليف بناء وإدارة أجهزة البنية التحتية للشبكات هي أهم مزايا الشبكات المعرفة بالبرمجيات كما تقدم تحكم وإدارة مركزية وشبكة قابلة للبرمجة مما يتيح المرونة في التصميم والإدارة وتحسين في أمن ووثوقية الشبكة.

إن المتحكم في الشبكات المعرفة بالبرمجيات مسؤول عن المحافظة على استقرار الشبكة وتطبيق السياسات والقواعد في الشبكة بالإضافة إلى توزيع التعليمات إلى أجهزة الشبكة المختلفة. فالمتحكم يملك نظرة عامة منطقية عن كل مكونات الشبكة.

اقتُرحتِ العديدُ من المتحكمات منذ أن تم إيجاد بروتوكولOpenFlow . أول متحكم مفتوح المصدر اقتُرح كان المتحكم NOX [7] . لاحقاً تم اصدار متحكمات أخرى مثل Maestro مبني بلغة جافا وBeacon وهو متحكم متعدد النياسب مبني بلغة الجافا ومتحكمات أخرى عديدة كـ ONOS وFloodlight وOpendaylight وغيرها.

بالنسبة إلى تطبيقات الشبكات المعرفة بالبرمجيات، تتمثل إحدى الوظائف الرئيسية التي يوفرها المتحكم في واجهة برمجة التطبيقات API للوصول إلى الشبكة. في بعض الحالات، تكون واجهة برمجة التطبيقات الشمالية هي واجهة ذات مستوى منخفض، مما يوفر إمكانية الوصول إلى أجهزة الشبكة بطريقة مشتركة ومتسقة. قد يوفر المتحكم واجهات برمجة التطبيقات عالية المستوى تعطي تجريدًا للشبكة نفسها، بحيث لا يَلزم أن يهتم مطور التطبيق بالأجهزة الفردية بل يهتم بالشبكة ككل.

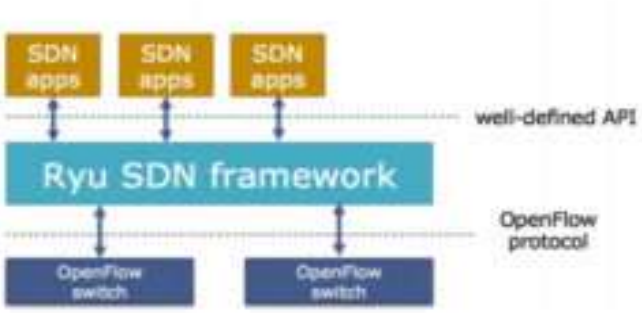
1-1 المتحكم NOX [9,7]

أول متحكم تم تطويره في سنة 2008 جنباً إلى جنب مع بروتوكول OpenFlow وقد كان الأساس لتطوير العديد من المتحكمات التي جاءت بعده. يقدم هذا المتحكم واجهة برمجة تطبيقات API مكتوبة بلغة c++ وهو يدعم الإصدار 1.0 من بروتوكولOpenFlow . يتضمن مكونات بسيطة لاكتشاف الشبكة والتعامل مع مبدلات OpenFlow وهو لا يدعم تعدد النياسب.

تقدم نواة NOX وظائف مساعدة كمعالجة الرزم وإدارة الأحداث المختلفة بالإضافة إلى واجهة التخاطب مع مبدلات OpenFlowوعمليات الادخال والإخراج I/O. وتتضمن الطبقة الوسطى المكونات المبنية ضمناً وهي مدير الاتصال – مدير OpenFlow – مدير الأحداث. يمثل نظام الأحداث أحداثاً عالية المستوى ومنخفضة المستوى في الشبكة، هذه الأحداث تقدّم معلومات معينة وآلية معالجتها. تتضمن هذه الأحداث ما يلي: اتصال مبدل جديد – حذف مبدل – وصول رزمة جديدة – إضافة مدخل إلى جدول التوجيه ...

2-1 المتحكم RYU [10]

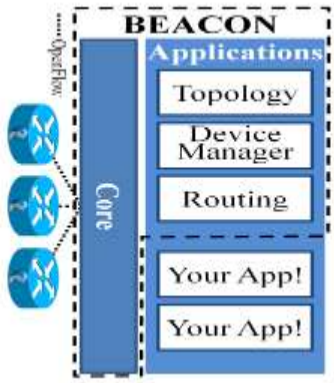
هو متحكم مفتوح المصدر مكتوب بلغة البايثون، يقدم مكونات مُعرَّفة بشكل جيد تمكن المطوّرين من كتابة تطبيقات عديدة للشبكة. يدعم RYU معظم إصدارات بروتوكول OpenFlow من 1.0 وحتى 1.5. يدعم بروتوكول STP وبذلك يمنع حدوث الحلقات في الشبكة. وهو لا يدعم تعدد النياسب (Multithreading) في عمله. يبين الشكل (3) هيكلية المتحكم RYU



الشكل (3) هيكلية RYU

3-1-3 المتحكم Beacon [11]

المتحكم Beacon هو متحكم مفتوح المصدر مبني بلغة الجافا ويدعم بروتوكول OpenFlow في عمله. تم إنشاؤه عام 2010 وهو مستخدم بشكل كبير في الأبحاث والتعليم وكان الأساس للمتحكم Floodlight والعمليات المُقادة بالأحداث بالإضافة إلى النّمطية وتعدد النياسب. يقدم Beacon منصة للتحكم بأجهزة الشبكة باستخدام بروتوكول OpenFlow بالإضافة لمجموعة من التطبيقات المبينة داخلياً والتي تؤمن وظائف مستوى التحكم Control Plane الرئيسية. يمكن له أن يعمل على أية منصة تشغيل بالإضافة إلى أنه ديناميكي أي يمكن تشغيل وإيقاف وإعادة تشغيل أية خدمة فيه دون الحاجة لإعادة تشغيله. يبين الشكل (4) هيكلية المتحكم



الشكل (4) هيكلية المتحكم Beacon

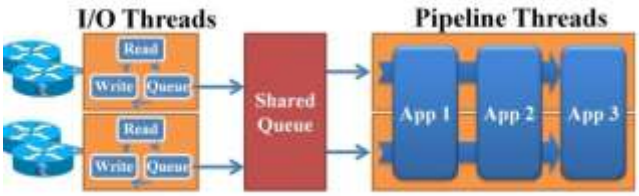
يوفر Beacon أداء عالٍ جداً من خلال دعمه لتعدد النياسب وذلك من خلال طريقتين: الرتل المشترك – Run To Completion

1-3-1 الرتل المشترك:

يتكون من مجموعتين من النياسب. أول مجموعة من النياسب هي نياسب I/O تقوم بقراءة رسائل OpenFlow من المبدلات ثم وضعها في رتلٍ مشترك. يتم اسناد كل مبدل لنيسب إدخال / إخراج واحد، يمكن إسناد عدة مبدلات لنفس النيسب. يقوم هذا النيسب أيضاً بكتابة رسائل OpenFlow الخارجة لمبدلاتها.

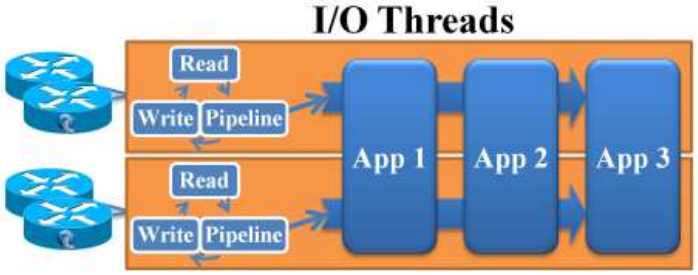
المجموعة الثانية من النياسب، تقوم بإخراج الرسائل من الرتل المشترك وتنفيذ كل رسالة خلال خط معالجة يعود إلى نوع معين من الرسائل. هذا التصميم يستلزم وجود قفل على الرتل المشترك. وهو متغير يمكن أن يمتلك عدة أرتال،

ربما واحد لكل مبدل أو حتى أكثر من واحد لكل مبدل وكل رتل بخصائص مختلفة. هذا يؤدي إلى تحسين وزيادة معالجة الرسائل بين المبدلات من خلال خوارزمية round-robin التي تخدّم الرتل. الشكل (5) يوضح آلية الرتل المشترك.



الشكل (5) الرتل المشترك

Run to completion 2-3-1: تملك مجموعة واحدة من النياسب. كل نيسب يشبه في عمله نياسب I/O في الرتل المشترك ولكن عوضاً عن وضع الرسائل في رتل ليتم تخديمها بخطوط النياسب، فإنه يقوم بقراءة الرسالة كاملة حتى يتم الانتهاء منها والانتقال إلى الرسالة التالية. هذه الطريقة لا تحتاج إلى قفل. يتم تخصيص نيسب لكل مبدل يقوم بقراءَة الرسائل ومعالجتها. كما في الشكل (6)



الشكل (6) آلية Run to Completion

يعاني هذا التصميم من قصور، حيث أنه من الممكن أن تكون أحد النياسب مشغولة بعدد كبير من الرسائل في حين توجد نياسب أخرى في حالة Idle. يستخدم Beacon هذا النوع من المعالجة عندما يكون عدد النياسب محدد مسبقاً.

4-1 المتحكم Floodlight [12]

متحكم مقدم من قبل شركة Big switch يدعم بروتوكول OpenFlow بمختلف إصداراته. وهو مفتوح المصدر مبني بلغة جافا. ويقدم واجهة برمجة تطبيقات REST API متقدمة تساعد على التحكم بكل وظائف المتحكم وعرض قيم بارامتراته في صفحة ويب. عند تشغيل المتحكم يتم معه تشغيل العديد من التطبيقات المكتوبة بلغة الجافا. تعمل الخدمات المقدمة من قبل الواجهة الشمالية REST API على المنفذ 8080 ويمكن لأي تطبيق مكتوب بأية لغة برمجة النفاذ إلى معلومات المتحكم وعرضها وتنفيذ أوامر معينة.

يتضمن المتحكم العديد من الوحدات البرمجية المختلفة التي تلبي متطلبات شبكة SDN مثل اكتشاف الشبكة – إدارة الأجهزة – مدير طوبولوجيا الشبكة – اكتشاف الوصلات – التوجيه وغيرها.. تتصل هذه المكونات مع بعضها عبر واجهات interfaces برمجية تقوم بتحقيقها.

عندما يقوم مضيف ما في الشبكة بإرسال طرد ما إلى مضيف آخر، فإن هذه الرزمة أولاً تصل إلى المبدل، يبحث المبدل في جدول التدفق الخاص به Flow Table عن وجود تطابق، في حال عدم وجود التطابق تُرسَل الرزمة إلى

المتحكم على شكل رزمة packet_in ليقرر بعدها المتحكم وجهة الرزمة وإعلام المبدل بذلك الذي يقوم بإضافة مدخل Flow Entry إلى جدول التدفق الخاص به.

1-5 أداة البحث

تم استخدام المحاكي Mininet [13] وهو محاكي يُنشئ شبكة من المضيفات الظاهرية، والمبدلات، والمتحكمات، والروابط. تستطيع مضيفات Mininet تشغيل برامج نظام Linux القياسية، كما تدعم مبدلاته البروتوكول OpenFlow لتحقيق توجيه مخصص للرزم عالي المرونة. يدعم Mininet البحث والتطوير والتعلم والنماذج الأولية والاختبار وتصحيح الأخطاء وأي مهام أخرى من خلال تقديم شبكة كاملة على كمبيوتر محمول أو أي جهاز آخر.

يعاني هذا المحاكي من قيود تمنعه من تجاوز عرض الحزمة المتاح على مخدم واحد، كما أنه من غير الممكن تشغيل مبدلات وتطبيقات OpenFlow غير المتوافقة مع نظام Linux.

كما تم استخدام الأداة CBench [14] من أجل قياس إنتاجية المتحكم وفق عدد مبدلات ومضيفين معين. إضافةً إلى استخدام الأداة Wireshark من أجل تحليل الرزم المرسلة في الشبكة.

2 الدراسة العملية:

في البداية قمنا بتقييم أداء المتحكمات الأربعة المدروسة من حيث الإنتاجية حيث تم استخدام الأداة CBench والتي تقوم بإنشاء شبكة بعدد معين من المبدلات والمضيفين وتوليد الملايين من رزم packet_in وإرسالها إلى المتحكم مباشرة ليعمل على معالجتها. إن عدد الرزم packet_in التي يمكن للمتحكم أن يعالجها في الثانية الواحدة يُعبِّر عن إنتاجيته، حيث قمنا بتطبيق عدة سيناريوهات من خلال زيادة عدد المبدلات في الشبكة وقياس إنتاجية كل متحكم، ثم زيادة عدد المضيفين مع بقاء عدد المبدلات ثابت.

ثم قمنا بقياس زمن الرحلة الانكفائية Round Trip Time الذي تأخذه أول رزمة تُرسل في الشبكة والزمن اللازم حتى يتم تأسيس الاتصال بين المتحكم والمبدل، وأخيراً الزمن اللازم لإضافة مدخل إلى جدول التدفق الخاص بالمبدل، كل ذلك من خلال بناء شبكة مؤلفة من مبدل ومضيفَين متصلين به باستخدام المحاكي Mininet

مواصفات الجهاز الذي تم استخدامه في القيام بتلك التجارب هي: جهاز بنظام تشغيل ubuntu وبمعالج Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz (4 CPUs)، ~2.4GHz وذاكرة وصول عشوائية 4 غيغابايت.

النتائج والمناقشة:

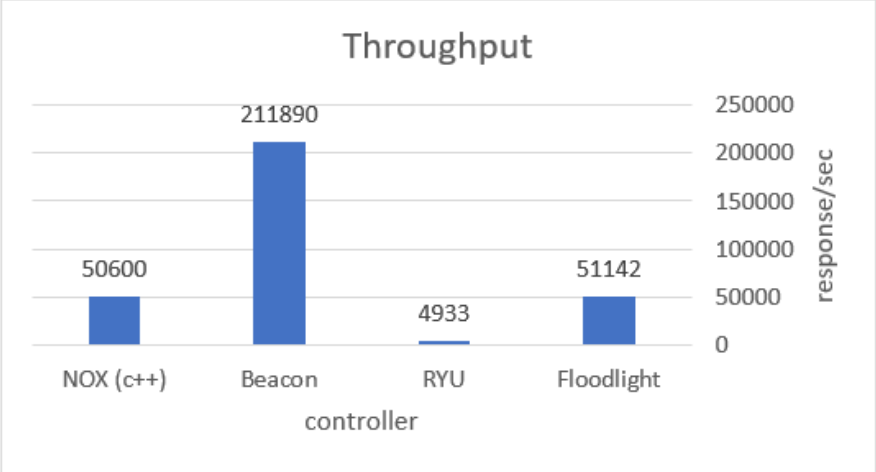
تم تنفيذ عدة سيناريوهات لقياس الإنتاجية

السيناريو الأول: قمنا في البداية بتوليد شبكة تحوي مبدل واحد و1000 مضيف متصل بالمبدل باستخدام الأداة CBench وذلك لغرض توليد عدد ملموس من الرزم ومن ثم تطبيق تأخير بسيط بمقدار 200 ميلي ثانية لنضمن أن المتحكم قد أتم عملية تأسيس الاتصال مع المبدل وذلك كي لا يتم إهمال أية رزم في البداية. ثم قمنا بتشغيل الأداة على العنوان IP والمنفذ الخاص بالمتحكم وإعادة السيناريو 10 مرات.

cbench -c 127.0.0.1 -p 6653 -l 10 -s 1 -M 1000 -D 200

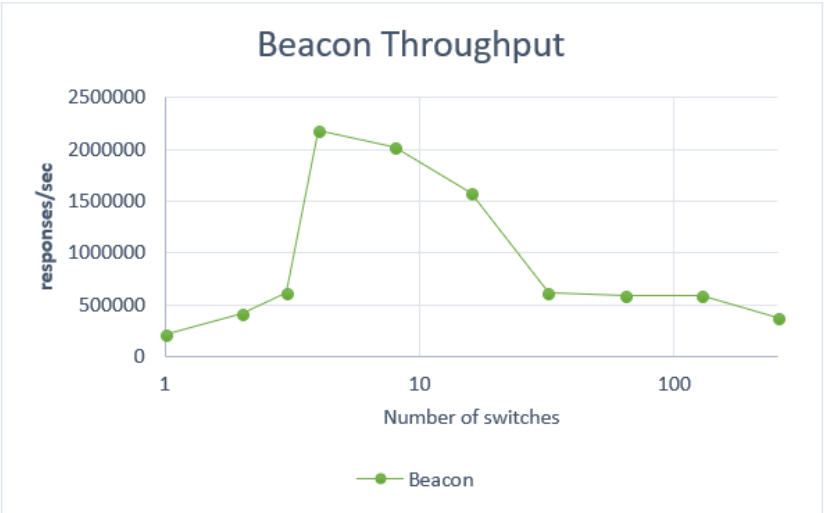
حيث c: عنوان المتحكم، p: المنفذ الخاص بالمتحكم، s عدد المبدلات في الشبكة، M عدد المضيفين في الشبكة، D التأخير الزمني قبل البدء بإرسال رزم packet_in. و l: عدد مرات إعادة الاختبار

أظهرت النتائج – الشكل (7) – تفوق المتحكم Beacon في الإنتاجية بمعدل 211890 عملية معالجة في الثانية في حين كان RYU هو الأقل إنتاجية

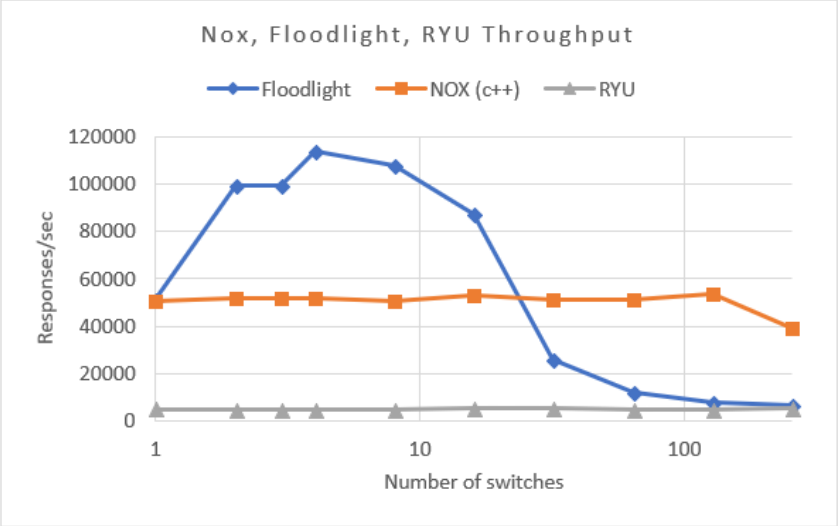


الشكل (7) إنتاجية كل متحكم من أجل مبدل واحد و1000 مضيف

السيناريو الثاني: قمنا بزيادة عدد المبدلات بشكل تدريجي 1، 2، 3، 4، 8، 16، 32، 64 ... 256 مع الأخذ بعين الاعتبار نفس الشروط السابقة ومن ثم تشغيل الأداة وأخذ النتائج في كل مرة. أظهرت النتائج بأن إنتاجية المتحكم Beacon تزداد مع ازدياد عدد المبدلات ليبلغ أقصى إنتاجية له بمعدل أكبر من 2 مليون عملية في الثانية عندما يكون عدد المبدلات يساوي 4 أي مساوياً لعدد الأنوية cores للجهاز الذي يعمل عيه المتحكم. لتبدأ بعدها الإنتاجية بالانخفاض بشكل تدريجي مع ازدياد عدد المبدلات لتصبح أقل من 400 ألف عملية في الثانية عندما يصبح عدد المبدلات كبيراً جداً في الشبكة. أما بالنسبة للمتحكم NOX و RYU فقد بقيت إنتاجية كل منهما ثابتة تقريباً مهما كان عدد المبدلات في الشبكة، أي أن زيادة عدد المبدلات لا يؤثر في الإنتاجية. وأخيراً كان سلوك المتحكم Floodlight مشابهاً لسلوك Beacon ولكن بإنتاجية أقل بكثير منها في Beacon. كما يُظهر الشكلين (8) و (9)



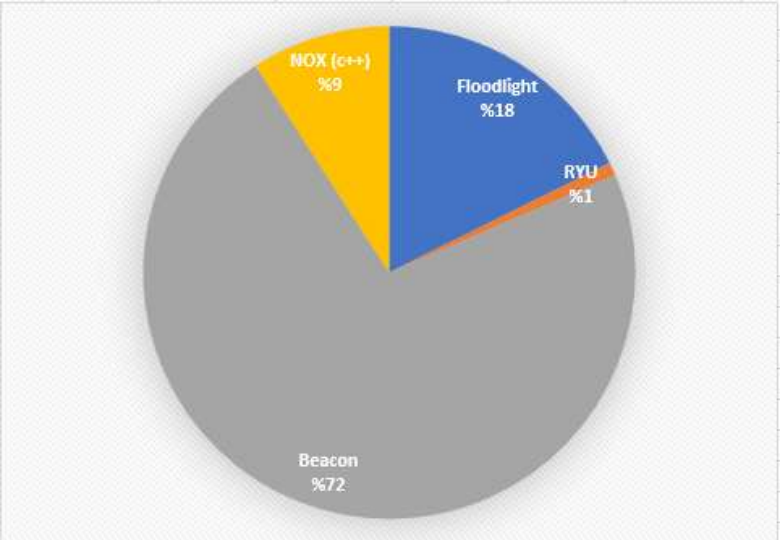
الشكل (8) إنتاجية المتحكم Beacon بازدياد عدد المبدلات



الشكل (9) إنتاجية المتحكمات NOX، RYU، Floodlight بازدياد عدد المبدلات

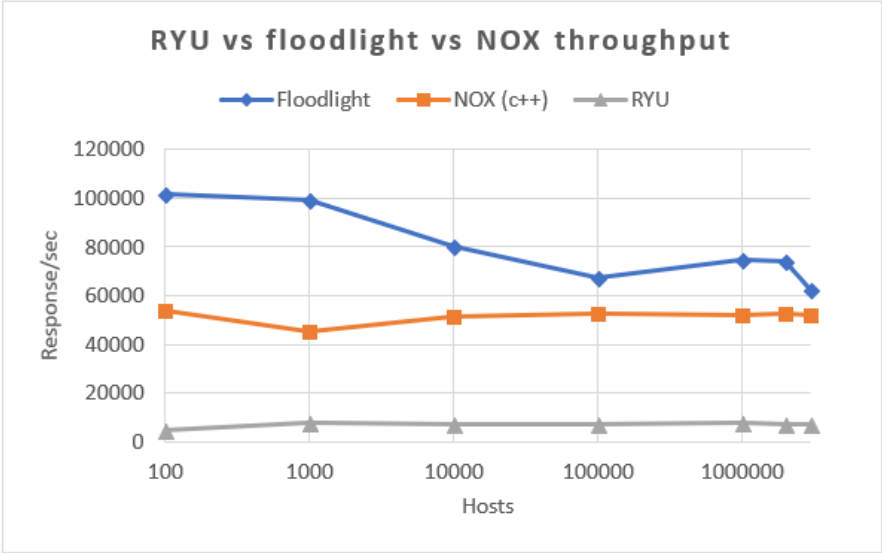
إنّ سبب الأداء العالي للمتحكم Beacon هو دعمه لتعدد النياسب واستخدامه لآلية الرتل المشترك في معالجة رزم packet_in الواردة إليه. حيث أن استخدام خوارزمية Round-Robin يعمل على زيادة كفاءة استغلال النياسب العاملة حيث لا يكون هناك أي نيسب في حالة idle. ويكون عدد النياسب العاملة أعظمي عندما يكون عدد مبدلات الشبكة 4 مساوياً لعدد أنوية معالج الجهاز الذي يعمل عليه المتحكم. على العكس من ذلك وعلى الرغم من أن المتحكم Floodlight يدعم تعدد النياسب إلا أنه يستخدم المكتبة Netty [12] في إدارة عمليات الادخال والإخراج I/O حيث أن كل رزمة packet_in سيتم معالجته باستخدام نيسب واحد، بعدها ستمر الرزمة عبر سلسلة وحدات Modules تقوم كل منها بإجراء عمليات معينة عليه. من وجهة نظر المكتبة Netty فإن كل مبدل سيتم مقابلته بنسيب واحد. ولكن مع ورود عدد كبير من الرزم وبسبب وجود عمليات متعددة على الرزمة عندها يتوجب على الرُزم القادمة انتظار النيسب الذي خُصص لها في رتل انتظار، في هذه الحالة قد يكون هناك نياسب في حالة idle في حين توجد رزم قيد الانتظار وهذا ما يخفض من إنتاجية المتحكم. أما بالنسبة للمتحكمين RYU وNOX فإن زيادة عدد المبدلات في الشبكة لن يؤثر في أداء أي منهما وذلك لعدم دعمها للنياسب المتعددة في عملهما، هذا ما يحد من الأداء.

يبين الشكل التالي النسبة المئوية للإنتاجية من أجل مبدل واحد

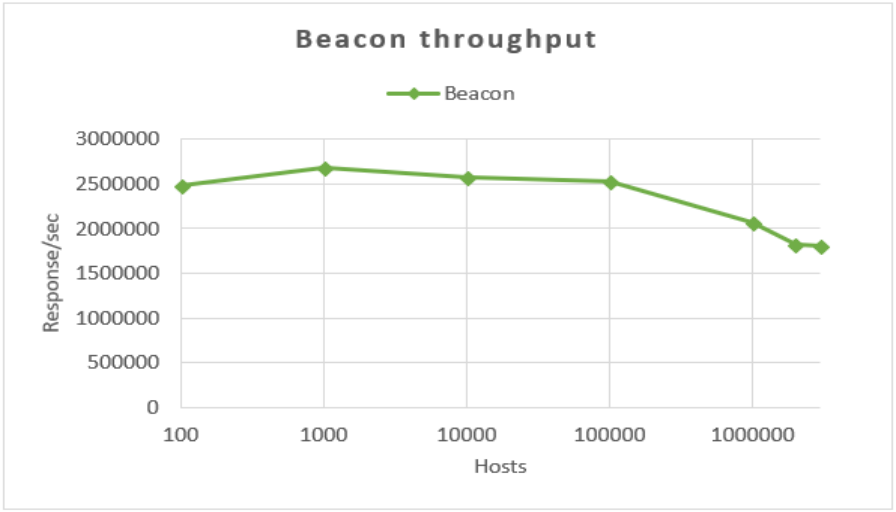


الشكل (10) النسبة المئوية لإنتاجية المتحكمات من أجل مبدل واحد

في السيناريو الثالث: قمنا بتثبيت عدد المبدلات على 4 وزيادة عدد المضيفين ومن ثم تطبيق الأداة CBench وكانت النتائج كما في الشكلين التاليين:



الشكل (11) إنتاجية المتحكمات NOX, RYU, Floodlight بازدياد عدد المضيفين

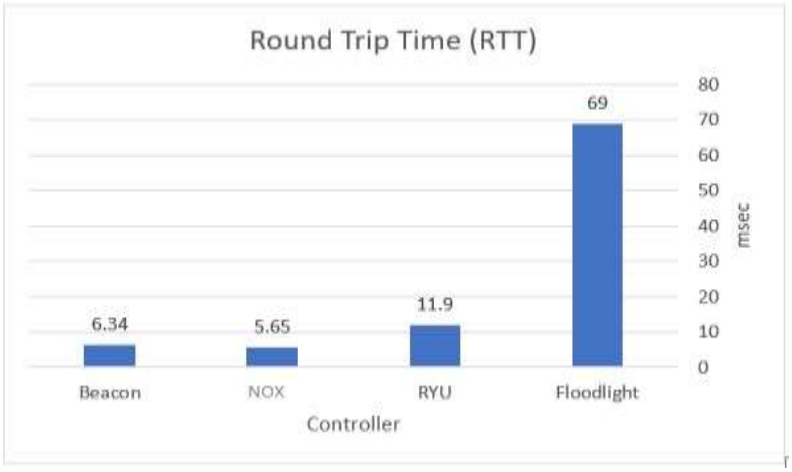


الشكل (12) إنتاجية المتحكم Beacon بازدياد عدد المضيفين

نلاحظ بأن تَغيُّر الإنتاجية كان طفيفاً في كل المتحكمات ما عدا المتحكم Floodlight الذي انخفضت انتاجيته بحدود 40 بالمئة عندما ازداد عدد المضيفين بشكل كبير جداً في الشبكة وذلك بسبب تجاهل رزم عديدة بسبب الآلية الضعيفة في توزيع الطرود على النياسب العاملة ووجود زمن إضافي أثناء معالجة كل طرد.

بالنسبة للبارامتر الثاني وهو زمن **الرحلة الانكفائية RTT** فقد تم بناء شبكة مكونة من مبدل ومضيفين اثنين باستخدام المحاكي mininet، وهي أفضل حالة لقياس الـ RTT حيث لا يوجد سوى مسار واحد للرزمة وبالتالي احتمالية وجود تأخير على المسار مهمل، ثم قياس الزمن اللازم لإرسال رزمة ICMP من أحد المضيفين واستقبال الرد لها وذلك من خلال تحليل عملية الارسال باستخدام برنامج Wireshark. بفرض أن جدول التوجيه الخاص بالمبدل فارغ. عندها سيقوم المبدل بتمرير الرزمة إلى المتحكم كرسالة Packet_in ليقرر المتحكم وجهتها ويُخبر المبدل بذلك ومن ثم يقوم المبدل بتمرير الرزمة مجدداً.

يظهر الشكل (13) نتائج قياس زمن الرحلة الانكفائية للرزمة في المبدلات المدروسة.

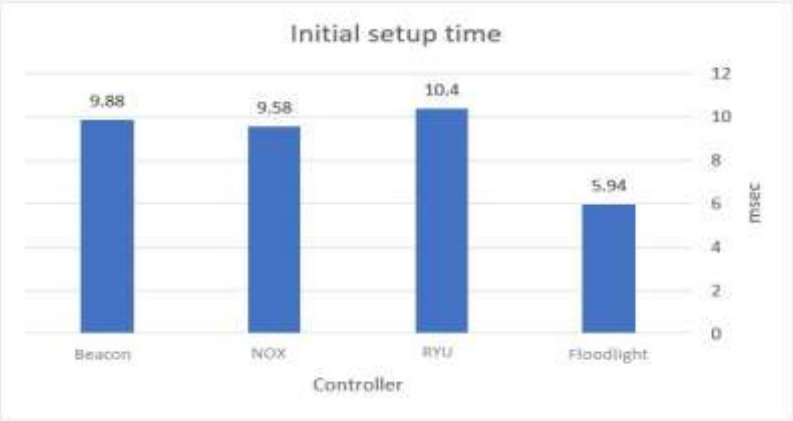


الشكل (13) زمن الرحلة الانكفائية

تُظهر النتائج بأن الزمن باستخدام المتحكم NOX أقل منه في بقية المتحكمات، ويعود السبب في ذلك إلى لغة البرمجة التي تم تطوير المتحكم بها. حيث أن مفسر لغة C++ هو الأسرع، وبالتالي سرعة في التنفيذ، أما الزمن الكبير للمتحكم Floodlight فيعود إلى سلسلة العمليات التي تتم على الطرد عند استقباله في المتحكم.

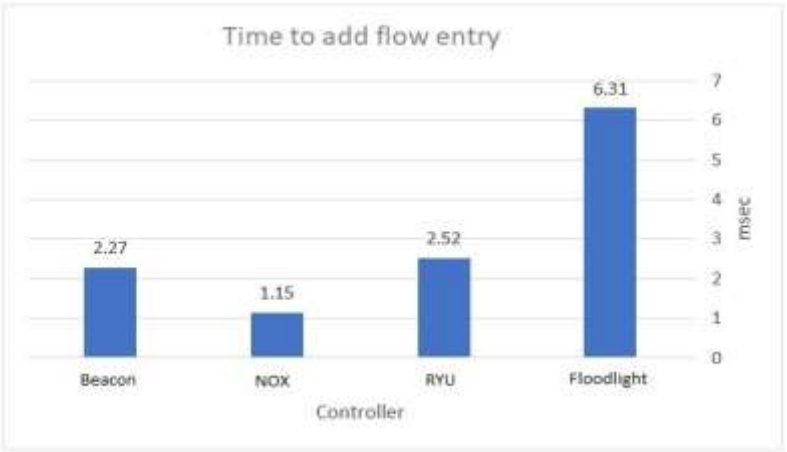
ومن أجل قياس البارامتر الثالث وهو زمن **تأسيس الاتصال بين المبدل والمتحكم** وباستخدام نفس الشبكة السابقة تم تتبع وتحليل عملية تأسيس الاتصال بين المتحكم والمبدل باستخدام Wireshark وحساب الزمن من لحظة تأسيس اتصال TCP بين المتحكم والمبدل وحتى انتهاء عملية التهيئة.

أظهرت النتائج بأن المتحكم Floodlight تطلب زمناً أقل لتأسيس الاتصال مع المبدل حوالي 6 ميلي ثانية في حين تطلَّب المتحكم RYU أعلى زمن لتأسيس الاتصال. يعود السبب في ذلك إلى أن حجم رزم تأسيس الاتصال التي أرسالها المتحكم Floodlight كان أقل منه في المتحكمات الأخرى. يُظهر الشكل (14) تلك النتائج.



الشكل (14) زمن تأسيس اتصال المتحكم مع المبدل في الشبكة

أخيراً قمنا أيضاً باستخدام نفس الشبكة السابقة المكونة من مبدل ومضيفين بقياس الزمن الذي يأخذه كل متحكم لُيضيف مدخل إلى جدول تدفق المبدل Flow table حيث بفرض أن المبدل قد قام بإرسال طلب ARP إلى المتحكم لمعرفة عناوين المضيفين، وبتتبع وتحليل عملية إضافة مدخل إلى جدول التوجيه باستخدام Wireshark حصلنا على النتائج في الشكل (15)



الشكل (15) زمن إضافة مدخل إلى جدول التدفق

تُشير النتائج إلى أن المتحكم NOX تطلب أقل زمن لإضافة مدخل إلى جدول التدفق وهنا أيضاً يأتي دور لغة البرمجة التي طُوّر بها المتحكم، وهذا ما يؤكد على صحة ما تم التوصل له فيما يخص زمن الرحلة الانكفائية من حيث الزمن المُستهلك في كل متحكم.

الاستنتاجات والتوصيات:

قُمنا في هذا البحثِ بتقييمِ أداءِ أربَعةِ متحكماتٍ NOX, RYU, Floodlight, Beacon من خلالِ أربعةِ بارامترات مختلفة، الإنتاجية – زمن الرحلة الانكفائية – الزمن اللازم لتأسيس الاتصال بين المتحكم والمبدل وأخيراً الزمن اللازم لإضافة مدخل إلى جدول تدفق المبدل.

بناءً على النتائج التي تم التوصل إليها في البحث يمكن استنتاج ما يلي:

- بالنسبة للإنتاجية يزداد أداء المتحكمات التي تدعم تعدد النياسب مع زيادة عدد المبدلات في الشبكة حتى يصبح مساوياً لعدد أنوية المعالج الذي يعمل عليه المتحكم، في حين أن أداء المتحكمات التي لا تدعم تعدد النياسب زيادة عدد المبدلات لا يؤثر في انتاجيتها. حيث لاحظنا تفوق المتحكم Beacon في جميع السيناريوهات التي تم تطبيقها وقد بلغ أقصى أداء له عندما أصبح عدد المبدلات مساوياً لعدد الأنوية للجهاز الذي يعمل عليه. في حين بقي أداء RYU وNOX ثابتاً في كل الحالات.
- لم يكن لزيادة عدد المضيفين في الشبكة تأثير كبير في الإنتاجية إلا في حالة المتحكم Floodlight الذي قام بإهدار عدد كبير من الرزم الواردة إليه بسبب الآلية المستخدمة في استقبالها ومعالجتها.
- تلعب لغة البرمجة التي طُوِّر بها المتحكم دوراً كبيراً في سرعة تنفيذ العمليات المختلفة التي يقوم بها المتحكم وقد لاحظنا ذلك عند قياس زمن الرحلة الانكفائية وزمن إضافة مدخل إلى جدول التدفق الخاص بالمبدل، حيث أن مفسر لغة c++ في حالة المتحكم NOX هو الأسرع وهذا ما ينعكس ايجابياً على المتحكم.

نوصي بناءً على ما سبق باستخدام المتحكم Beacon في حال كان التطبيق يتطلب معدل إرسال بياناتٍ عالٍ جداً في الشبكة، أما في حال كانت الشبكة تحوي عدد كبير جداً من المبدلات وتتطلب معدل إرسال بياناتٍ متوسط فمن الأفضل استخدام المتحكم Floodlight فبذلك يكون الزمن الكلي اللازم لتأسيس الاتصال مع مبدلات الشبكة أقل بكثير منه باستخدام المتحكمات الأُخرى. أما التطبيقات التي تتطلب سرعة في التنفيذ كتطبيقات الزمن الحقيقي والتي لا تتطلب معدل ارسال بيانات عالٍ فنوصي باستخدام المتحكم NOX لأن عامل الزمن هو الأهم.

المراجع:

[1] M. CASADO, T. KOPONEN, D. MOON, and S. SHENKER. *Rethinking Packet Forwarding Hardware*. Seventh ACM Workshop. On HotNets-VII, 2008.

[2] ONF,–*SDN Architecture Overview*. Version 1.0, URL:https://www.opennetworking.org/ images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview 1.0. pdf, 2013.

[3] MCKEOWN, N. *Openflow: enabling innovation in campus networks*. Commun. SIGCOMM Comput, Rev., 38(2), PP :69–74, April 2008.

[4] Open Networking Foundation, https://www.opennetworking.org/wp- content/uploads /2013/04/openflow-spec-v1.0.0.pdf, 2013.

[5] MCKEOWN,N, - *"OpenFlow and Software Defined Networks"*. NSF, Stanford Clean Slate Program, URL: <https://www.nanog.org/meetings/nanog50/presentations/ Monday/NANOG50.Talk42.McKeown-100410-1541.pdf >. Pages:1-39, 2011.
[6] M. MONACO, O. MICHEL and E. KELLER, *Applying Operating System Principles to SDN Controller Design*. Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, Article No. 2, 2013.
[7] NATASHA, G. **"***NOX: towards an operating system for networks"*. SIGCOMM CCR, Volume 38 Issue 3. Pages: 105-110, 2008.
[8] Khan, S., Gani, A., and Abdul Wahab, A. *"Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art*". IEEE Comun. Vol 19. Pages: 303 - 324, 2017.
[9] NOX. Retrieved from <https://github.com/noxrepo/> September 15, 2018
[10] Ryu. Retrieved from <http://osrg.github.com/ryu >. last visited at August 13, 2018.
[11] Erickson, David. **"***The beacon openflow controller*", HotSDN '13, In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, Pages 13-18, 2013.
[12] Floodlight project, available at: <http://floodlight.atlassian.net>
[13] http://mininet.org, last visited at June 15, 2018.
[14] CBench, available at: <https://github.com/mininet/oflops/tree/master/cbench>