**The Beacon OpenFlow Controller**

**Beacon** – מבוסס JAVA, נוצר ב-2010, מתפקד כמנהל OpenFlow בגישת open source.

נעשה בו שימוש רחב למטרת לימוד, מחקר ובסיס למוצרי תאורה Floodlight.

**המטרות של Beacon:**

1. **שיפור היצרנות (הפריון) של המפתח.**
2. **לספק את היכולת לפתוח אפליקציות חדשות ולעצור אפליקציות רצות, בזמן ריצה.**
3. **להיות בעל ביצועים טובים.**
4. **הקדמה**

**מנהל התקשורת הראשון בגישת open source שקדם ל-Beacon הוא NOX.  
NOX מאפשרת פיתוח בשפת Python שהיא נוחה וידידותית יותר עבור פיתוח אפליקציות רשת או בשפת C++ שהיא בעלת ביצועים טובים יותר.**

**NOX מציבה עבור המפתחים בה את ההפרדה בין נוחות לאיכות.**

**שאלות שנתעסק בהן:**

1. במנהל ה- OpenFlow NOX ישנה הפרדה בין כתיבת אפליקציות רשת בשפה ידידותית, נוחה וקלה כמו Python לבין כתיבה בשפה יעילה יותר מבחינת ביצועים (C, C++) – כלומר אפליקציות באיכות גבוהה.

**האם מנהל OpenFlow יכול לקבל את שניהם?**

**להיכתב בשפה ידידותית, נוחה וקלה מחד ולהיות בעל ביצועים טוב מאידך?**

1. **אם מנהל OpenFlow דומה במהותו למערכת הפעלת תקשורת (Network Operation System) האם הוא צריך להיות בעל יכולת לפתיחת וסגירת אפליקציות בזמן ריצה?**

שפת הפיתוח וסביבת הפיתוח חשובות מאוד ומהוות **חלק ניכר** מרמת היצרנות של המפתח ובאיכות המוצר שייצר.

ישנן המון שפות פיתוח ידידותיות אך טיב הביצוע שלהן בכל מה קשור לניהול OpenFlow לא היה ידוע בתקופת NOX.

1. **עבודות דומות:**

**NOX – קצת רקע**

פלטפורמת מנהל ה-OpenFlow הראשון של NOX יצא ב-2008 עבד בשיטת ביצוע שיתופי בין תהליכונים דרך תהליך אחד. ב2011 יצאה עוד גירסא של NOX שמשתמת בריבוי תהליכים והחלפת אפליקציות. הרבה קונטרולרים יצאו מאז שיצא NOX. כולם התחלקו בערך ל2 קטגוריות: open source single-instance Controller and commercial closed source distributed controllers.

הקונטרולרים עם המאגר הפתוח שימשו למחקר ופיתוח. ההבדל ביניהם היה השפה שבה הם היו כתובים.

פה יש רשימה של קונטרולרים שנכתבו והשפות שבהן כתבו אותם.

1. **יעילות פיתוח**

**3.1 שפות פיתוח**

C וC++ הן שפות שאולי כדאי לעבוד איתם. אפשר לפתח איתן אפליקציות עם ביצועים טובים. אבל.. הן באת גם עם כמה חסרונות:

**חסרונות של C ו- C++**

* זמן קומפילציה **מלאה** ארוך מאוד (יותר מ-10 דקות)
* שגיאות קומפילציה לא מובהקות, לא קל להבין מה יוצר את הבעיה.
* הקצאות ושחרור ידני של זכרון, יוצר קריסות עקב עומס על זכרון שמנוהל לא טוב או דליפות זכרון (זכרון שלא שוחרר כראוי ותופס ללא שימוש).

יש גישות שונות שטוענות שאפשר להנמיך את החסרונות שלהן: לא לקמפל רכיבים שוליים כל פעם שמקמפלים יוריד את זמן הקומפילציה הארוך, וגם שימוש בטכניקות חכמות כמו למשלמצביעים חכמים למקומות בזיכרון לשיפור שגיאות הזיכרון.

גישות אלא לא מושלמות, כך מעידים המפתחים. בחירת C ו++C לסביבות פיתוח הוא ההחלטה הנכונה, אולם, מאמר זה חוקר האם ניתן להשתמש בשפות "חברותיות" יותר לפיתוח open flow controller עם ביצועים טובים יותר על חומרה שבשוק. (כאשר מעבדים וזיכרון אפשר לדרג בקלות)

3 היבטים של שפות תיכנות נבחנות כשאר אנו באים לבחור שפת תיכנות: ניהול זיכרון, יכול לרוץ על פלטפורמות שונות, ביצועים טובים.

ניהול אוטומטי של זיכרון (garbage collector) יכול למנוע את רוב שגיאות הזיכרון. שפות עם יכולת זו אין להן (כמעט) זמן המתנה של קומפילציה. (זמן קומפילציה ארוך). שפות אלה מספקות דיווח מדויק במקרה של נפילה, ממש שורה מדויקת שבה קרתה השגיאה. השפות המועמדות להיבחר היו C# Python Java. אותן המחבר הכיר במיוחד. מערכת ההפעלה שמעליה היה אמור לרוץ הקונטרולר היה linux. אבל הייתי משוכנע שזה יכול לרוץ גם על MAC\_OS או על WINDOWS בלי בעיות.

כל השפות הללו היו יכולות לרוץ גם המערכות הפעלה כאלה, אבל למתורגמן הרישמי של C# היה חסר תרגום למערכות הפעלה שהן לא WINDOWS. לכן C# ירד מהפרק.

ביצועים טובים זה עניין סובייקטיבי. היה מחסור בתרגים רישמי לריבוי תהליכים בPython, ולכן גם הוא היה מועמד מוטל בספק. אז עכשיו נשארנו עם JAVA. כאשר אנחנו לא יודעים בוודאות מהי רמת הביצועים שלה. תוכנות אחרות שנכתבו בJAVA היו בעלות ביצועים טובים ולכן הנחתי שJAVA הוא המועמד הטוב ביותר לכתיבת הController.

חלק 6 במאמר זה עוסק בביצועים של beacon.

* 1. **סיפריות**:

Beacon עצמו ממנף את מספר הסיפריות, תוך ניסיון להשתמש בקוד קיים כמה שיותר, ולהקל על פתוח הקונטרוללר עצמו והאפליקציות שעליו. הספריה המשמעותית ביותר היא Spring.

שני רכיבים עיקריים של Spring משומשים בBeacon : The Inversion of Controller container and the Web framework.

--------השאר בחלק זה לא היה מעניין-----

* 1. API:

הAPI של Beacon עוצב להיות פשוט ולא מטיל שום מגבלות, המפתחים יכולים להשתמש באיזה רכיב של JAVA שרוצים. (תהליכונים, טיימר, סוקטים ועוד)

הקונטרולר מתקשר על הswitches בצורה של events. הוא משתמש בתבנית התיכול הצופה Observer כאשר האוגרים נרשמים לקבל events שרלוונטים אליהם.

Beacon משתמש בספריית OpenFlowJ, שזוהי ספרייה מונחת עצמים של JAVA שמממשת OpenFlow V1.

התקשורת עם הswitches נעשית דרך IBeaconProvider (ממשק). מאזינים שונים "נרשמים" לקבל הודעות כאשר קורה אירוע כלשהו. (פירוט סוגי מאזינים- במאמר).

בנוסף ישנם אפליקציות שנמצאות שם Build in, שמוסיפות גם כן לAPI של Beacon:

* Device Manager: מנהלת את כל הקשור לDevices שמצטרפים, עוזבים או מתעדכנים. מודעת לכל הכתובות שלהן (MAC וIP) מתי נראו לאחרונה. וכו... בקיצור,מנהלת את כל המחשבים של הרשת. (כולל לאיזה Switches הם מחוברים, באיזה פורטים וכו..)
* Topology: מגלה קשרים חדשים בין Switches. מקבלת התראות כשאר לינקים נוספים או נעלמים.
* Routing: מציעה מסלול קצר ביותר בין כל 2 מחשבים ברשת.
* Web: מספק שירותי WEB.