

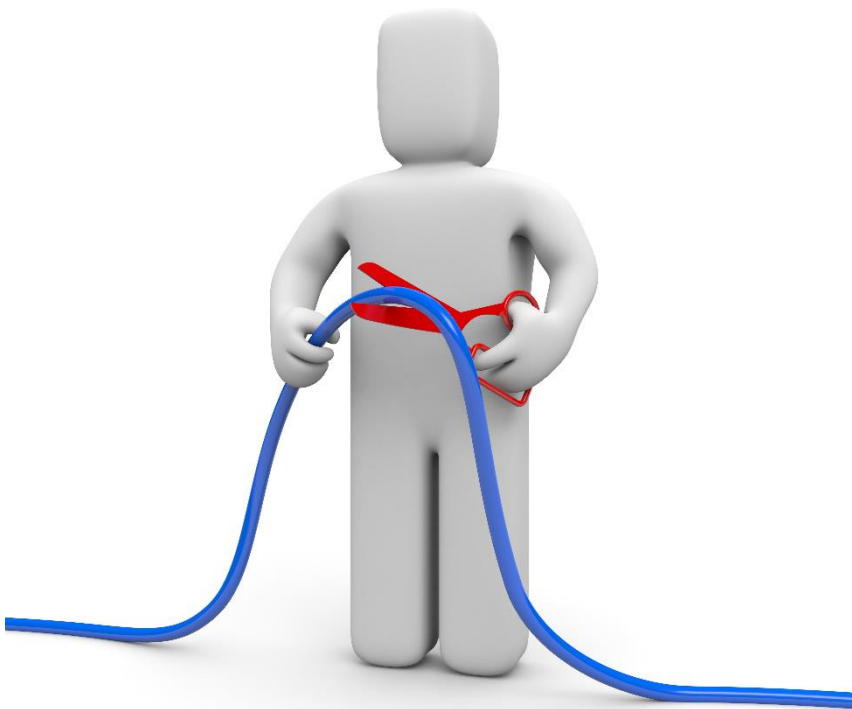
Implementing Link Aggregation on Silicom Directory Card

מגישים:

נגה ליבהבר
אלכס אייזנשטט

מנחה:

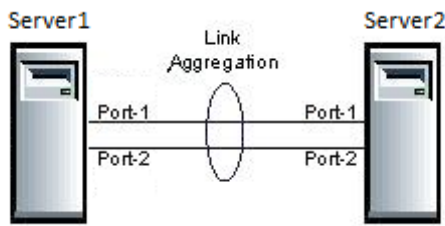
רועי מטרני



תוכן עניינים

3.....	תקציר
4.....	DPDK (Data Plane Development Kit)
6.....	Poll Mode Driver (PMD)
7.....	Link Bonding Poll Mode Driver Library
11.....	מדריך התקנה של DPDK
14.....	Pktgen
17.....	סכמת המערכת
18.....	מימוש הפרויקט
27.....	סטטיסטיקות
33.....	מסקנות
34.....	ביבליוגרפיה

תקציר



Link aggregation זהו שם נפוץ לתקן IEEE 802.3ad המאפשר איחוד של מספר פורטים פיזיים לקבוצה אחת (LAG – Link Aggregation Group) והפיכתם לפורט לוגי יחיד.

Link aggregation נותן מענה ל-2 בעיות מרכזיות בתעבורת רשת: רוחב פס מוגבל וכושר התאוששות נמוך במקרה של נפילה.

רוחב הפס מוגבל למהירות התעבורה של פורט בודד. Link aggregation מאפשר לחבר את מהירויות התעבורה של כל הפורטים שנמצאים ב-LAG, ובכך להגדיל את רוחב הפס.

המענה לבעיה השנייה בא לידי ביטוי בחלוקת עומסים שעושה link aggregation בין הממשקים הפיזיים. במקרה של נפילת פורט פיזי או כבל המחבר בין 2 פורטים, הפורטים האחרים ב-LAG יחלקו ביניהם את העומס של הפורט שנפל והרמות העליונות במודל ה-OSI לא ירגישו את הנפילה ובכך קטן באופן משמעותי זמן ההתאוששות במקרה של תקלה.

Link aggregation ניתן לממש בכל אחת משלושת השכבות התחתונות של מודל ה-OSI:

- בשכבה הפיזית ניתן להשתמש בהתקני רשת המשלבים תדרים מרובים.
- בשכבת הקו ניתן לאחד מספר פורטים פיזיים או וירטואליים של המתג וליצור מהם פורט לוגי יחיד. הפורטים הפיזיים חולקים אותה כתובת MAC לוגית.
- בשכבת הרשת ניתן להשתמש בתזמון מסוג round-robin. הממשקים הפיזיים חולקים אותה כתובת IP לוגית.

מטרת הפרויקט היא ליישם link aggregation בין מספר פורטים פיזיים ליצירת פורט לוגי יחיד, ע"י שימוש בפלטפורמה חדשה הנקראת DPDK.

DPDK (Data Plane Development Kit)

סקירה כללית

DPDK זהו שם המתאר אוסף של ספריות ודרייברים לעיבוד מהיר של חבילות תקשורת. הוא מספק framework לתכנתנים עבור מעבדי Intel x86, IBM Power 8, EZchip TILE-Gx, ARM, המאפשר פיתוח מהיר של אפליקציות רשת המבוססות על חבילות מידע המועברות במהירויות גבוהות.

EAL (Environment Abstraction Layer)

ה-framework של DPDK מספק אוסף של ספריות לסביבות חומרה/תוכנה ספציפיות ע"י יצירת EAL, שכבת אבסטרקציה מעל כרטיסי הרשת. EAL מסתיר בתוכו את ההגדרות הספציפיות של סביבת העבודה ומספק ממשק סטנדרטי עבור המתכנת, אשר מכיל ספריות, מאיצי חומרה זמינים ואלמנטים אחרים של החומרה ושל מערכת ההפעלה. ברגע ש-EAL נוצר, המתכנתים משתמשים בספריות המסופקות ליצירת אפליקציות. באמצעות תהליכון של EAL מוגדרת ליבה לוגית (lcore), אשר משמשת כיחידת ביצוע לוגית. למעשה ליבה לוגית היא pthread של לינוקס. היצירה והניהול שלהן מתבצעים באמצעות ה-EAL. בדרך כלל, ליבה לוגית מתקשרת ל-CPU פיזי וה-ID של הליבה הלוגית זהה ל-ID של ה-CPU אליה היא מקושרת.

Data plane

Data plane מגדיר את החלק בארכיטקטורה של המתג שמחליט מה לעשות עם החבילות הנכנסות. על מנת להגיע לביצועים גבוהים בביצוע data plane, DPDK מיישם את העקרונות הבאים:

1. מודל לתקורה נמוכה בשם run-to-completion. run-to-completion הינו מנגנון תזמון שבו כל משימה רצה עד שהיא מסתיימת או שמחזירה באופן מפורש את השליטה למנגנון התזמון.
2. ה-DPDK ניגש להתקנים באמצעות מנגנון ה-polling, כלומר, כאשר המחשב מחכה לקבל קלט מההתקן, הוא מתשאל אותו כל הזמן. זאת במקום השימוש במנגנון הפסיקות, אשר מגדיל את התקורה.
3. הקצאת poll של אובייקטים במרחב הזיכרון מחולק ל-hugepages. בכך ישנה גישה גדולה ורציפה לזיכרון, והקטנה של טבלת ה-TLB.
4. שימוש במנגנון ה-pipeline אשר מקטין את שלבי העבודה ומאפשר להעלות את תדר הביצוע.

רכיבי ליבה של DPDK

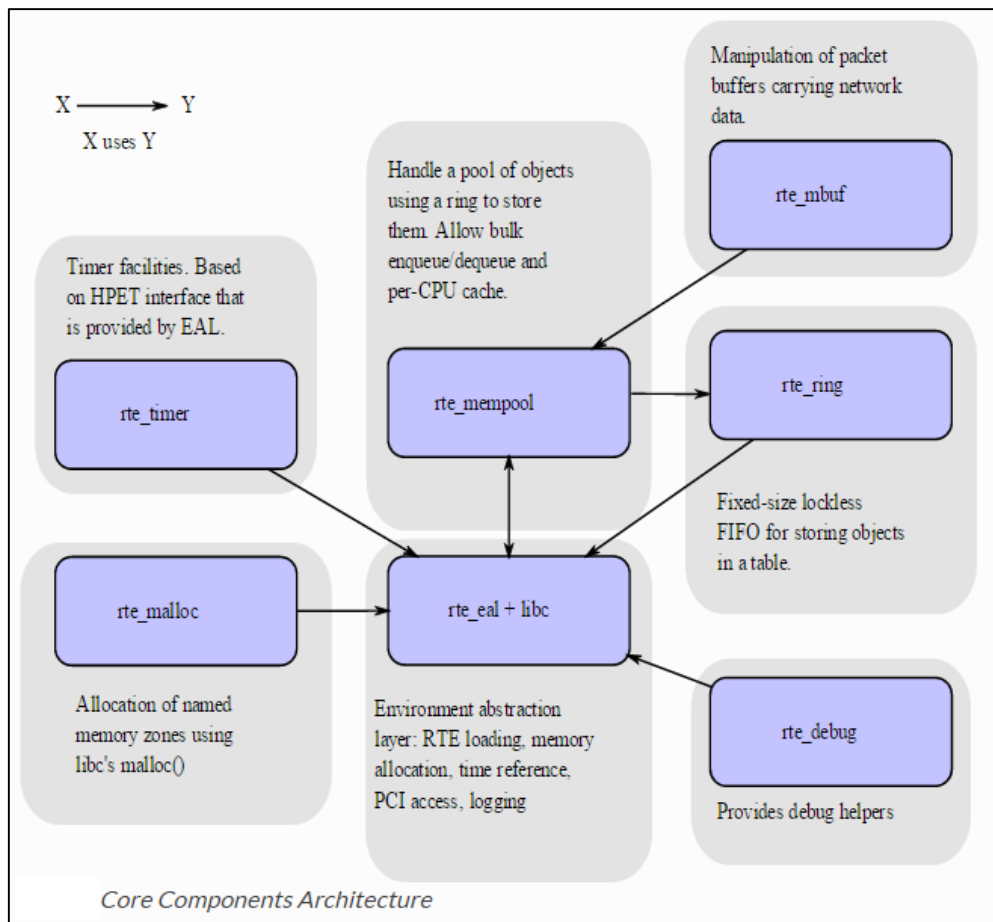
DPDK מכיל ספריות הקשורות ל-data plane ואופטימיזציה לדרייברים של כרטיסי רשת. הספריות המרכזיות הן:
Ring Manager – מבנה הנתונים של טבעת מספק FIFO API חסר מנעולים עבור מספר רב של יצרנים וצרכנים בטבלה בגודל סופי. למבנה נתונים זה יש מספר יתרונות על פני תור חסר מנעולים: קל יותר למימוש, מותאם לכמות גדולה של פעולות ומהיר יותר.
Memory Pool Manager – אחראי על הקצאת מספר אובייקטים בזיכרון.

Network Packet Buffer Management – אלה ספריות המספקות מנגנון אשר מאפשר ליצור ולהרוס חוצצים ש-DPDK משתמש בהם על מנת לאחסן הודעות.

Timer Manager – ספרייה זו מספקת טיימר עבור יחידת הביצוע של ה-DPDK, המאפשרת הרצה של פונקציות בצורה אסינכרונית.

Poll Mode Drivers (PMD) – מורכב ממספר API אשר מאפשרים להגדיר התקנים ואת התורים שלהם. בנוסף PMD יכול לגשת ישירות למזהים של RX ו-TX ללא פסיקות, על מנת לקבל במהירות, לעבד ולהעביר את החבילות בתוך אפליקציית המשתמש.

Packet Forwarding Algorithm Support – DPDK מכיל ספריות ערבול ו-Longest Prefix Match כדי לתמוך באלגוריתמי העברת חבילות.



Poll Mode Driver (PMD)

כפי שנאמר מקודם, PMD הוא אחד מרכיבי הליבה של ה-DPDK והוא מורכב ממספר API'ים אשר מאפשרים להגדיר את ההתקנים ואת התורים שלהם.

דרישות והנחות

סביבת ה-DPDK מאפשרת לאפליקציות שמבצעות עיבוד לחבילות תקשורת לעבוד ב-2 מודלים:

1. **run-to-completion** – במודל זה מזהה RX של פורט מסויים מושך חבילות באמצעות ה-API החבילות מעובדות על אותה הליבה ומועברות למזהה ה-TX של הפורט באמצעות ה-API כדי לבצע שליחה.

2. **pipe-line** – במודל זה ליבה אחת מושכת חבילות למזהה RX של פורט אחד או יותר תוך שימוש ב-API החבילות מתקבלות ומועברות לליבה אחרת. הליבה השנייה ממשיכה לבצע עיבוד על החבילות ולאחר מכן היא יכולה להעבירן למזהה TX של הפורט באמצעות ה-API כדי לבצע שליחה.

במודל סינכרוני של run-to-completion כל ליבה לוגית מוקצת ל-DPDK ומבצעת בלולאה את העיבוד של החבילות. העיבוד כולל את השלבים הבאים:

- קבלת החבילות הנכנסות באמצעות API של ה-PMD שאחראי על הקבלה.
- ביצוע עיבוד על החבילות שהתקבלו, חבילה אחת בכל פעם, עד להעברתה.
- שליחת החבילות שממתינות לשליחה באמצעות API של ה-PMD שאחראי על השליחה.

במודל האסינכרוני של pipe-line חלק מהליבות הלוגיות יכולות להיות מוקצות לקבלה של החבילות הנכנסות, ושאר הליבות הלוגיות יכולות להיות מוקצות לעיבוד על החבילות שהתקבלו. החבילות שהתקבלו מועברות בין הליבות הלוגיות באמצעות מנגנון הטבעת. הלולאה שבה מתקבלות החבילות מכילה את השלבים הבאים:

- קבלת החבילות הנכנסות באמצעות API של ה-PMD שאחראי על הקבלה.
- העברה של החבילות שהתקבלו לליבות הלוגיות שמבצעות את העיבוד עליהן באמצעות תור חבילות.

הלולאה שבה מתבצע עיבוד על החבילות מכילה את השלבים הבאים:

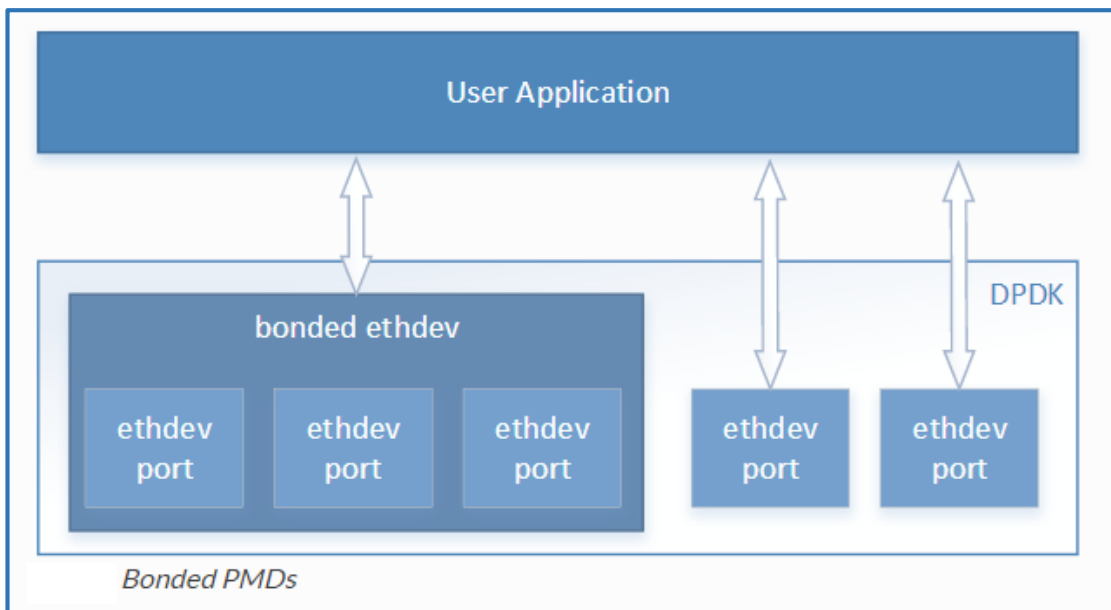
- שליפה של החבילות שהתקבלו מתור החבילות.
- ביצוע עיבוד על החבילות עד לשליחתן הלאה.

במטרה למנוע את התקורה מביצוע פסיקות לא נחוצות, סביבת הביצוע לא משתמשת במנגנון התראות אסינכרוני. אך כשהדבר הכרחי, התקשורת האסינכרונית מתבצעת בעזרת שימוש במנגנון הטבעת. בעבודה בסביבה של ריבוי ליבות, המפתח הוא הימנעות מהשימוש במנעולים. על מנת להשיג מטרה זו, PMD תוכנן לעבוד ככל האפשר עם משאבים מקומיים עבור כל ליבה. למשל, PMD מחזיק תור של חבילות נשלחות פר ליבה ופר פורט. כל תור קבלה של פורט מנוהל על ידי ליבה לוגית אחת.

Link Bonding Poll Mode Driver Library

בנוסף לדרייברים שמגיעים ב-Poll Mode Drivers (PMD) עבור חומרה פיזית ווירטואלית, DPDK מכיל ספריות תוכנה טהורות, אשר מאפשרות לחבר יחד מספר PMD'ים פיזיים וליצור מהם PMD לוגי יחיד.

הספרייה של Link Bonding PMD, בדומה לדרייבר של bonding הקיים ב-Linux, תומכת בחיבור של קבוצה של פורטים בעלי אותה מהירות ומצב עבודה (full-duplex / half-duplex) לפורט לוגי בודד. הפורט הלוגי החדש יתבסס על הפורטים הפיזיים ויספק תמיכה לתכונות כגון שרידות בקישוריות, עמידות בפני נפילות ואיזון עומסים.

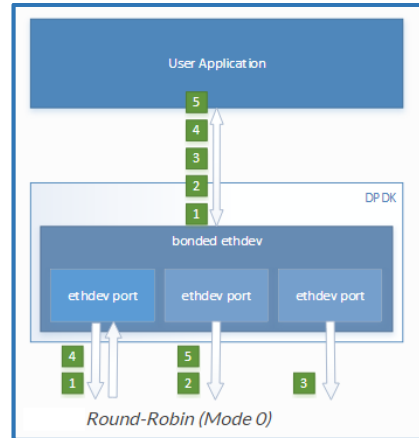


הספרייה חושפת API שכתוב בשפת C ומספק API ליצירה, קביעת תצורה וניהול של הפורטים המחוברים יחדיו לפורט לוגי אחד.

ישנם מספר מצבי עבודה שנתמכים ע"י ספריית Link Bonding PMD:

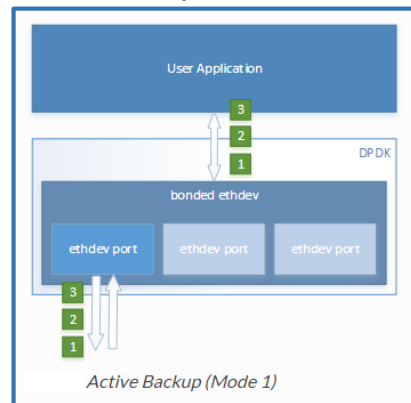
Round-Robin (מצב עבודה 0):

מצב עבודה זה מספק איזון עומסים ועמידות בפני נפילה על ידי שליחת החבילות בצורה סדרתית לפורט הפנוי הראשון בקבוצה, לפורט הפנוי השני בקבוצה וכך הלאה. החבילות מסודרות מחדש לתורים ונשלחות בתצורת round-robin. מצב עבודה זה לא מבטיח שקבלת החבילות תהיה באותו סדר כמו שליחתם, והצד המקבל צריך לדעת להתמודד עם הודעות שמגיעות לא לפי הסדר.



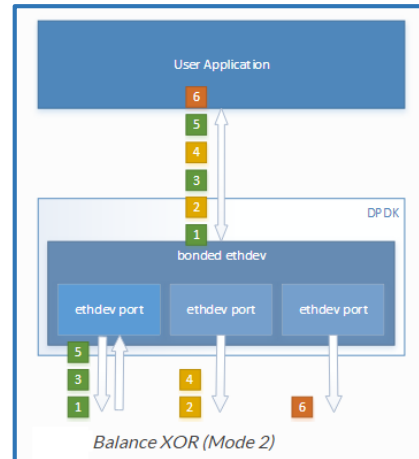
Active Backup (מצב עבודה 1):

במצב עבודה זה בכל זמן נתון רק פורט אחד פעיל (ומוגדר כפורט האקטיבי), ושאר הפורטים משמשים כגיבוי. הפורט שבגיבוי יחל להעביר תעבורה אם ורק אם הפורט האקטיבי נופל. במידה והפורט האקטיבי עולה בחזרה, התעבורה תחזור לעבור דרכו.



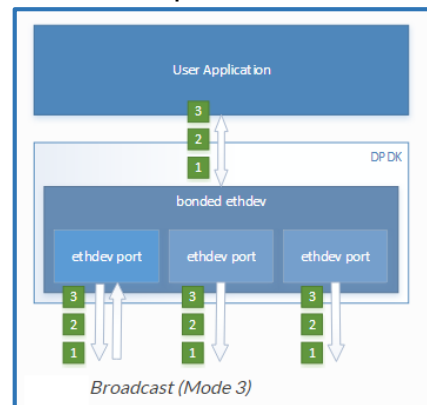
Balance XOR (מצב עבודה 2):

מצב עבודה זה מספק איזון עומסים בעת שליחת הודעות ועמידות בפני נפילות. על מנת לבחור לאיזה פורט לנתב את החבילות, יש לבחור במדיניות מסוימת לבחירת הפורט. מדיניות ברירת מחדל היא ביצוע חישוב בשכבה השנייה, המבוסס על כתובות ה-MAC של המקור והיעד ומספר הפורטים הפעילים. מדיניות נוספת שניתן לבחור היא בשכבות 2 ו-3, במקרה זה החישוב משתמש בכתובת IP של המקור והיעד. מדיניות אחרת היא בשכבות 3 ו-4 ומבוססת על כתובת IP ועל הפורט של המקור והיעד.



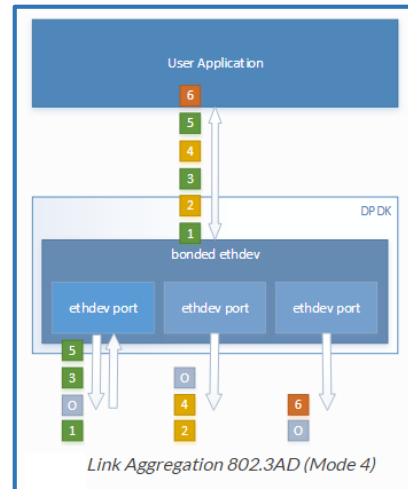
Broadcast (מצב עבודה 3):

מצב עבודה זה מספק עמידות בפני נפילות על ידי שליחת החבילה על כל הפורטים.



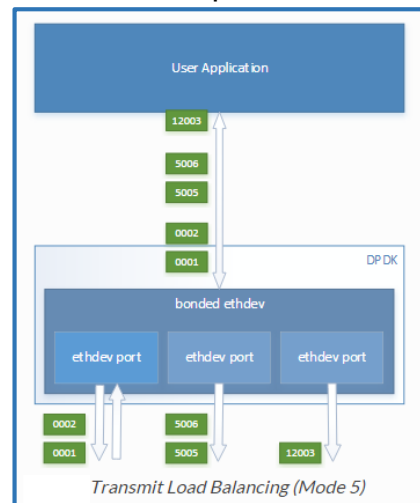
Link Aggregation (מצב עבודה 4):

מצב עבודה זה מאפשר link aggregation באופן דינאמי בהתאם לתקן IEEE 802.3ad. הוא מנטר אחר קבוצת הפורטים המאוחדים על מנת לאזן את העומסים בתעבורה היוצאת.



Transmit Load Balancing (מצב עבודה 5):

מצב עבודה זה מספק איזון עומסים דינאמי בזמן שליחת החבילות. הוא משנה באופן דינאמי את הפורטים האקטיביים בהתאם לחישוב של העומס.



מדריך התקנה של DPDK

דרישות מערכת

ישנם מספר התאמות שיש לעשות על המכונה עליה מתקינים את ה-DPDK.

- גרסת kernel מינימאלית היא 2.6.33 (ניתן לבדוק ע"י הפקודה `uname -r`)
- גרסת Glibc מינימאלית של 2.7 (ניתן לבדוק ע"י הפקודה `ldd -version`)
- בהפצות לינוקס כמו Red Hat Enterprise, Fedora, Ubuntu, ו-Red Hat Enterprise מובנת של ה-kernel מאפשרת שימוש ברוב האפליקציות של DPDK.
- ה-kernel צריך לתמוך ב-UIO. ניתן להוריד ולהפעיל ע"י הפעולות הבאות:

```
sudo -S modprobe uio
sudo insmod ~/dpdk-1.8.0/build/kmod/igb_uio.ko
```

שימוש ב-hugepages בסביבת לינוקס

התמיכה ב-hugepages נדרשת על מנת לאפשר הקצאה של כמות גדולה של זיכרון המשמש עבור חוצץ לחבילות.

על מנת לאפשר את התמיכה ב-hugepages יש להפעיל את האופציות הבאות ב-kernel:

```
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y
```

על ידי שימוש בהקצאות מסוג hugepages יש שיפור בביצועים, מכיוון שיש צורך בפחות דפים, ולכן פחות תרגום כתובות באמצעות Translation Lookaside Buffers, אשר מקצר את הזמן הנדרש לתרגום כתובות וירטואליות לכתובות פיזיות. ללא השימוש ב-hugepages, אלא שימוש בגודל ברירת המחדל של הדפים (4KB) תהיה עליה בקצב של הפספוסים בתרגום ובכך תהיה ירידה בביצועים.

הקצאת hugepages עבור DPDK

את ההקצאה של ה-hugepages יש לעשות בזמן העלייה של המערכת הפעלה או קרוב ככל האפשר לעליית המערכת, על מנת למנוע מהזיכרון להיות קטוע בזיכרון הפיזי. כדי להקצות hugepages בזמן העלייה, יש להעביר ל-kernel של לינוקס פרמטר בשורת הפקודה של ה-kernel.

לדוגמה, כדי להקצות 1024 דפים בגודל 2MB, יש להעביר את השורה הבאה כפרמטר של hugepages ל-kernel:

```
hugepages=1024
```

עבור גדלים אחרים של hugepages, הגודל צריך להיקבע באופן מפורש. למשל, עבור הקצאה בזיכרון של 4G של hugepages כאשר כל דף הוא בגודל 1G, הפרמטרים הבאים צריכים להיות מועברים ל-kernel:

```
default_hugepagesz=1G hugepagesz=1G hugepages=4
```

דרך אלטרנטיבית היא להקצות hugepages לאחר שהמערכת עלתה. ניתן לעשות זאת באמצעות הפקודה הבאה:

```
echo 1024 > /proc/sys/vm/nr_hugepages
```

בנוסף יש להוסיף לסוף של הקובץ `/etc/sysctl.conf` את השורה הבאה:

```
vm.nr_hugepages=1024
```

ניתן לבדוק את הסטטוס של hugepages באמצעות הפקודה הבאה:

```
grep -i huge /proc/meminfo
```

במצב תקין נקבל מסך דומה למסך הבא:

```
linkagg@silicom2:~/dpdk-1.8.0$ grep -i huge /proc/meminfo
AnonHugePages:      571392 kB
HugePages_Total:    1024
HugePages_Free:     1024
HugePages_Rsvd:     0
HugePages_Surp:     0
Hugepagesize:      2048 kB
```

אם HugePages_Free שווה ל-0, על מנת לאפשר ל-DPDK להשתמש בהם ניתן לשחרר אותם באמצעות הפקודות הבאות:

```
sudo lsof | grep -i huge
sudo kill -9 (process #)
sudo rm /mnt/huge/*
```

שימוש ב-hugepages על ידי ה-DPDK

לאחר ש-hugepages הוקצאו בזיכרון, יש צורך להפוך אותם להיות זמינים לשימוש של ה-DPDK. ניתן לעשות זאת באמצעות הפקודות הבאות:

```
mkdir /mnt/huge
mount -t hugetlbfs nodev /mnt/huge
```

בשביל שההגדרה תהיה קבועה ותישמר גם לאחר האיתחול, יש להוסיף לקובץ /etc/fstab את השורה הבאה:

```
nodev /mnt/huge hugetlbfs defaults 0 0
```

התקנת DPDK

יש לפתוח את הארכיון המקובץ של DPDK. DPDK מכיל מספר תיקיות:

- lib – קוד מקור של ספריות ה-DPDK.
- drivers – קוד מקור של הדרייברים של poll-mode.
- app – קוד מקור של אפליקציות ה-DPDK, כולל בדיקות אוטומטיות.
- examples – קוד מקור של אפליקציות לדוגמה של DPDK.
- config, tools, scripts, mk – קבצים שקשורים ל-framework של DPDK, קבצי make, סקריפטים והגדרות.

על מנת להתקין ולבצע make לקבצים יש להריץ את הפקודה הבאה מהתיקייה העליונה ב-DPDK:

```
make install T=i686-native-linuxapp-gcc
```

ברגע שהסביבה נוצרה, ניתן לעשות make לשינויים בקוד ולקמפל מחדש. כמו כן, אפשר לעשות שינויים לתצורה של DPDK שנקבעת בזמן קומפילציה על ידי עריכת הקובץ config. בתיקיית build.

קישור פורטי רשת למודולים של ה-kernel

כל הפורטים שאפליקציית DPDK משתמשת בהם צריכים להיות קשורים ל-igb_uio לפני שהאפליקציה מופעלת. דרייבר poll-mode של ה-DPDK יתעלם מכל פורטי הרשת שנמצאים תחת השליטה של לינוקס והאפליקציות לא יוכלו להשתמש בהם.

באמצעות הסקריפט dpdk_nic_bind.py הנמצא בתיקיית tools, ניתן לקשר את הפורטים למודול של igb_uio על מנת שה-DPDK יוכל להשתמש בהם ובאופן דומה ניתן לנתק את הפורטים מהמודול ולהחזיר אותם לשליטה של הלינוקס. כלי זה מאפשר גם לראות תמונת מצב של הפורטים במערכת.

הפקודה שבאמצעותה מצרפים את הפורטים למודול היא:

```
sudo /home/linkagg/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio 05:00:0
```

```
sudo /home/linkagg/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio 05:00:1
```

הפרמטר האחרון של הפקודה הוא כתובת ה-PCI של כרטיס הרשת ומספר הפורט אותו נרצה לקשר ל-igb_uio. במקרה זה הכתובת והפורטים הם 05:00:0 ו-05:00:1. כתובת ה-PCI אינה קבועה ויכולה להשתנות בשרתים אחרים.

על מנת לראות את הסטאטוס של הפורטים במערכת ניתן להשתמש בפקודה הבאה:

```
sudo /home/linkagg/dpdk-1.8.0/tools/dpdk_nic_bind.py --status
```

דוגמא לפלט:

```
linkagg@silicom2:~/dpdk-1.8.0$ ./tools/dpdk_nic_bind.py --status
Network devices using DPDK-compatible driver
=====
0000:05:00.0 'Ethernet Controller 10-Gigabit X540-AT2' drv=igb_uio unused=
0000:05:00.1 'Ethernet Controller 10-Gigabit X540-AT2' drv=igb_uio unused=
Network devices using kernel driver
=====
0000:03:00.0 'NetXtreme BCM5717 Gigabit Ethernet PCIe' if=eth0 drv=tg3 unused=igb_uio *Active*
0000:03:00.1 'NetXtreme BCM5717 Gigabit Ethernet PCIe' if=eth1 drv=tg3 unused=igb_uio
```

Pktgen

Pktgen (Packet Generator) הינה תוכנה אשר מחוללת תעבורת רשת ומפעילה framework לעיבוד מהיר של החבילות של ה-DPDK. תכונות של ה-pktgen:

- מחולל תעבורת רשת בקצב של 10Gbit עם גודל מסגרת של 64 בתים.
- יכול לעבוד בתור הצד המקבל או בתור הצד השולח.
- ישנה סביבת ריצה שבה ניתן לקבוע את ההגדרות, להתחיל ולעצור את זרימת התעבורה.
- מציג מדדים בזמן אמת עבור מספר פורטים.
- יכול לשלוח חבילות על פי הסדר על ידי מעבר על כתובת MAC, IP והפורטים של המקור והיעד.

מדריך התקנה ל-Pktgen

הדרישה המרכזית על מנת להתקין את ה-Pktgen היא שהמערכת תתמוך ב-Framework של ה-DPDK.

לאחר שה-DPDK הותקן, נעתיק את הספריות של ה-Pktgen ונקמפל אותם באמצעות הפקודה `make`. על מנת להגדיר את סביבת העבודה של ש-Pktgen נריץ את הסקריפט `setup.sh` אשר נמצא בתיקיה העליונה בהיררכיה.

הרצת Pktgen

שורת פקודה בסיסית להריץ את ה-Pktgen תראה מהצורה הבאה:

```
sudo -E ./pktgen -c f -n 4 -- -P -m 1.0,2.1
```

Pktgen, בדומה לאפליקציות אחרות של DPDK, מפריד את הארגומנטים בשורת הפקודה על ידי התווים – לארגומנטים של ה-DPDK עצמו ולארגומנטים של ה-Pktgen.

הפרמטר `-P` מגדיר מצב עבודה `Promiscuous`, שבו הבקר מעביר את כל תעבורת הרשת אשר הוא מקבל ליחידת העיבוד (ללא סינון), לעומת המצב ברגיל שבו הבקר מעביר רק את המיועדות לו.

הפרמטר `-m` ממפה את הפורטים לליבות הלוגיות. בדוגמה הנ"ל הוגדר כי ליבה 1 מטפלת בתור rx ו-tx של פורט 0 וליבה 2 מטפלת בתור rx ו-tx של פורט 1.

הפלט המתקבל מהרצת Pktgen נראה כך:

```
-----
Copyright notices
-----
EAL: Detected lcore 0 as core 0 on socket 0
EAL: Detected lcore 1 as core 1 on socket 0
...
EAL: PCI device 0000:07:00.1 on NUMA socket 0
EAL: probe driver: 8086:1521 rte_igb_pmd
EAL: 0000:07:00.1 not managed by UIO driver, skipping
Lua 5.2.1 Copyright (C) 1994-2012 Lua.org, PUC-Rio

>>> Packet Burst 16, RX Desc 256, TX Desc 256, mbufs/port 2048, mbuf cache 256

=== port to lcore mapping table (# lcores 5) ===
lcore:  0  1  2  3  4
port  0: D: T 1: 0 0: 0 0: 1 0: 0 = 1: 1
port  1: D: T 0: 0 1: 0 0: 0 0: 1 = 1: 1
Total  : 0: 0 1: 0 1: 0 0: 1 0: 1
        Display and Timer on lcore 0, rx:tx counts per port/lcore

Configuring 6 ports, MBUF Size 1984, MBUF Cache Size 256
Lcore:
  1, type RX , rx_cnt 1, tx_cnt 0, RX (pid:qid): ( 0: 0) , TX (pid:qid):
  2, type RX , rx_cnt 1, tx_cnt 0, RX (pid:qid): ( 1: 0) , TX (pid:qid):
  3, type TX , rx_cnt 0, tx_cnt 1, RX (pid:qid): , TX (pid:qid): ( 0: 0)
  4, type TX , rx_cnt 0, tx_cnt 1, RX (pid:qid): , TX (pid:qid): ( 1: 0)

Port :
  0, nb_lcores 2, private 0x7d08d8, lcores: 1 3
  1, nb_lcores 2, private 0x7d1c48, lcores: 2 4

Initialize Port 0 -- TxQ 1, RxQ 1, Src MAC 90:e2:ba:5a:f7:90
Create: Default RX 0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Default TX 0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Range TX 0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Sequence TX 0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Special TX 0:0 - Mem (MBUFs 64 x (1984 + 64)) + 790720 = 901 KB

Port memory used = 20373 KB

Initialize Port 1 -- TxQ 1, RxQ 1, Src MAC 90:e2:ba:5a:f7:91
Create: Default RX 1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Default TX 1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Range TX 1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Sequence TX 1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
Create: Special TX 1:0 - Mem (MBUFs 64 x (1984 + 64)) + 790720 = 901 KB

Port memory used = 20373 KB
Total memory used = 40746 KB

Port 0: Link Up - speed 10000 Mbps - full-duplex <Enable promiscuous mode>
Port 1: Link Up - speed 10000 Mbps - full-duplex <Enable promiscuous mode>

=== Display processing on lcore 0
=== RX processing on lcore 1, rxcnt 1, port/qid, 0/0
=== RX processing on lcore 2, rxcnt 1, port/qid, 1/0
=== TX processing on lcore 3, txcnt 1, port/qid, 0/0
=== TX processing on lcore 4, txcnt 1, port/qid, 1/0
...

```

כאשר ה-Pktgen רץ הפלט על המסך נראה כך:

```
- Ports 0-3 of 6  ** Main Page **  Copyright
  Flags:Port      :  P-----:0      P-----:1
Link State       :      <UP-10000-FD>      <UP-10000-FD>  ---TotalRate---
Pkts/s Rx       :              0              0              0
Tx              :              0              0              0
Mbits/s Rx/Tx   :              0/0              0/0              0/0
Broadcast        :              0              0
Multicast       :              0              0
  64 Bytes      :              0              0
  65-127        :              0              0
  128-255       :              0              0
  256-511       :              0              0
  512-1023      :              0              0
  1024-1518     :              0              0
Runts/Jumbos    :              0/0              0/0
Errors Rx/Tx    :              0/0              0/0
Total Rx Pkts   :              0              0
  Tx Pkts       :              0              0
  Rx MBs        :              0              0
  Tx MBs        :              0              0
ARP/ICMP Pkts  :              0/0              0/0
:
Tx Count/% Rate :      Forever/100%      Forever/100%
PktSize/Tx Burst:           64/16           64/16
Src/Dst Port    :           1234/5678           1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001  IPv4/TCP:0001
Dst IP Address  :           192.168.1.1           192.168.0.1
Src IP Address  :           192.168.0.1/24       192.168.1.1/24
Dst MAC Address :  90:e2:ba:5a:f7:91  90:e2:ba:5a:f7:90
Src MAC Address :  90:e2:ba:5a:f7:90  90:e2:ba:5a:f7:91

Pktgen>
```

פקודות של Pktgen בזמן ריצה

בזמן הריצה של ה-Pktgen ניתן להריץ פקודות על מנת לשנות את מצב העבודה, להתחיל ולעצור את התעבורה. הפקודות העיקריות הן:
על מנת להגדיר את גודל החבילות שנשלחות נריץ את הפקודה הבאה:

```
set all size 1024
```

start – הפקודה מתחילה לשלוח את החבילות. דוגמה לשימוש:

```
start all
```

הפקודה תתחיל לשלוח חבילות על כל הפורטים המוגדרים.

stop – הפקודה עוצרת את השליחה של החבילות. דוגמה לשימוש

```
stop all
```

הפקודה תעצור את השליחה על כל הפורטים המוגדרים.

סכמת המערכת

המערכת מורכבת משני שרתי HP.
השרתים מריצים מערכת הפעלה Ubuntu 14.04 64-bit.
בכל שרת מותקן כרטיס רשת מסוג Silicom Directory Card . הכרטיס תומך בטכנולוגיה של DPDK של אינטל. כל כרטיס מכיל 2 פורטים המסוגלים לעבוד במהירות של 10Gigabit.
הגרסה של ה-DPDK שהתקנו על השרתים היא 1.8.0.
הגרסה של ה-Pktgen אתה עבדנו היא: 2.8.4.
השרתים מחוברים ביניהם בקו ישיר.



קבלת ושליחת התעבורה תבצע ע"י pktgen אשר ישתמש בספריות dpdk כדי לממש את
הlink aggregation.

מימוש הפרויקט

מימוש ה-link aggregation מבוצע בתוך הקוד של ה-pktgen תוך שימוש בספריות DPDK. החיבור הלוגי משתמש בפונקציות מתוך ספריית `rte_eth_bond.h`. ערכנו את הקובצים `wr_l2p.c` ו-`pktgen-main.ci` על מנת ליצור את הפורט הלוגי בנוסף לשני הפורטים הפיזיים. הפורט הלוגי יכול לעבוד בשלושה מצבי עבודה:

- Round-Robin – הגדרת שני הפורטים בתור slave של הפורט הלוגי, חלוקת החבילות בין שני הפורטים.
- Active Backup – הגדרת פורט אחד כאקטיבי ופורט שני כפאסיבי.
- Broadcast – הגדרת שני הפורטים בתור slave של הפורט הלוגי ושכפול המידע בין שני הפורטים.

כל פורט (לוגי/פיזי) ממופה ל-core ייעודי שיטפל בו שבפועל ממופה ל-pthread חומרתי של לינוקס. כל thread חומרתי רץ על core בודד.

על מנת ליצור תמיכה דינאמית במצבי העבודה השונים ערכנו את הקובץ `pktgen-main.c`. הקובץ מכיל פונקציות הנקראות ישר לאחר הכנסת שורת הפקודה. הוספנו בקובץ קוד שיתמוך בפרמטר mode (M) שהתווסף לשורת ההרצה של ה-pktgen ושיעבוד במצב העבודה המבוקש.

```
sudo -E ./pktgen -c f -n 4 -- -P -M 0 -m 1.0,2.1
```

כתוצאה מהרצת הפקודה נקבל על המסך תמונת מצב של הפורטים:

```

| Ports 0-2 of 3 ** Main Page ** Copyright (c) <2010-2015>, Wind River Systems, Inc. All rights reserved. Powered by Intel® DPDK
Flags:Port : P-----:0 P-----:1 P-----:2
Link State : <UP-10000-FD> <UP-10000-FD> <UP-10000-FD> ---TotalRate---
Pkts/s Rx : 0 0 0 0
Tx : 0 0 0 0
MBytes/s Rx/Tx : 0/0 0/0 0/0 0/0
Broadcast : 0 0 0 0
Multicast : 0 0 0 0
64 Bytes : 0 0 0 0
65-127 : 0 0 0 0
128-255 : 0 0 0 0
256-511 : 0 0 0 0
512-1023 : 0 0 0 0
1024-1518 : 0 0 0 0
Runts/Jumbos : 0/0 0/0 0/0 0/0
Errors Rx/Tx : 0/0 0/0 0/0 0/0
Total Rx Pkts : 0 0 0 0
Tx Pkts : 0 0 0 0
Rx Mbs : 0 0 0 0
Tx Mbs : 0 0 0 0
ARP/ICMP Pkts : 0/0 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst: 64/32 64/32 64/32
Src/Dest Port : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address : 00:e0:ed:35:5d:16 00:e0:ed:35:5d:16 00:00:00:00:00:00
Src MAC Address : 00:e0:ed:35:5d:16 00:e0:ed:35:5d:16 00:e0:ed:35:5d:16
-- Pktgen Ver:2.8.4 (DPDK-1.8.0) -----
Pktgen>

```

נשים לב כי ישנם 3 פורטים: 2 פורטים פיזיים ופורט 1 לוגי, שהוא ה-link aggregation של שני הפורטים הפיזיים.

באמצעות הנוסחה הבאה נחשב את קצב השליחה והקבלה של ההודעות:

$$rate = \frac{T_x Pkts \cdot PktSize \cdot 8}{10^9 \cdot seconds}$$

הוספנו את קצב הקבלה/השליחה לשורות הסטטיסטיקה:

```

info = &pktgen.info[pid + sp];
rx = ((double)info->rate_stats.ipackets *
1518*8)/(1000000000);
tx = ((double)info->rate_stats.opackets *
1518*8)/(1000000000);
wr_scrn_printf(row++, col, "%*f", COLUMN_WIDTH_1, rx);
wr_scrn_printf(row++, col, "%*f", COLUMN_WIDTH_1, tx);

```

```

/ Ports 0-2 of 3  ** Main Page ** Copyright (c) <2010-2015>, Wind River System
Flags:Port      : P-----:0 P-----:1 P-----:2
Link State      : <UP-10000-FD> <UP-10000-FD> <UP-10000-FD>
---TotalRate--- 748679 0 748697
Tx              0 0 0
Rate Giga/s Rx  : 9.091958 0.000000 9.092176
Tx            : 0.000000 0.000000 0.000000
Multicast      : 0 0 0
64 Bytes      : 0 0 0
65-127        : 0 0 0
128-255       : 0 0 0
256-511       : 0 0 0
512-1023      : 0 0 0
1024-1518     : 3534975 0 2909949
Runts/Jumbos   : 0/0 0/0 0/0
Errors Rx/Tx   : 483467/0 0/0 483467/0
Total Rx Pkts  : 5771221 0 5771434
Tx Pkts        : 0 0 0
Rx MBs         : 71009 0 71011
Tx MBs         : 0 0 0
ARP/ICMP Pkts : 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst : 1518/32 1518/32 1518/32
Src/Dest Port    : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address   : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address   : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address  : 00:e0:ed:35:5d:1d 00:e0:ed:35:5d:1c 00:00:00:00:00:00
Src MAC Address  : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1d 00:e0:ed:35:5d:1c
-- Pktgen Ver:2.8.4(DPDK-1.8.0)

```

הקוד שהתווסף לפרויקט:

```

pktgen_parse_args(int argc, char **argv)
{
    int opt, ret, port, mode, port_id;
    char **argvopt;
    int option_index;
    char *prgname = argv[0], * p;
    static struct option lgopts[] = {
        {NULL, 0, 0, 0}
    };

    argvopt = argv;

    pktgen.hostname = (char *)strdup(pktgen.hostname,
"localhost");
    pktgen.socket_port = 0x5606;

    pktgen.argc = argc;
    for (opt = 0; opt < argc; opt++)
        pktgen.argv[opt] = strdup(argv[opt]);

    while ((opt = getopt_long(argc, argvopt, "p:M:m:f:l:s:g:hPNGT",
        lgopts, &option_index)) != EOF) {
        switch (opt) {

            case 'M':
            {
                //create bonded device 0 0
                char ethdev_name[256];
                snprintf(ethdev_name, 256, "eth_bond_testpmd_%d",
                    bond_dev_num++);

                mode = strtol(optarg, NULL, 10);
                /* Create a new bonded device. */
            }
        }
    }
}

```

```

    port_id = rte_eth_bond_create(ethdev_name, mode, 0);
    if (port_id < 0) {
        printf("\t Failed to create bonded device.\n");
        return -1;
    } else {
        printf("Created new bonded device %s mode %d on (port
%d).\n", ethdev_name,
            mode, port_id);
    }

    //add bonding slave
    /* Set the primary slave for a bonded device. */
    if (0 != rte_eth_bond_slave_add(port_id, 0)) {
        printf("\t Failed to add slave %d to master port =
%d.\n",
            0, port_id);
        return -1;
    }
    //init_port_config();
    if (0 != rte_eth_bond_slave_add(port_id, 1)) {
        printf("\t Failed to add slave %d to master port =
%d.\n",
            1, port_id);
        return -1;
    }

    if (mode == 1) {
        if (0 != rte_eth_bond_primary_set(port_id, 0)) {
            printf("\t Failed to create primary
slave.\n");
            return -1;
        }
    }
    break;
}

int wr_parse_matrix(lp_t * lp, char * str)
{
    char * lcore_port[MAX_MATRIX_ENTRIES];
    char buff[256];
    int i, m, k, lid_type, pid_type;
    uint32_t pid, lid;
    lp_t lp;
    rxtx_t cnt, n;

    const char* extension = ",3.2";
    char* new_str;
    new_str = malloc(strlen(str)+1+4); /* make space for the new
string (should check the return value ...) */
    strcpy(new_str, str); /* copy name into the new var */
    strcat(new_str, extension); /* add the extension */

    wr_strncpy(buff, new_str, " \r\n\t");
    free (new_str);

    // Split up the string into <lcore-port>, ...
    k = wr_strparse(buff, ",", lcore_port, countof(lcore_port));
    if ( k <= 0 ) {
        fprintf(stderr, "%s: could not parse (%s) string\n",
            __FUNCTION__, buff);
        return 0;
    }
}

```

```

}

for(i = 0; (i < k) && lcore_port[i]; i++) {
    char    * arr[3];
    char    str[64];

    // Grab a private copy of the string.
    strncpy(str, lcore_port[i], sizeof(str));

    // Parse the string into <lcore-list> and <port-list>
    m = wr_strparse( lcore_port[i], ".", arr, 3 );
    if ( m != 2 ) {
        fprintf(stderr, "%s: could not parse <lcore-list>.<port-
list> (%s) string\n",
                __FUNCTION__, lcore_port[i]);
        return 0;
    }

    memset(&lp, '\0', sizeof(lp));

    if ( wr_parse_lcore_list(arr[0], lp.lcores) ) {
string\n",
        fprintf(stderr, "%s: could not parse <lcore-list> (%s)
                __FUNCTION__, arr[0]);
        return 0;
    }

    if ( wr_parse_port_list(arr[1], lp.ports) ) {
string\n",
        fprintf(stderr, "%s: could not parse <port-list> (%s)
                __FUNCTION__, arr[1]);
        return 0;
    }

    // Handle the lcore and port list maps
    fprintf(stderr, "%-16s lcores: ", str);
    wr_dump_lcore_map("RX", lp.lcores[RX_IDX].ls);
    wr_dump_lcore_map("TX", lp.lcores[TX_IDX].ls);
    fprintf(stderr, " ports: ");
    wr_dump_port_map("RX", lp.ports[RX_IDX].ps);
    wr_dump_port_map("TX", lp.ports[TX_IDX].ps);
    fprintf(stderr, "\n");

    for(lid = 0; lid < RTE_MAX_LCORE; lid++) {
        lid_type = 0;
        if ( _btst(lp.lcores[RX_IDX].ls, lid) )
            lid_type |= RX_TYPE;
        if ( _btst(lp.lcores[TX_IDX].ls, lid) )
            lid_type |= TX_TYPE;
        if ( lid_type == 0 )
            continue;

        for(pid = 0; pid < RTE_MAX_ETHPORTS; pid++) {
            pid_type = 0;
            if ( _btst(lp.ports[RX_IDX].ps, pid) )
                pid_type |= RX_TYPE;
            if ( _btst(lp.ports[TX_IDX].ps, pid) )
                pid_type |= TX_TYPE;
            if ( pid_type == 0 )
                continue;
            wr_l2p_connect(l2p, pid, lid, lid_type);
        }
    }
}

```

```

    }
}

for(pid = 0; pid < RTE_MAX_ETHPORTS; pid++) {
    n.rxtx = 0;
    for(lid = 0; lid < RTE_MAX_LCORE; lid++) {
        if ( (cnt.rxtx = wr_get_map(l2p, pid, lid)) > 0 ) {
            if ( cnt.tx > 0 )
                n.tx++;
            if ( cnt.rx > 0 )
                n.rx++;
        }
    }
    l2p->map[pid][lid].rxtx = n.rxtx;           // Update the
lcores per port
}
for(lid = 0; lid < RTE_MAX_LCORE; lid++) {
    n.rxtx = 0;
    for(pid = 0; pid < RTE_MAX_ETHPORTS; pid++) {
        if ( (cnt.rxtx = wr_get_map(l2p, pid, lid)) > 0 ) {
            if ( cnt.tx > 0 )
                n.tx++;
            if ( cnt.rx > 0 )
                n.rx++;
        }
    }
    l2p->map[pid][lid].rxtx = n.rxtx;           // Update the
ports per lcore
}

return 0;
}

```

סנכרון תורי קבלה ושליחה:

כברירת מחדל, כל הפונקציות שה-PMD חושף הינן פונקציות ללא מנעולים. ליבות לוגיות (lcore) לא משתפות ביניהן את תורי השליחה והקבלה מכיוון שפעולה זו תדרוש שימוש במנעולים גלובליים ותוביל לירידה בביצועים. ההנחה היא שהקריאות לפונקציות אלה לא יהיו מקבילות על ליבות לוגיות שונות שעובדים על אותו אובייקט.

כיוון שהפורטים הפיזיים והלוגיים רצים על lcore שונים נדרשנו להוסיף מנגנון סנכרון בין התורים. מנגנון הסנכרון משתמש במנעולים המסופקים ע"י פלטפורמת ה-DPDK `rte_rwlock_write_unlock`. לשם כך, ערכנו את הקובץ `pktgen.c`, שם הכנסנו שינויים במימוש של הפונקציות המטפלות בשליחה ובקבלה על מנת שיעבדו בצורה מסונכרנת.

הקוד שהתווסף לפרויקט:

```
static __inline__ void
pktgen_main_transmit(port_info_t * info, uint16_t qid)
{
    /******LinkAgg*****/
    rte_rwlock_write_lock(&transmit_lock);
    uint32_t flags;
    flags = rte_atomic32_read(&info->port_flags);

    /*
     * Transmit ARP/Ping packets if needed
     */
    pktgen_send_special(info);

    // When not transmitting on this port then continue.
    if ( likely( (flags & SENDING_PACKETS) == SENDING_PACKETS ) ) {
        struct rte_mempool * mp = info->q[qid].tx_mp;

        if ( unlikely(flags &
(SEND_RANGE_PKTS|SEND_PCAP_PKTS|SEND_SEQ_PKTS)) ) {
            if ( flags & SEND_RANGE_PKTS )
                mp = info->q[qid].range_mp;
            else if ( flags & SEND_SEQ_PKTS )
                mp = info->q[qid].seq_mp;
            else if ( flags & SEND_PCAP_PKTS )
                mp = info->q[qid].pcap_mp;
        }

        pktgen_send_pkts(info, qid, mp);

        flags = rte_atomic32_read(&info->q[qid].flags);
        if ( unlikely(flags & (DO_TX_CLEANUP | DO_TX_FLUSH)) ) {
            if ( flags & DO_TX_CLEANUP )
                pktgen_tx_cleanup(info, qid);
            else if ( flags & DO_TX_FLUSH )
                pktgen_tx_flush(info, qid);
        }
    }
    /******LinkAgg*****/
    rte_rwlock_write_unlock(&transmit_lock);
}

static __inline__ void
pktgen_main_receive(port_info_t * info, uint8_t lid, struct rte_mbuf
*pkts_burst[])
```



```

{

uint8_t pid;
uint16_t qid, nb_rx;
capture_t *capture;

/*****LinkAgg*****/
rte_rwlock_write_lock(&recieve_lock);

pid = info->pid;
qid = wr_get_rxque(pktgen.l2p, lid, pid);
//pktgen_log_info("pktgen_main_RECEIVE: core: %d, qid:
%d\n",lid,qid);

/*
 * Read packet from RX queues and free the mbufs
 */
if ( (nb_rx = rte_eth_rx_burst(pid, qid, pkts_burst, info-
>tx_burst)) == 0 )
{
/*****LinkAgg*****/
rte_rwlock_write_unlock(&recieve_lock);
return;
}

// packets are not freed in the next call.
pktgen_packet_classify_bulk(pkts_burst, nb_rx, pid);

if ( unlikely(info->dump_count > 0) ){
//pktgen_log_info("inside dump_count\n");
pktgen_packet_dump_bulk(pkts_burst, nb_rx, pid);
}

if ( unlikely(rte_atomic32_read(&info->port_flags) &
CAPTURE_PKTS) ) {
capture = &pktgen.capture[pktgen.core_info[lid].s.socket_id];
if ( unlikely((capture->port == pid) && (capture->lcore ==
lid)) ) {
pktgen_packet_capture_bulk(pkts_burst, nb_rx, capture);
}
}
rte_pktmbuf_free_bulk(pkts_burst, nb_rx);

/*****LinkAgg*****/
rte_rwlock_write_unlock(&recieve_lock);
}

```

הגדלת תעבורת ה-pktgen:

פלטפורמת ה-pktgen נועדה לשליחת תעבורה מקסימלית של 10 Gbit/Sec. נרצה להגיע כיוון ששני כרטיסי הרשת שלנו תומכים בתעבורה מקסימלית של 10 Gbit/Sec, נרצה להגיע בפורט הלוגי לתעבורה מקסימלית של 20 Gbit/Sec. לכן, שינינו את זמן השליחה של החבילות כך שהזמן בין כל שליחת burst של החבילות מתקצר. כך התעבורה ברוטו גדלה ל- 20 Gbit/Sec.

הקוד שהתווסף (זמן ה-tx cycles קטן):

```
pktgen_packet_rate(port_info_t * info)
{
    /* 0 1 2 3 4 5 6 7 8 9
10 */
    static int64_t ff[11] = { 31, 30, 25, 30, 17, 17, 17, 20, 50, 60,
90 };
    uint64_t wire_size = (pktgen_wire_size(info) * 8);
    uint64_t link = (uint64_t)info->link.link_speed * Million;
    uint64_t pps = ((link/wire_size) * info->tx_rate)/100;
    uint64_t cpp = (pps > 0) ? (pktgen.hz/pps) : (pktgen.hz / 4);

    info->tx_pps = pps;
    info->tx_cycles = ((cpp * info-
>tx_burst)/wr_get_port_txcnt(pktgen.l2p, info->pid))/4;
    info->tx_cycles -= ff[info->tx_rate/10];
}
```

סטטיסטיקות

על מנת לאמוד את הביצועים של ה-link aggregation נריץ מספר בדיקות ונציג מידע סטטיסטי לגביהן.

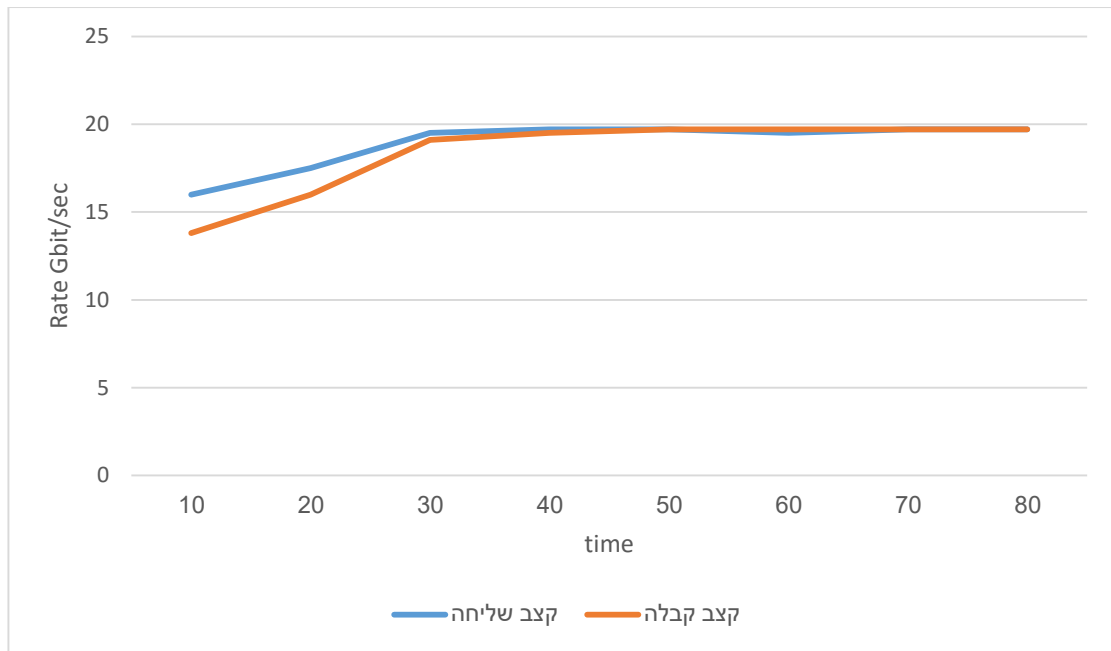
Round-Robin

המסך הבא התקבל כתוצאה מהפעלת ה-pktgen במצב של Round-Robin לאחר 60 שניות: צד מקבל:

```
| Ports 0-2 of 3 ** Main Page ** Copyright (c) <2010-2015>, Wind River System
Flags:Port : P-----:0 P-----:1 P-----:2
Link State : <UP-10000-FD> <UP-10000-FD> <UP-10000-FD>
---TotalRate-- 811267 811372 1623053
Tx : 0 0 0
Rate Giga/s Rx : 9.852026 9.853302 19.710356
Tx : 0.000000 0.000000 0.000000
3245692
0
39935/0
Multicast : 0 0 0
64 Bytes : 0 0 0
65-127 : 0 0 0
128-255 : 0 0 0
256-511 : 0 0 0
512-1023 : 0 0 0
1024-1518 : 8762995 8667553 14514793
Runts/Jumbos : 0/0 0/0 0/0
Errors Rx/Tx : 1709671/0 1706921/0 3416592/0
Total Rx Pkts : 15319734 15322298 30642446
Tx Pkts : 44950208 44946304 89896512
Rx MBs : 188494 188525 377024
Tx MBs : 553067 553019 1106086
ARP/ICMP Pkts : 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst : 1518/32 1518/32 1518/32
Src/Dest Port : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:00:00:00:00:00
Src MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c
-- Pktgen Ver:2.8.4(DPDK-1.8.0) -----
```

צד שולח:

```
\ Ports 0-2 of 3 ** Main Page ** Copyright (c) <2010-2015>, Wind River System
Flags:Port : P-----:0 P-----:1 P-----:2
Link State : <UP-10000-FD> <UP-10000-FD> <UP-10000-FD>
---TotalRate-- 0 0 0
Tx : 792286 792250 1584536
Rate Giga/s Rx : 0.000000 0.000000 0.000000
Tx : 9.621521 9.621084 19.242605
0
3169072
0/38992
Multicast : 0 0 0
64 Bytes : 0 0 0
65-127 : 0 0 0
128-255 : 0 0 0
256-511 : 0 0 0
512-1023 : 0 0 0
1024-1518 : 0 0 0
Runts/Jumbos : 0/0 0/0 0/0
Errors Rx/Tx : 0/0 0/0 0/0
Total Rx Pkts : 0 0 0
Tx Pkts : 17265099 17263380 34528908
Rx MBs : 0 0 0
Tx MBs : 212429 212408 424843
ARP/ICMP Pkts : 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst : 1518/32 1518/32 1518/32
Src/Dest Port : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:00:00:00:00:00
Src MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c
-- Pktgen Ver:2.8.4(DPDK-1.8.0) -----
```



ניתן לראות כי התעבורה מגיעה לקצב מקסימלי בערך לאחר כ-30 שניות.
 הקצב המקסימלי שהושג הוא 19.7 Gbit/Sec .

Active-Backup

הפעלה של pktgen במצב עבודה Active-Backup כאשר הפורט האקטיבי הוא פורט 0 ופורט 1. הגיבוי הוא פורט 2.

המסך הבא התקבל כתוצאה מהפעלת ה-pktgen במצב של Active-Backup לאחר 60 שניות: צד שולח:

```

/ Ports 0-2 of 3 ** Main Page ** Copyright (c) <2010-2015>, Wind River System
Flags:Port : P-----:0 P-----:1 P-----:2
Link State : <UP-10000-FD> <UP-10000-FD> <UP-10000-FD>
---TotalRate--
Tx : 748679 0 748697
Rate Giga/s Rx : 9.091958 0.000000 9.092176
Tx : 0.000000 0.000000 0.000000
1497376
0
18423/0
Multicast : 0 0 0
64 Bytes : 0 0 0
65-127 : 0 0 0
128-255 : 0 0 0
256-511 : 0 0 0
512-1023 : 0 0 0
1024-1518 : 3534975 0 2909949
Runts/Jumbos : 0/0 0/0 0/0
Errors Rx/Tx : 483467/0 0/0 483467/0
Total Rx Pkts : 5771221 0 5771434
Tx Pkts : 0 0 0
Rx MBS : 71009 0 71011
Tx MBS : 0 0 0
ARP/ICMP Pkts : 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst : 1518/32 1518/32 1518/32
Src/Dest Port : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address : 00:e0:ed:35:5d:1d 00:e0:ed:35:5d:1c 00:00:00:00:00:00
Src MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1d 00:e0:ed:35:5d:1c
-- Pktgen Ver:2.8.4(DPDK-1.8.0) -----

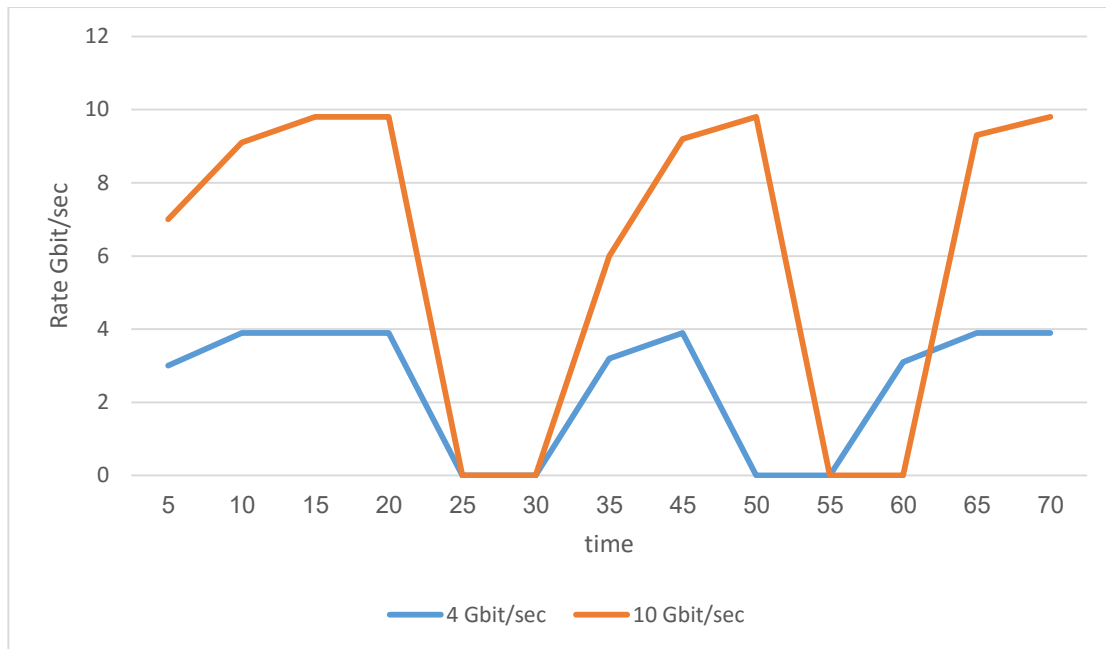
```

צד מקבל:

```

- Ports 0-2 of 3 ** Main Page ** Copyright (c) <2010-2015>, Wind River Sy
Flags:Port : P-----:0 P-----:1 P-----:2
Link State : <--Down--> <UP-10000-FD> <UP-10000-FD>
---TotalRate--
Tx : 0 749644 749670
Rate Giga/s Rx : 0.000000 9.103677 9.103992
Tx : 0.000000 0.000000 0.000000
1499314
0
18447/0
Multicast : 0 0 0
64 Bytes : 0 0 0
65-127 : 0 0 0
128-255 : 0 0 0
256-511 : 0 0 0
512-1023 : 0 0 0
1024-1518 : 8566797 3319749 12707873
Runts/Jumbos : 0/0 0/0 0/0
Errors Rx/Tx : 1504398/0 1015976/0 2520374/0
Total Rx Pkts : 16487397 7451531 23939145
Tx Pkts : 0 0 0
Rx MBS : 202860 91683 294547
Tx MBS : 0 0 0
ARP/ICMP Pkts : 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst : 1518/32 1518/32 1518/32
Src/Dest Port : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address : 00:e0:ed:35:5d:1d 00:e0:ed:35:5d:1c 00:00:00:00:00:00
Src MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1d 00:e0:ed:35:5d:1c
-- Pktgen Ver:2.8.4(DPDK-1.8.0) -----

```



ניתן לראות כי התעבורה מגיעה לקצב מקסימלי בערך לאחר כ-15 שניות.
 זמן ההתאוששות בשני המקרים היה כ-5 שניות.
 ההגעה לקצב המקסימלי עבור $4 \frac{Gbit}{Sec}$, $10 \frac{Gbit}{Sec}$ לפורט יחיד הייתה $3.9 \frac{Gbit}{Sec}$, $9.8 \frac{Gbit}{Sec}$ בהתאמה.

Broadcast

המסך הבא התקבל כתוצאה מהפעלת ה-pktgen במצב של Broadcast לאחר 60 שניות:
צד מקבל:

```

/ Ports 0-2 of 3  ** Main Page **  Copyright (c) <2010-2015>, WInd River Sys
  Flags:Port      : P-----:0 P-----:1 P-----:2
Link State       : <UP-10000-FD> <UP-10000-FD> <UP-10000-FD>
Pkts/s Rx       : 807997 808000 1615995
Tx               : 0 0 0
Rate Giga/s Rx  : 9.812316 9.812352 19.624643
Tx               : 0.000000 0.000000 0.000000
Missed Rx       : 2.373227 2.292961 0.000000
Mbits/s Rx/Tx   : 9941/0 9941/0 19883/0
Broadcast       : 0 0 0
Multicast       : 0 0 0
  64 Bytes      : 0 0 0
  65-127        : 0 0 0
  128-255       : 0 0 0
  256-511       : 0 0 0
  512-1023      : 0 0 0
  1024-1518     : 29774997 30458905 51504046
Runts/Jumbos    : 0/0 0/0 0/0
Errors Rx/Tx    : 6820948/0 6731380/0 13552328/0
Total Rx Pkts   : 55136201 55225871 110362486
Tx Pkts         : 0 0 0
Rx MBs          : 678395 679499 1357900
Tx MBs          : 0 0 0
ARP/ICMP Pkts  : 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst: 1518/32 1518/32 1518/32
Src/Dest Port   : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address  : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address  : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c
Src MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c
-- Pktgen Ver:2.8.4(DPDK-1.8.0) -----

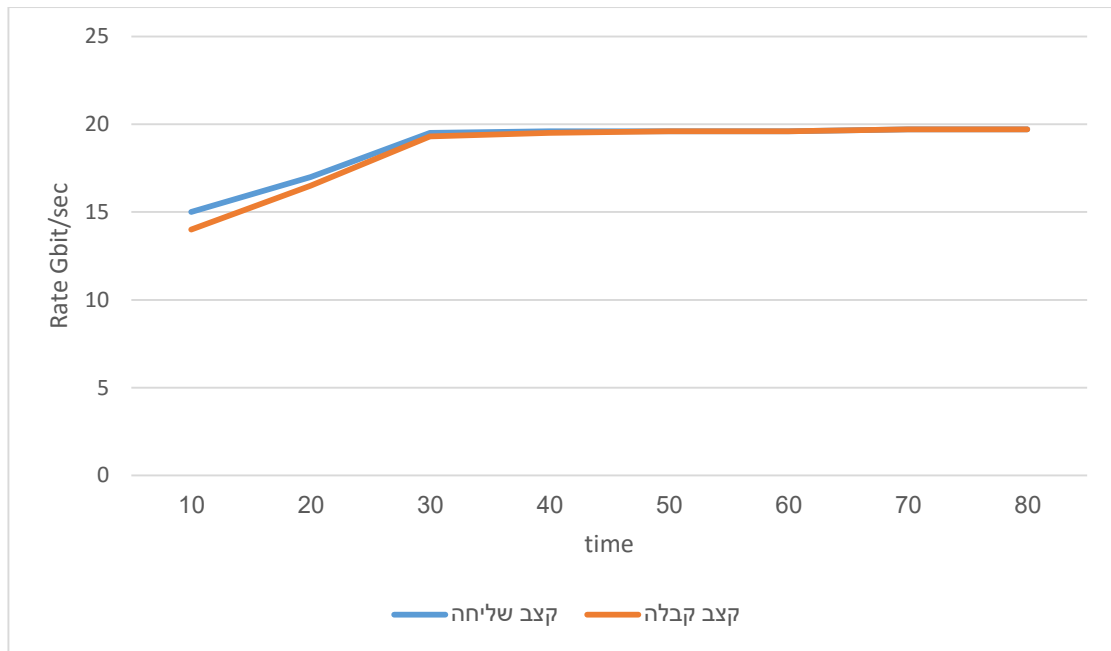
```

צד שולח:

```

\ Ports 0-2 of 3  ** Main Page **  Copyright (c) <2010-2015>, WInd River Sy
  Flags:Port      : P-----:0 P-----:1 P-----:2
Link State       : <UP-10000-FD> <UP-10000-FD> <UP-10000-FD>
Pkts/s Rx       : 0 0 0
Tx               : 806461 806470 1612934
Rate Giga/s Rx  : 0.000000 0.000000 0.000000
Tx               : 9.793662 9.793772 19.587470
Missed Rx       : 6.367806 6.176746 0.000000
Mbits/s Rx/Tx   : 0/9922 0/9922 0/19845
Broadcast       : 0 0 0
Multicast       : 0 0 0
  64 Bytes      : 0 0 0
  65-127        : 0 0 0
  128-255       : 0 0 0
  256-511       : 0 0 0
  512-1023      : 0 0 0
  1024-1518     : 0 0 0
Runts/Jumbos    : 0/0 0/0 0/0
Errors Rx/Tx    : 0/0 0/0 0/0
Total Rx Pkts   : 0 0 0
Tx Pkts         : 54124308 54123239 108247980
Rx MBs          : 0 0 0
Tx MBs          : 665945 665932 1331882
ARP/ICMP Pkts  : 0/0 0/0 0/0
Tx Count/% Rate : Forever/100% Forever/100% Forever/100%
PktSize/Tx Burst: 1518/32 1518/32 1518/32
Src/Dest Port   : 1234/5678 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001 IPv4/TCP:0001
Dst IP Address  : 192.168.1.1 192.168.0.1 192.168.3.1
Src IP Address  : 192.168.0.1/24 192.168.1.1/24 192.168.2.1/24
Dst MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c
Src MAC Address : 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c 00:e0:ed:35:5d:1c
-- Pktgen Ver:2.8.4(DPDK-1.8.0) -----

```



ניתן לראות כי התעבורה מגיעה לקצב מקסימלי בערך לאחר כ-30 שניות.
 הקצב המקסימלי שהושג הוא 19.6 Gbit/Sec .

מסקנות

ביצוע link aggregation באמצעות DPDK ניתן ליישום.
נראה כי יש הבדל בין המצבים השונים של link aggregation:

1. קצב השליחה/הקבלה אינו מגיע לקצב המקסימלי 20 Gbit/Sec , אך מאוד קרוב אליו. הסיבה לכך היא ההכנסה של סנכרון התורים שהיה נחוץ להכניס, המוסיף תקורה לשליחה וקבלה.
2. ב-Round-Robin קצב השליחה והקבלה קרובים מאוד לקצב המקסימלי. לאחר ההתייצבות הקצב מגיע ל- 19.7 Gbit/Sec .
3. נראה שקצב השליחה המקסימלי ב-broadcast נמוך במקצת מאשר קצב השליחה ב-Round-Robin. הסיבה לכך היא הוספת תקורה נוספת עבור שכפול החבילה לשם שליחתה בשני הפורטים.
4. ב-Active Backup אין שיפור בביצועים לעומת עבודה ללא link aggregation. קצב השליחה והקבלה שהתקבל הינו 9.8 Gbit/sec , קצת פחות מהקצב המקסימלי של 10 Gbit/sec בעבודה ללא link aggregation. הסיבה לכך היא שבמצב עבודה זה רק פורט אחד מעביר את התעבורה בפועל, והפורט השני הוא גיבוי.
5. כאשר קצב השליחה נמוך יותר מתקבלת פחות תקורה ועקב כך הגעה לקצבים מקסימלים גבוהים יותר (קרובים למקסימלי האידיאלי).
6. זמן ההתאוששות של לינק שנפל זהה בקירוב לקצבי שליחה שונים.

ביבליוגרפיה

https://en.wikipedia.org/wiki/Link_aggregation

<http://dpdk.org/>

<http://dpdk.org/doc/guides/index.html>

https://en.wikipedia.org/wiki/Data_Plane_Development_Kit

<http://pktgen.readthedocs.org/en/latest/index.html>