דוח פרויקט - רשתות מהירות ומיתוג

Project - Flow Optimization in Multipath Network using Maximum Flow Algorithm to Improve the Bandwidth Usage in SDN

מגישים:

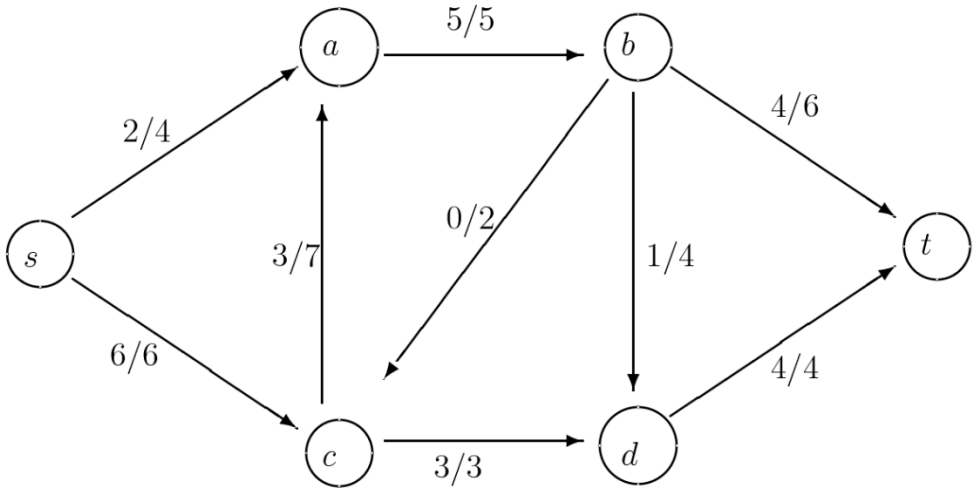
דניאל בלוזרוב - 315306464

שחף זוהר - 205978000

עידן כהן – 207598509

רעיון כללי:

מימוש אלגוריתם Max Flow ברשת Multipath שהיא SOFTWARE DEFINED NETWORK והשוואה אל מול אלגוריתם ניתוב דייקסטרה.
Max Flow הוא אלגוריתם מוכר בתורת הגרפים.



עבור גרף לא מכוון ובעל קיבולת על הקשתות, ועבור 2 צמתים שהם מקור ובור, ניתן להגדיר זרימה.
הזרימה יכולה לתאר מים שעוברים בצינורות, או במקרה שלנו, מידע שעובר בין לינקים של סוויצ'ים, כאשר אנחנו מוגבלים על ידי ה bandwidth בכל לינק.
אנחנו רוצים למקסם את כמות התעבורה שיכולה לעבור מ node A אל node B.
אנחנו נשתמש באלגוריתם של Dinic , בשימוש בטופולוגייה שלנו שנשלטת על ידי RYU CONTROLLER שמהווה את המוח של הרשת, שהוא יממש את האלגוריתם וימצא מסלולי ניתוב בין s ל t כך שהנצילות תיהיה כמה שיותר גבוהה.
לאחר מכן אנחנו נשווה אל מול single path Dijkstra ונראה למי יש throughput גבוה יותר.(גם UDP וגם TCP)

האלגוריתם:

אנחנו נשתמש באלגוריתם DINIC (מויקיפדיה)

Definition [edit]

Let $G = ((V, E), c, f, s, t)$ be a network with $c(u, v)$ and $f(u, v)$ the capacity and the flow of the edge (u, v), respectively.

The **residual capacity** is a mapping $c_f \colon V \times V \to R^+$ defined as,

1. if $(u, v) \in E$,
   $c_f(u, v) = c(u, v) - f(u, v)$
2. if $(v, u) \in E$,
   $c_f(u, v) = f(v, u)$
3. $c_f(u, v) = 0$ otherwise.

The **residual graph** is an unweighted graph $G_f = ((V, E_f), c_f|_{E_f}, s, t)$, where

$E_f = \{(u, v) \in V \times V \colon c_f(u, v) > 0\}$.

An **augmenting path** is an s–t path in the residual graph G_f.

Define $\operatorname{dist}(v)$ to be the length of the shortest path from s to v in G_f. Then the **level graph** of G_f is the graph $G_L = ((V, E_L), c_f|_{E_L}, s, t)$, where

$E_L = \{(u, v) \in E_f \colon \operatorname{dist}(v) = \operatorname{dist}(u) + 1\}$.

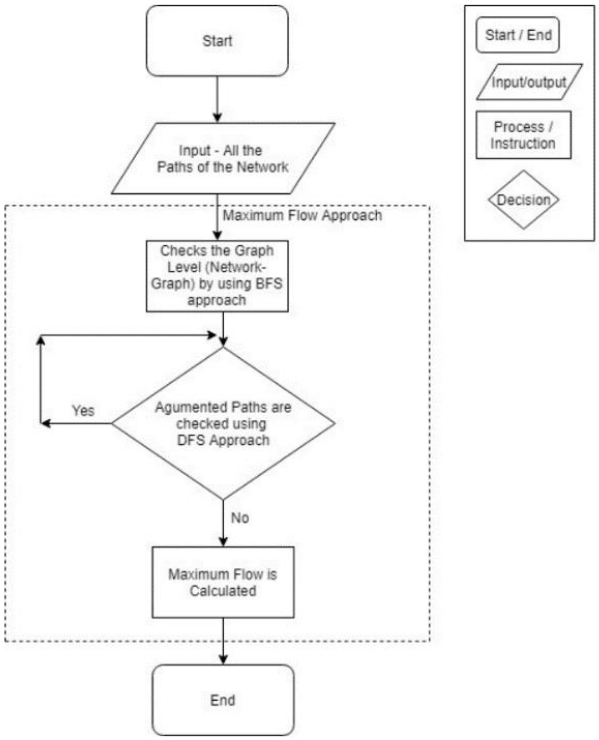
A **blocking flow** is an s–t flow f' such that the graph $G' = ((V, E'_L), s, t)$ with $E'_L = \{(u, v) \colon f'(u, v) < c_f|_{E_L}(u, v)\}$ contains no s–t path.
[Note 1][4]

Algorithm [edit]

Dinic's Algorithm

Input: A network $G = ((V, E), c, s, t)$.
Output: An s–t flow f of maximum value.

1. Set $f(e) = 0$ for each $e \in E$.
2. Construct G_L from G_f of G. If $\operatorname{dist}(t) = \infty$, stop and output f.
3. Find a blocking flow f' in G_L.
4. Add augment flow f by f' and go back to step 2.

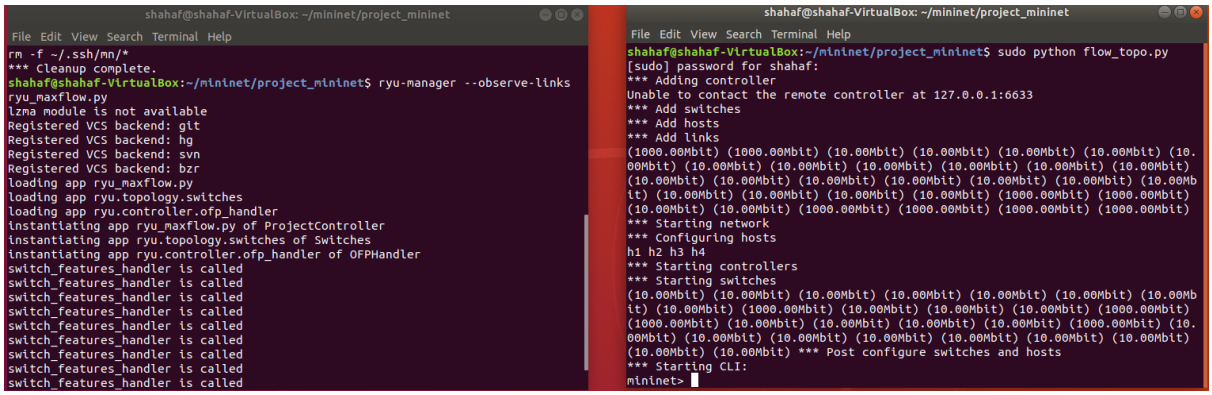


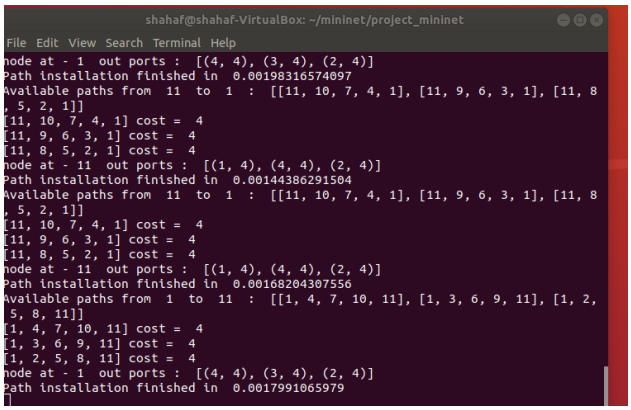
בסעיף זה נתאר את התרשים ברמה גבוהה של המימוש במחקר זה.

כדי להתחיל את גישת הזרימה המרבית, אנו זקוקים לכל הנתיבים ברשת כקלט. כאשר גישת הזרימה המרבית מתחילה, BFS (חיפוש לרוחב) בודק את הרמה של גרף הרשת מרביים. על ידי כך, נוכל לקבל את הערך של כל צומת כדי למצוא את המרחק הקצר מצומת המקור. והקשתות מגרף הרמה מבצעות מספר רב של גישות DFS (חיפוש לעומק) מהמקור אל היעד עד שאין נתיבים מרובים אפשריים, אז מוצאים את הזרימה המרבית. לאחר שקיבלנו את הערך של הזרימה המרבית, בקר RYU מגדיר את זה כערך קבוע לנתיבים המרביים שמשמשים לניתוב מרובה נתיבים. באמצעות גישה זו אנו יכולים לשפר את התפוקה. .

הניסוי שלנו:

בפרויקט כתבנו אלגוריתם בתוכנת Mininet אשר מממש את Max Flow בטופולוגיה שלנו(תמונה למטה).
בהתחלה הרמנו את הטופולוגיה, ואז הרצנו RYU Controller עם פונקציית ניתוב שונה (MF\Dijkstra) כדי לראות איפה יש load-balancing טוב יותר.
אנחנו משתמשים בפרוטוקול IPERF בשביל למדוד throughput עבור 3 מחשבים ששולחים מידע למחשב שרת ובודקים את ה load-balancing כך שתיהיה נצילות של כל הלינקים בשימוש של multi-path עבור כל flow שונה כאשר flow נקבע על ידי source IP, dest IP לפי פונקציית hash של OpenFlow 1.3.
השתמשנו ב iperf למשך 20 שניות ושלחנו חבילות בגודל המקסימלי שיכולנו ולקחנו את ה throughput שהשרת מצא עבור כל מחשב ששלח לו, גם עבור UDP וגם עבור TCP.לבסוף השוונו ביניהם.





בתמונה הראשונה אפשר לראות את הרצת הטופולוגיה ב mininet ביחד עם הryu-controller שמריץ maxflow .

לאחר מכן עשינו ping בשביל לראות שיש חיבור והכל עובד.

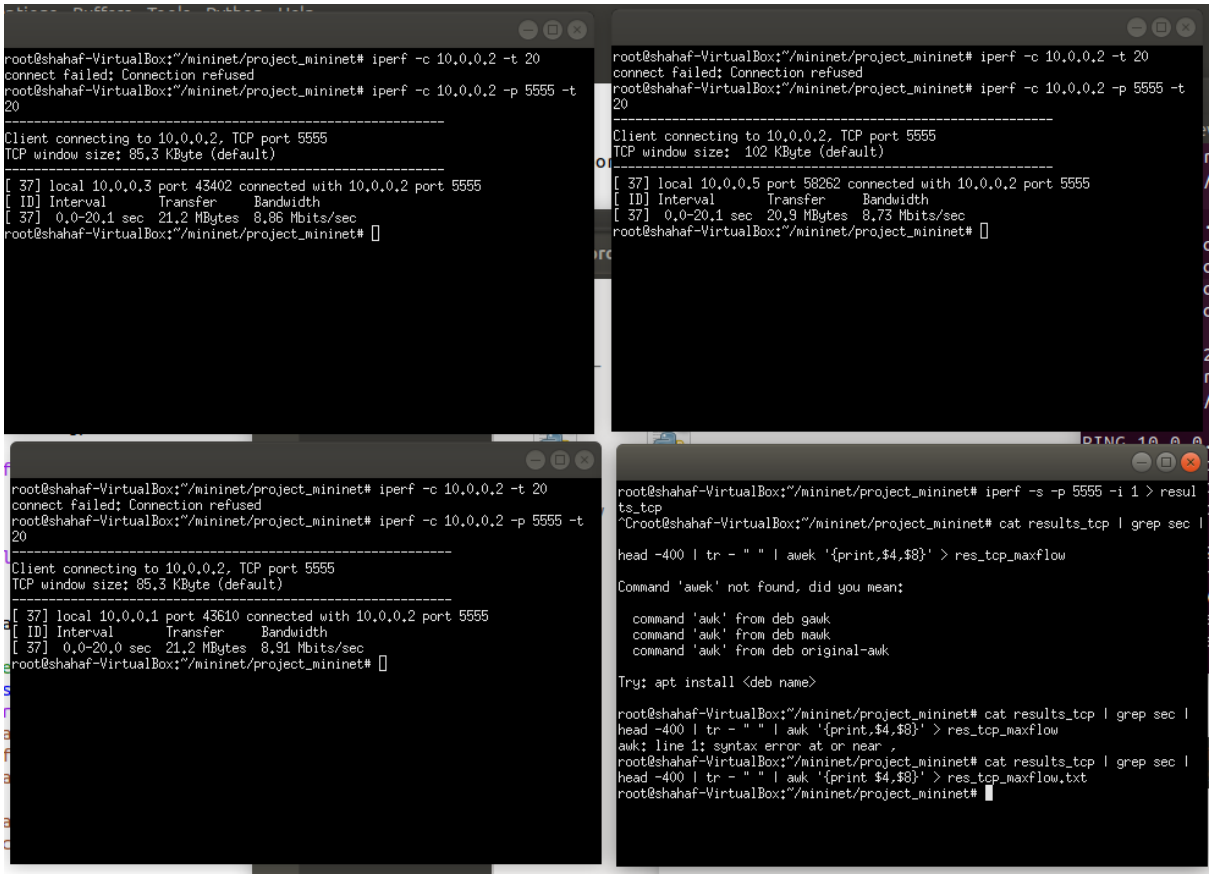
הרצנו xterm מארבעת המחשבים והפעלנו iperf עבור כל מחשב ועבור UDP\TCP ועבור Dijkstra ועבור MF.

שמרנו את התוצאות בקבצים, ואז ניתחנו את הקבצים בפייתון לבנות גרפים של throughputs לצורך ההשוואה.

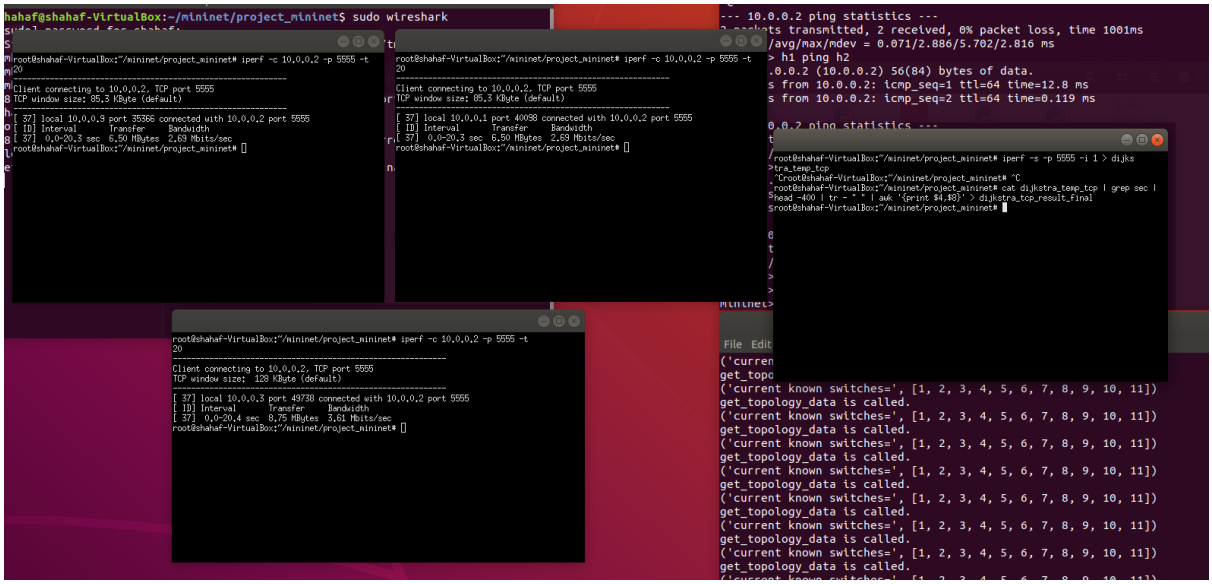
בתמונה השנייה אפשר לראות שאנחנו רואים את כל המסלולים האופטמליים שיש לנו.
בתמונות הבאות למטה מפורט איך ביצענו את ה iperf עם הארגומנטים.

TCP MAX FLOW

הטרמינל הימני למטה מריץ שרת, ולאחר מכן הוא כותב את התוצאות שלו בקובץ שנקרא res_tcp_maxflow שאנחנו ניקח ממנו את הנתונים ונעשה את אותו הדבר עבור כל אחד מהפרוטוקולים UDP\TCP בצורה דומה.

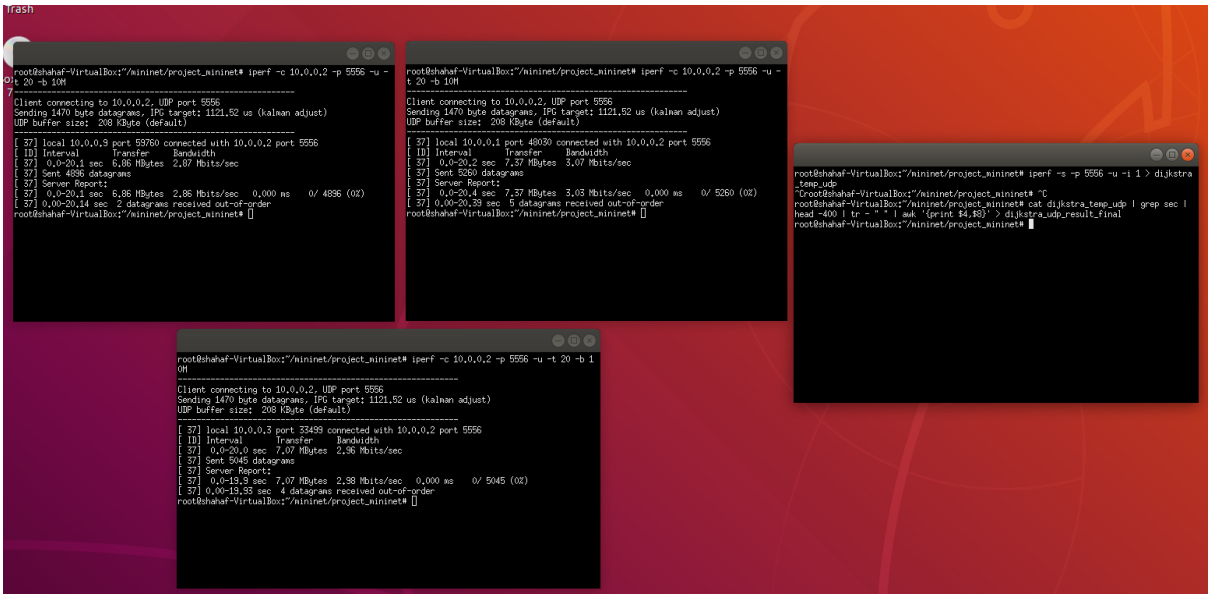


TCP Dijkstra



שלושת הטרמינלים השמאליים מריצים iperf -c עם האייפי של השרת 10.0.0.2 ופורט UDP, אפשר לראות את האופציות -t 20 ו -b 10M שאומרות כמה זמן לשלוח ובאיזה BW להשתמש(אין לנו טעם להשתמש ביותר מ10 מכיוון שזה ה bottleneck עבור כל מסלול בטופולוגיה שלנו)

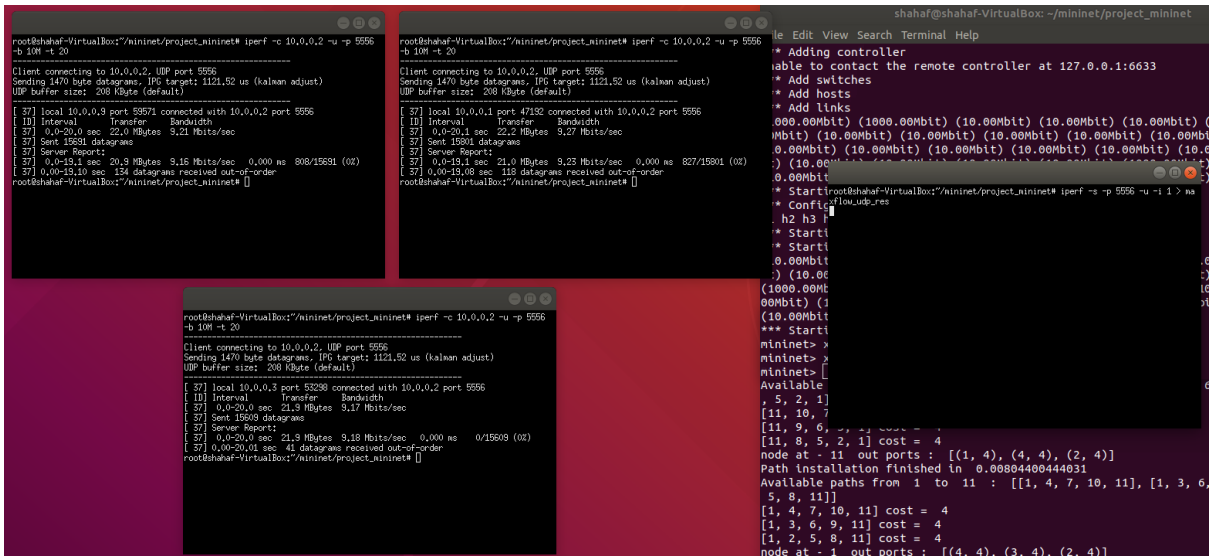
:UDP Dijkstra



שלושת הטרמינלים השמאליים מריצים iperf -c עם האייפי של השרת 10.0.0.2 ופורט UDP, אפשר לראות את האופציות -t 20 ו -b 10M שאומרות כמה זמן לשלוח ובאיזה BW להשתמש(אין לנו טעם להשתמש ביותר מ10 מכיוון שזה ה bottleneck עבור כל מסלול בטופולוגיה שלנו)

:UDP Max Flow

שלושת הטרמינלים השמאליים מריצים iperf -c עם האייפי של השרת 10.0.0.2 ופורט UDP, אפשר לראות את האופציות -t 20 ו -b 10M שאומרות כמה זמן לשלוח ובאיזה BW להשתמש(אין לנו טעם להשתמש ביותר מ10 מכיוון שזה ה bottleneck עבור כל מסלול בטופולוגיה שלנו)

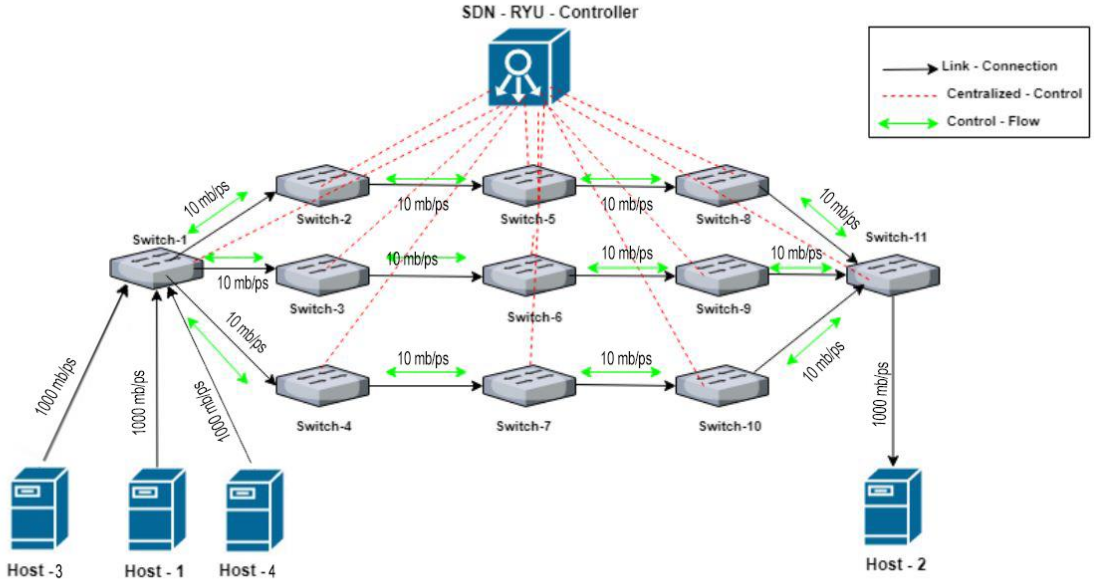


הטופולוגיה שלנו:

הטופולוגיה מכילה רשת סוויצים עם BW=10 mb\ps ו3 מחשבים שמחוברים לסוויץ' 1 שהקיבולת שלהם 1000 mb\ps וגם עבור switch 11 שמחובר ל host 2 יש bandwidth=1000 mb\ps.

הבקר הוא מסוג ryu controller וכל הסוויצ'ים משתמשים ב openVswitch וגרסת הבקר היא OpenFlow 1.3

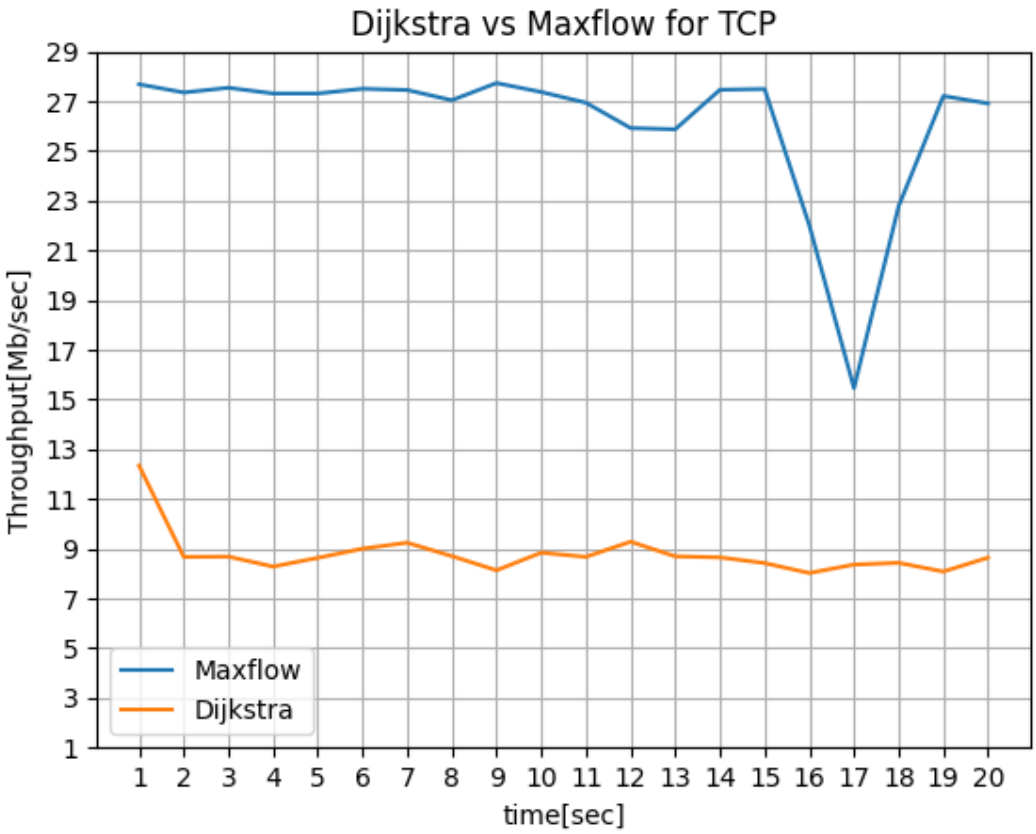
את הניסוי הרצנו ב host2=iperf server ו host1,host3,host4=iperf clients.



תוצאות:

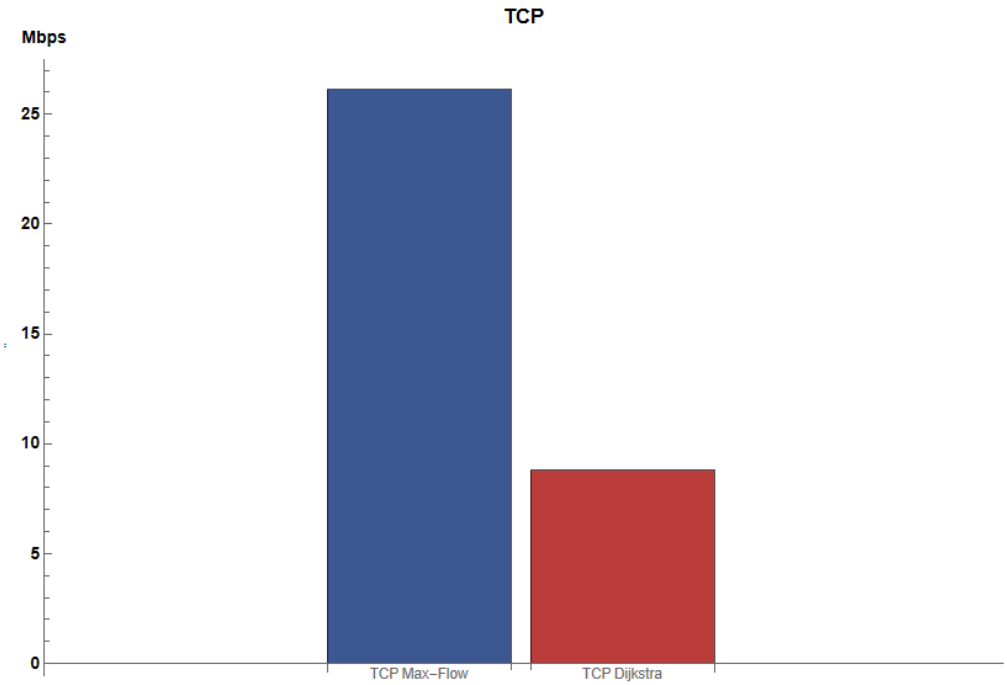
לקחנו את התוצאות שקיבלנו מ iperf ושמנו בקבצים, לאחר מכן עברנו עליהם בקובץ פייתון בשביל לסכום את ה throughput בכל רגע וסכמנו,כדי לראות את ה throughput הכולל והשוונו בין 2 האלגוריתמים.

עבור TCP :

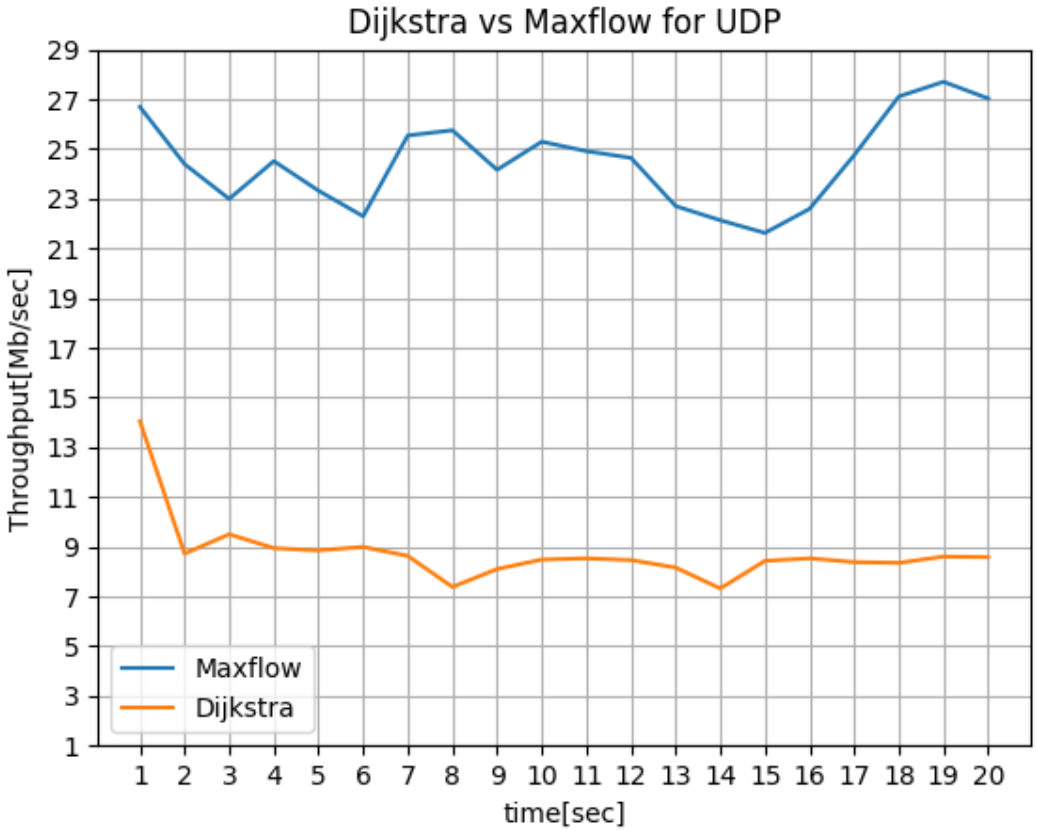


כפי שאפשר לראות וכפי שמצופה, ה throughput של max flow גבוהה בהרבה, מכיוון שאנחנו שולחים מידע כמה שיותר, אבל ב3 מסלולים (S1-S5-S8-S11, S1-S6-S9-S11,S1-S7-S10-S11) ולכן אנחנו מנצלים את ה bottleneck בכל אחד מהם, במקום רק באחד מהם.

אפשר לראות שהקצב הממוצע עבור tcp maxflow הוא בערך 25 mb/s ועבור Dijkstra הוא בערך 9 mb/s

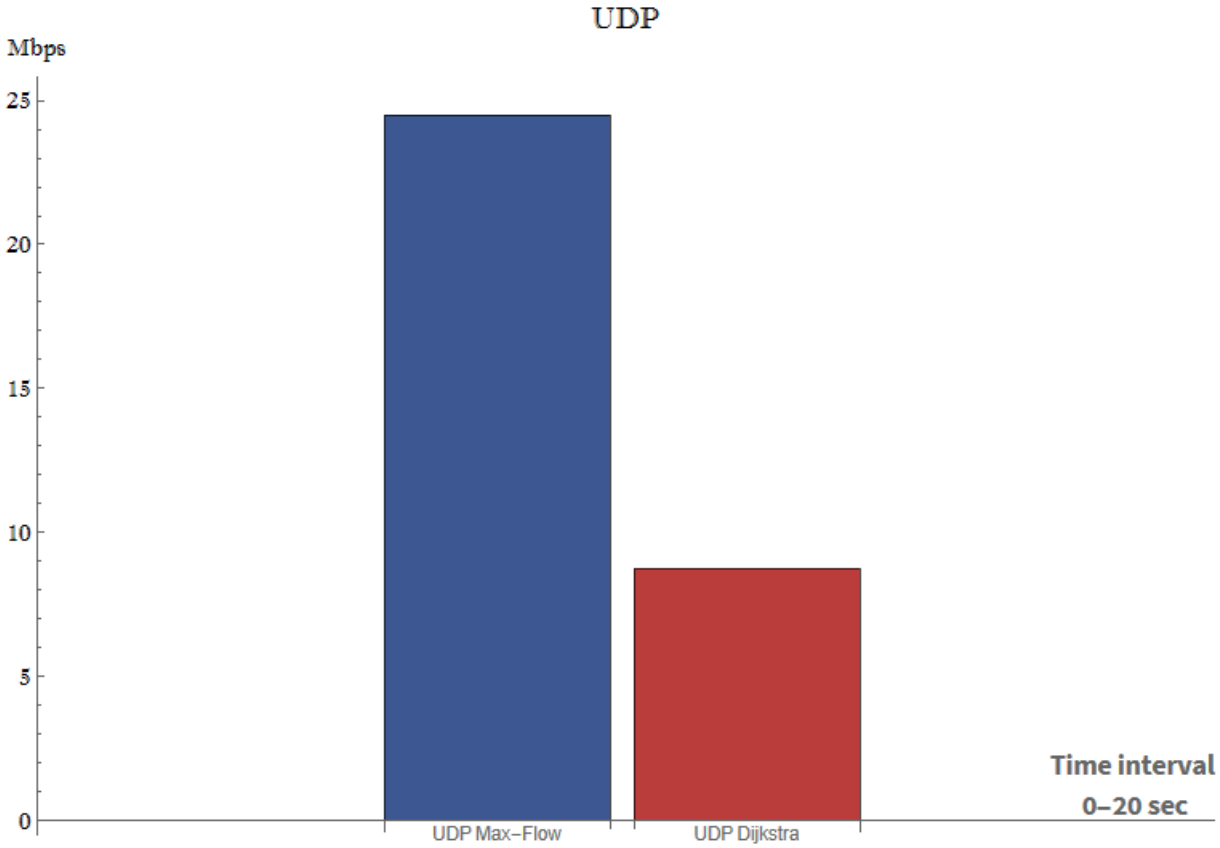


עבור UDP :



כפי שאפשר לראות וכפי שמצופה, ה throughput של max flow גבוהה בהרבה, מכיוון שאנחנו שולחים מידע כמה שיותר, אבל ב3 מסלולים (S1-S5-S8-S11, S1-S6-S9-S11,S1-S7-S10-S11) ולכן אנחנו מנצלים את ה bottleneck בכל אחד מהם, במקום רק באחד מהם.

אפשר לראות שהקצב הממוצע עבור udp maxflow הוא בערך 25 mb/s ועבור Dijkstra הוא בערך 9 mb/s

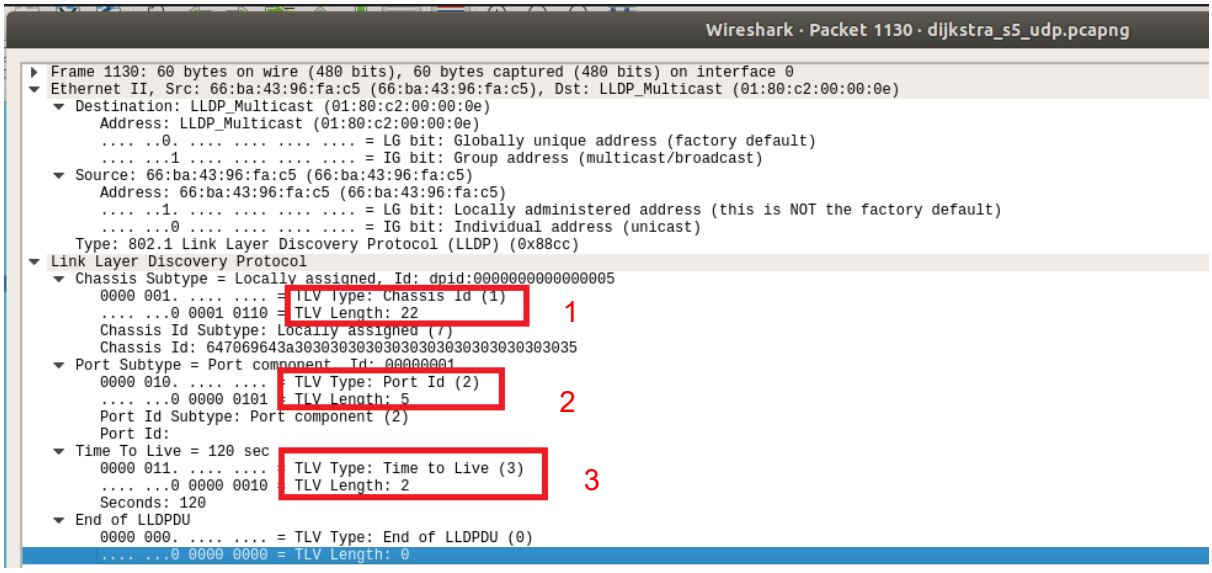


ניתוח חבילות ברשת (Wireshark)

LLDP -Link Layer Discovery Protocol

פרוטוקול CDP וLLDP הם פיצ'רים שמשתמשים כדי לאסוף מידע ברשת. CDP עובד על מנת לאתר מכישירי סיסקו שמחוברים ברשת, נאמר שסוויצ' מחובר לראוטר, הראוטר שולח CDP Advertisements או LLDP-Multicast, פרסומים אלו משתפים מידע על סוג המכשיר שהתגלה,שמו, הדגם, גרסת הפעלה, ואילו פורטים יש לו. השוני היחידי הוא שCDP שייך רק לחברת סיסקו הוא ברמת Layer2 והוא משמש על מנת לאסוף מידע על מכישירי סיסקו שמחוברים ברשת. גם LLDP עם אותו שימוש וגם נמצא ברמת Layer2 .

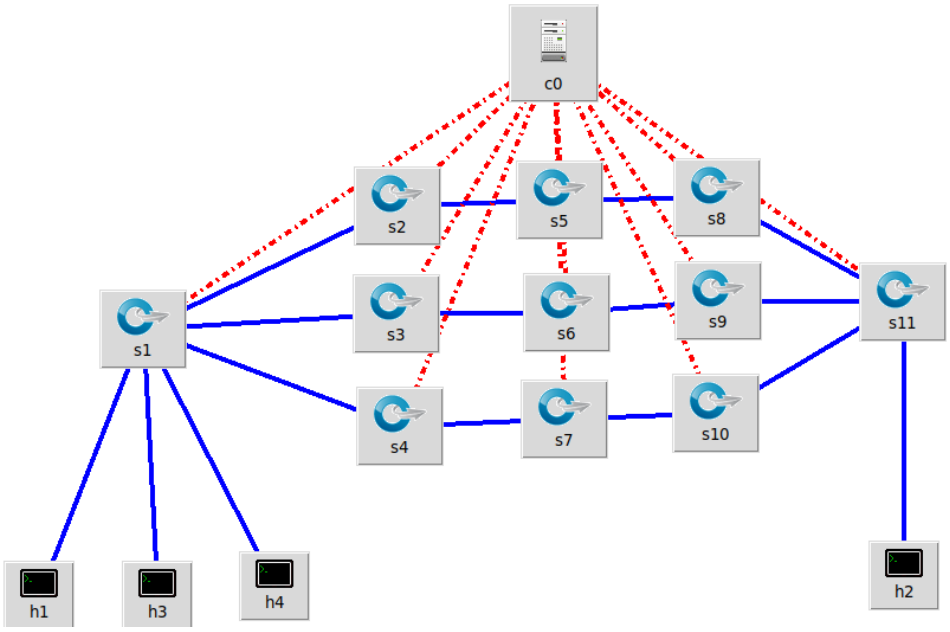
תוכן החבילה של LLDP

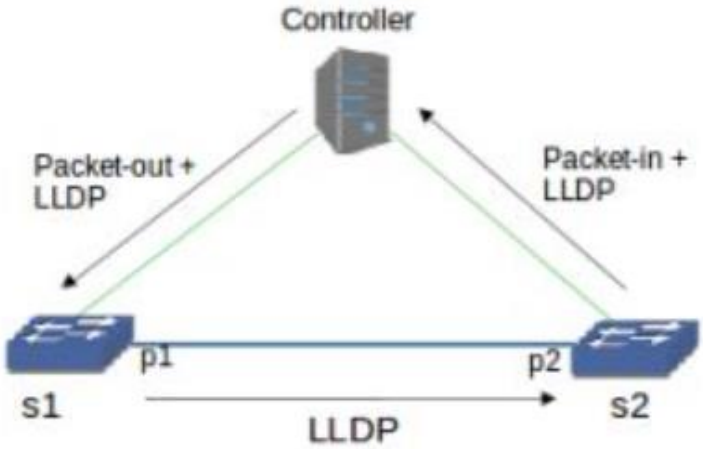


איך זה קשור?

ב OpenFlow כדי לקבל מידע, ה controller צריך לגשת לכל הסוויצ'ים בלינקים חבויים(הם לא נראים בטופולוגיה)

- כאן שחף שם תמונה של הטופולוגיה ב מיניאדיט





הוא ניגש אליהם באמצעות פרוטוקול שנקרא OFDP – OpenFlow Discovery Protocol שלמעשה מה שהוא עושה זה אנקפסולציה לחבילות LLDP, הבקר שולח הודעה לאחד הסוויצ'ים, הסוויץ' ניגש לסוויץ' אחר באמצעות LLDP ולאחר מכן הסוויץ' השני ניגש אל הבקר חזרה, ובכך הבקר מכיר לאט לאט בכל הסוויצ'ים והלינקים בטופולוגיה.

LLDP Ethernet frame structure

Preamble	Destination MAC	Source MAC	Ethertype	Chassis ID TLV	Port ID TLV	Time to live TLV	Optional TLVs	Optional End of LLDPDU TLV	Frame check sequence
	01:80:c2:00:00:0e, or 01:80:c2:00:00:03, or 01:80:c2:00:00:00	Station's address	0x88CC	Type=1	Type=2	Type=3	Zero or more complete TLVs	Type=0, Length=0	

מבנה החבילה כולל :

- Destination MAC - כתובת ייעודית של מולטיקאסט שכתובה ב 802.1d וכל bridge שמכיר ב RFC אמור להכיר אותה ולדעת את זה, כדי לא להעביר את ההודעה הלאה.
- Source MAC - כתובת המקור של מי ששולח את ההודעה, בציור למעלה למשל זה הנתב השמאלי.
- EtherType – קבוע כמו בתמונה
- Chassis ID ,Port ID, TTL TLVS - שלושה TLV שהם שדות חובה,לשלושתם יש 7 ביטים של TYPE שרשום למעלה, 9 ביטים של LENGTH ועוד 0-511 בתים של Value.
- Chassis ID– מזהה של MAC או כתובת IP.
- Port ID - מזהה של הפורט הספציפי (מבחינת INTERFACE PORT)
- TTL – זמן שהחבילה שהיא למעשה חבילת ETHERNET תטייל לכל היותר, משום שאנחנו עובדים בשיטת CRAWLING אבל לא רוצים שהחבילה תטייל ברשת בלוף אינסופי.

לאחר מכן, אפשר לשים OPTIONAL TLVS אבל במקרה שלנו הם לא עוברים, ולכן אנחנו משתמשים רק בשדות ההכרחיים.

בתמונה שלנו אפשר לראות שאנחנו שולחים 60 בתים של הודעה .

המידע הזה נשמר ב MIB (Management information base)של כל נתב ואנחנו יכולים לגשת אליו.

הערה חשובה:

כדי לגשת אל המידע הזה, למדנו בהרצאות שאנחנו יכולים להשתמש בפרוטוקול שנקרא SNMP כדי לגשת מרחוק אל המידע, אבל Mininet דואג לעשות אבסטרקציה לזה ולכן אנחנו לא רואים הודעות כאלה בלינקים, משום שהם מחוברים לבקר.

IPERF

Iperf הוא כלי למדידת ביצועי הרשת ותפוקה (Throughput). זהו כלי שתומך במגוון פלטפורמות ויכול ליצור מדידות ביצועים תקניות עבור כל רשת Iperf. מציע פונקציות של לקוח ושרת, ויכול ליצור זרמים נתונים כדי למדוד את הנפח הכללי בין השניים בכיוונים אחד או שניים. תוצאות Iperf טיפוסיות כוללות דוח מצופה בזמן של כמות הנתונים שהועברו והשיעור המדוד.

Iperf תומך ב TCP וגם בUDP כמו שאפשר לראות בניסוי שערכנו. בניסוי שלנו השתמשנו באפשרויות 5555 p- שמציין שאנחנו רוצים להשתמש ב TCP PORT 5555. כמו כן השתמשנו באופציה של u- שמאפשרת לנו להשתמש ב UDP בצורה דומה.
האופציות s- c- מציינות בהתאמה client\server מכיוון שאנחנו שולחים מהCLIENT אל ה SERVER חבילות.
האופציות n- מציינות את גודל המידע שאנחנו שולחים (לא רצינו לשלוח מעט מדי מידע משום שכך לא ננצל את ה BW של הלינקים) האופציות 20 t- מציינת שאנחנו שולחים במשך 20 שניות את כמות המידע המירבית, בחרנו 20 שניות כי זה זמן טוב מספיק למדוד ממוצע בצורה הוגנת, ולמרות ההתחלה הלא סינכרונית הממוצע יוצא דומה.
לבסוף השתמשנו גם באופציה 1 I- שגורמת לשרת לקחת את כל התוצאות ולשמור אותם בהפרש של שנייה אחת, זה הקל עלינו בכך שאת התוצאות אנחנו רושמים לתוך קובץ, ורצינו הפרשי זמן קבועים למען פשטות ולמען הגינות בהשוואות של UDP\TCP .