

שב 4 - תקשורת באינטרנט

המתרגל האחראי על התרגיל: מתן פוגץ' matan.pugach@cs.technion.ac.il

ההגשה בזוגות. אלקטרונית באתר.

תאריך הגשה: 11/6/19 ב23:55

SOCKET PROGRAMMING

במעבדה זו אנו הולכים לממש Proxy Load Balancer. אם אתם עוד לא ממש יודעים מה זה Load Balancer ואיך הוא עובד, זו הזדמנות טובה ללמוד. ועל הדרך זו תהיה סיבה טובה להתנסות בSocket Programming שכבר למדנו.

אז מה זה Load Balancer?

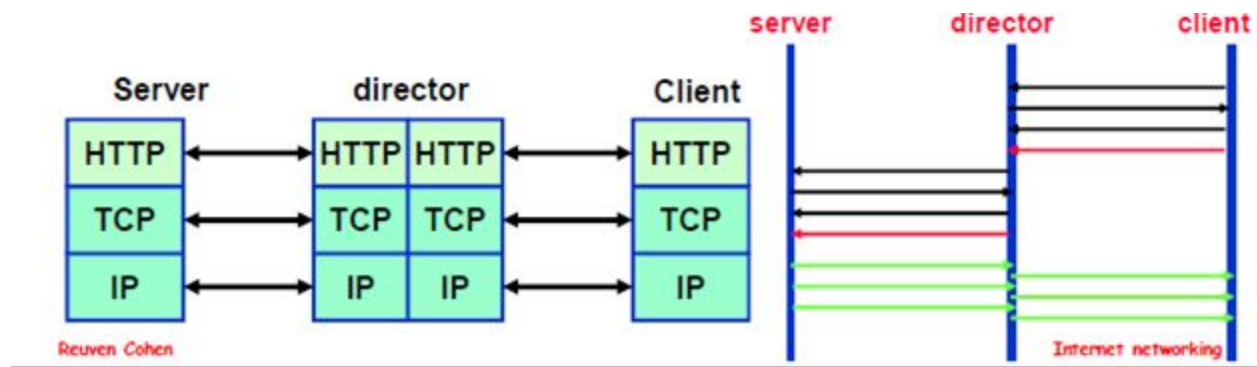
Load Balancing הוא בעצם תהליך של ביזור עבודה בין כמה משאבי מחשוב שונים (שרתים בדר"כ). לכן, Load Balancer, כשמו כן הוא, הוא רכיב רשתי שאחראי על תהליך זה. איך בדיוק הוא מבצע זאת? יש מספר דרכים שונות, שפועלות בשכבות שונות, ולכל אחת יתרונות וחסרונות משלה. את הפירוט על הדרכים השונות נשאיר להרצאה.

אבל בתרגיל זה אתם הולכים לממש Proxy Load Balancer, ולכן נתעכב רגע על דרך פעולה זו.

בשביל לפשט את ההסבר בואו נתבונן בבעיה מוחשית, נניח שאתם מנהלי הרשת של חברת תוכן כלשהי, לחברה יש 5 לקוחות. ובעקבות תלונות על איטיות ברשת, החברה שלכם החליטה להתחדש ב2 שרתים נוספים (עד עכשיו היה לה שרת אחד). עליכם הוטלה המשימה של התקנת השרתים החדשים, ומהר מאוד הבנתם שאתם בבעיה, והבעיה היא שאצל כל הלקוחות כבר מותקנת תכנת לקוח שיודעת לעבוד מול שרת אחד בלבד, ומכירה רק את הכתובת של שרת זה (נניח 10.0.0.1).

כלומר, מצד אחד צריך שהלקוחות ימשיכו להתחבר לכתובת הIP שהם מכירים, ומצד שני צריך לגרום לבקשות שלהם להגיע ולהיות מטופלות ע"י 3 שרתים שונים. אז מה עושים? הפתרון המתבקש הוא להוסיף מחשב (בואו נקרא לו Load Balancer או בקיצור LB) שהלקוחות ישלחו את הבקשות שלהם אליו, הוא ישב בכתובת של השרת המקורי, ושהוא ידאג כבר להעביר את הבקשות לשרתים (ולהחזיר את התשובות כמובן).

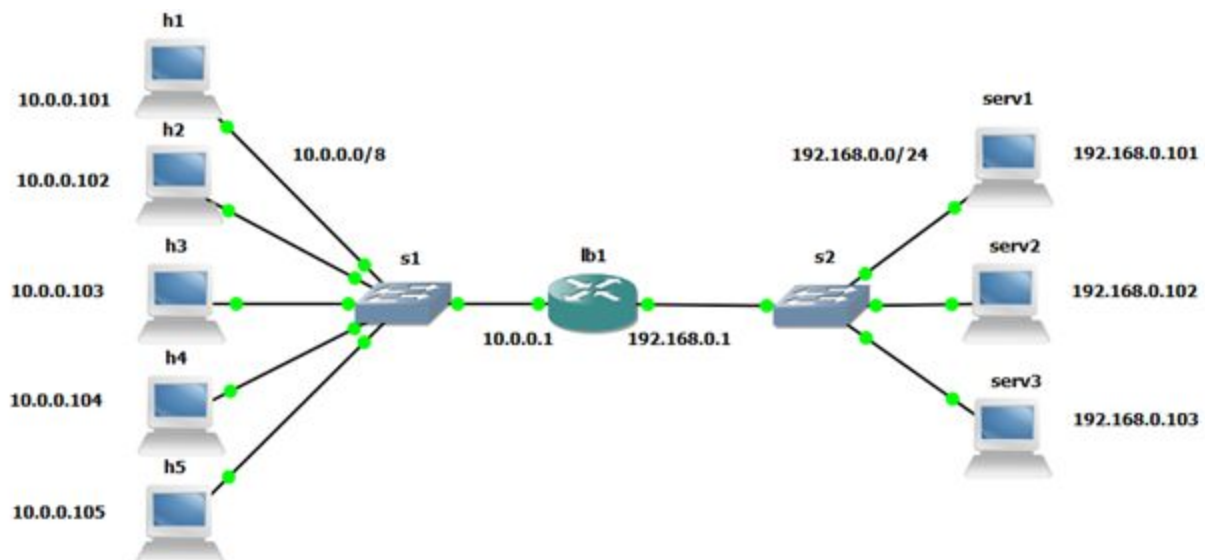
ואיך בדיוק זה אמור להתבצע? אז אמרנו כבר שיש כמה דרכים, כדי לתרגל קצת Socket Programming אנחנו נבחר בדרך שנקראת Proxy Load Balancing. אם נסתכל על הציור הרלוונטי מההרצאה:



אולי נוכל להבין יותר טוב מה בעצם Proxy LB עושה (בצורה מאוד מופשטת), דבר ראשון הLB מתחבר לשרתים, לאחר מכן הLB יאזין לבקשות מהלקוחות, ברגע שתגיע בקשה הוא יבחר את אחד השרתים (השרת יבחר לפי מדיניות תזמון כלשהי) יעתיק את תוכן הבקשה, ישלח אותה ויחכה לתשובה. ברגע שתגיע תשובה הוא ישלח את התשובה על החיבור של הלקוח ויסגור אותו, שימו לב שאין צורך לפתוח ולסגור את החיבורים בין הLB לשרתים כל פעם מחדש.

עכשיו אנחנו יכולים לעבור לדבר על התרגיל.

בתרגיל נתונה טופולוגיה בmininet שנראית כך:



ישנם 5 לקוחות, שנמצאים על רשת 10.0.0.0/8. ו 3 שרתים שנמצאים על רשת 192.168.0.0/24 המחוברים ביניהם בעזרת מתגים (s1, s2).

בנוסף ישנו LB אחד, שמחבר בין שתי הרשתות, שאת הלוגיקה שלו עליכם יהיה לממש (הקוד של השרתים והלקוחות נתון), שימו לב של LB יש שתי כתובות IP שונות.

התחילו בהורדת הקבצים הדרושים לתרגיל ע"י הרצת הפקודה:

git clone <https://github.com/cs236341/loadBalancer.git>

מתיקית הבית של המכונה הוירטואלית שלכם (אחרי שחיברתם את המכונה לאינטרנט כמובן).

כעת עלינו להתקין חבילת תכנה נוספת הדרושה לשם הרצת המעבדה, הריצו את הפקודה:

sudo apt-get install bridge-utils

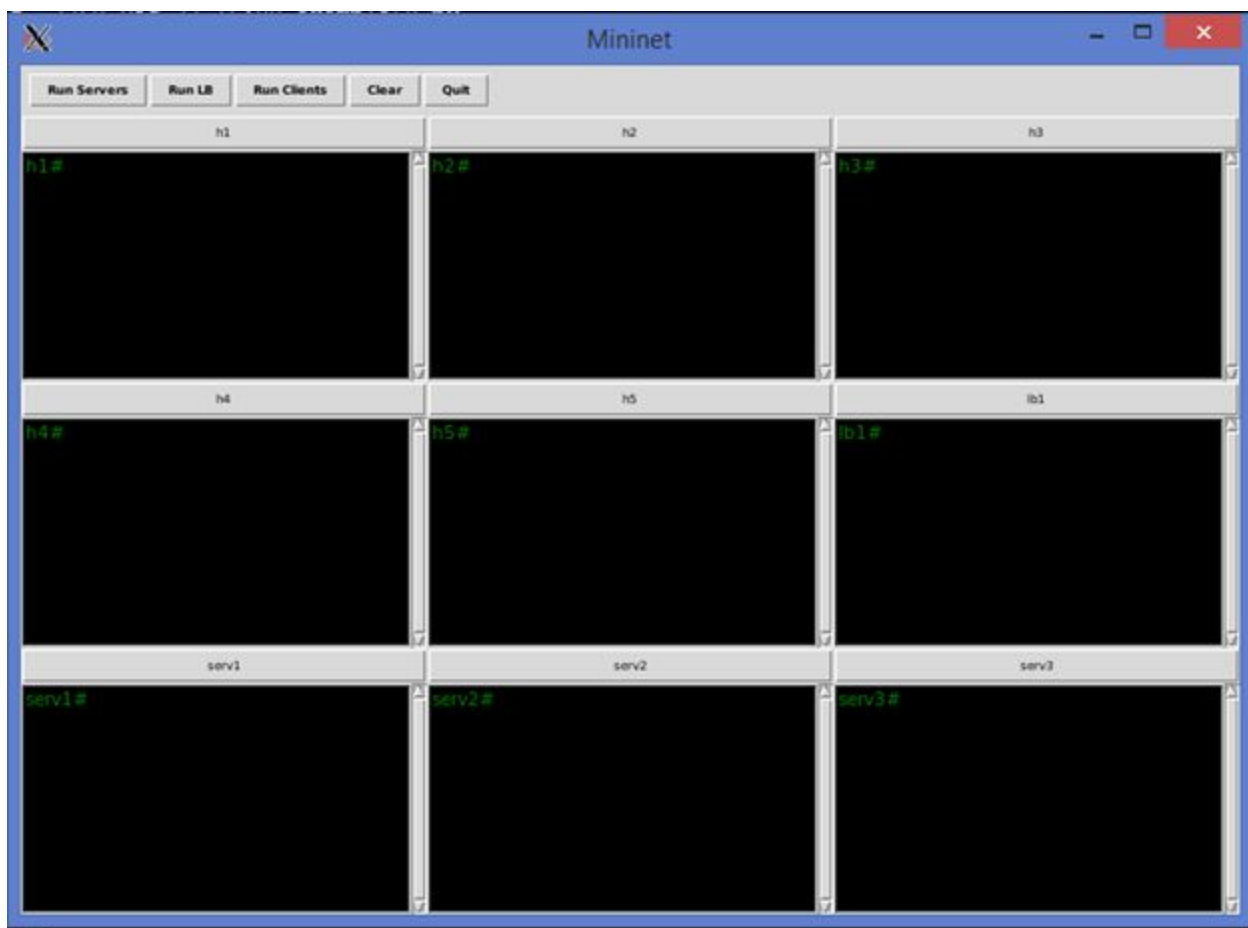
ובצעו את ההתקנה.

(אם נתקלתם בבעיה, נסו להריץ **sudo apt-get update** ואז את הפקודה פעם נוספת)

כעת, ודאו שxming עובד (או כל xserver אחר...) והריצו את הפקודה:

sudo python loadBalancerLab.py

פקודה זו תריץ את mininet שאתם כבר מכירים, ותאתחל את הטופולוגיה המדוברת, אבל מהר מאוד תבחינו במשהו שונה. אם הכל עובד כמו שצריך החלון הבא אמור להפתח:



על מנת להקל על העבודה, חלון זה מכיל בתוכו shell לכל אחד מהמחשבים ברשת (מומלץ להגדיל את החלון). מעל כל shell ישנו כפתור שפותח xterm אל המכונה הרלוונטית. בנוסף, ישנם כפתורים בחלק העליון שמקלים על הרצת המעבדה:

לחצו על הכפתור **Run Servers**, אתם אמורים לראות הודעות בשלושת החלונות של השרתים, המבשרת על תחילת עבודה וסוג השרת.

לחצו על הכפתור **Run LB**, אתם אמורים לראות הודעה בחלון lb1 של LB.

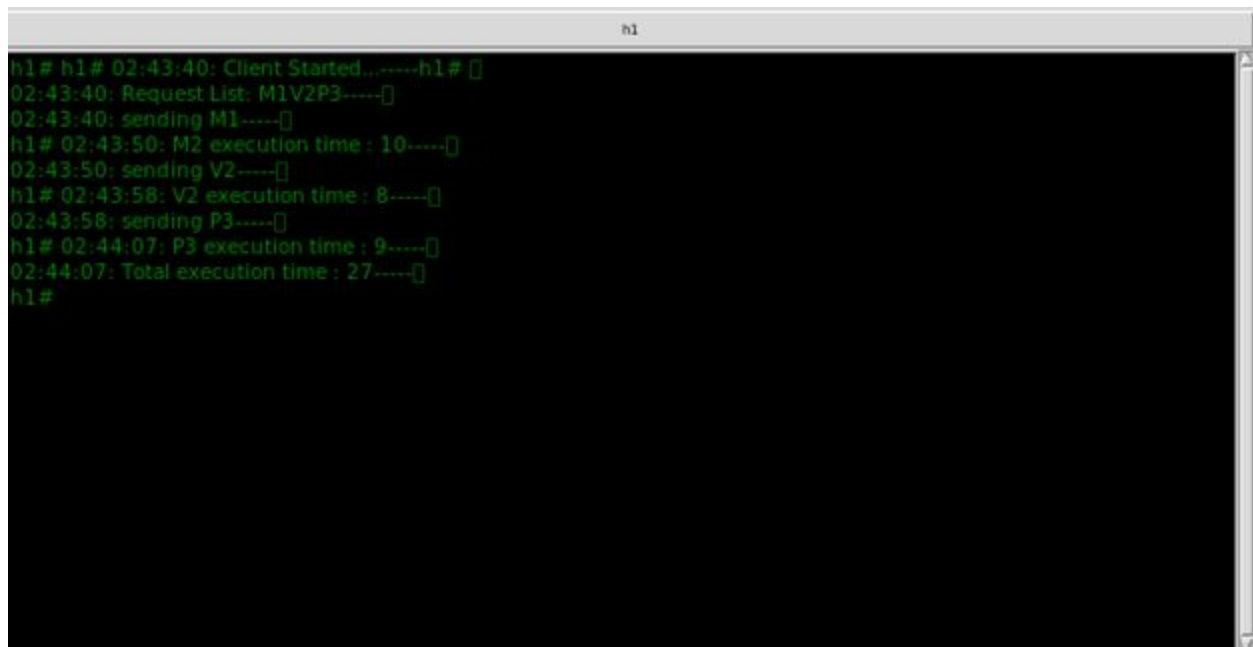
לחצו על הכפתור **Run Clients**, כעת אתם אמורים לראות מלא הודעות על בקשות שנשלחות ומתקבלות בכל החלונות. אחרי בערך 25 שניות הכל אמור להרגע.

מה בעצם ראינו? בשלב הראשון, הפעלנו תכנה של שרת על כל אחד ממחשבי השרת, יש שני שרתים המיועדים לוידאו, ושרת אחד המיועד למוזיקה.

לאחר מכן הפעלנו את Load Balancer, אם לרגע דאגתם שלא תשאר לכם עבודה זהו LB בסיסי המשמש להדגמה, והקוד שלו לא מסופק.

בשלב האחרון הפעלנו במקביל, על כל אחד מהלקוחות, תכנה של לקוח. הלקוחות שלחו בקשות לLB, שהעביר אותן לשרתים (במדיניות תזמון Round Robin), השרתים טיפלו בכל בקשה בתורה והחזירו תשובה לLB.

בואו נתבונן לעומק על הפלט מלקוח 1:



```
h1# h1# 02:43:40: Client Started.....h1# []
02:43:40: Request List: M1V2P3-----[]
02:43:40: sending M1-----[]
h1# 02:43:50: M2 execution time : 10-----[]
02:43:50: sending V2-----[]
h1# 02:43:58: V2 execution time : 8-----[]
02:43:58: sending P3-----[]
h1# 02:44:07: P3 execution time : 9-----[]
02:44:07: Total execution time : 27-----[]
h1#
```

הוא מדווח לנו שהוא התחיל ושרשימת הבקשות שלו היא M1V2P3
לאחר מכן הוא שולח כל בקשה בנפרד (עבור כל בקשה הוא יוצר חיבור חדש), ומדווח כמה זמן לקח לבצע אותה (כמה זמן לקח לתשובה להגיע).
לבסוף הוא מציין כמה זמן לקח לבצע את כל הבקשות ברשימה.

אם נתעכב רגע על מבנה של בקשה, נראה שכל בקשה מורכבת משני תווים, אות גדולה וספרה:

- האות M - מציינת בקשה לתוכן מסוג מוזיקה.
- האות V - מציינת בקשה לתוכן מסוג וידאו.
- האות P - מציינת בקשה לתוכן מסוג תמונה.
- הספרה מציינת את אורך הבקשה, בשניות, ניתן להניח כי לא ינתנו בקשות באורך קטן מ 1 שניה ולא ינתנו בקשות באורך גדול מ 9 שניות (אורך הבקשה יהיה תו אחד בלבד).

את רשימת הבקשות של h1 ניתן לשנות ע"י שינוי הרשימה בקובץ h1.in (על הרשימה להיות בפורמט המתאים, ארוכה כרצוננו) וכך ניתן לעשות גם לשאר הלקוחות.

כעת, בואו נתבונן בפלט של שרת 1:

```

serv1# serv1# 02:43:36: Server Started.....serv1# []
serv1# 02:43:36: Server Type: VIDEO.....serv1# []
serv1# 02:43:37: LB Connected.....[]
serv1# 02:43:40: recieved V1, execution time will be 1.0.....[]
serv1# 02:43:41: finished executing V1, sending response-----serv1# []
02:43:41: recieved M1, execution time will be 2.0.....[]
serv1# 02:43:43: finished executing M1, sending response-----serv1# []
02:43:43: recieved P2, execution time will be 2.0.....[]
serv1# 02:43:45: finished executing P2, sending response-----serv1# []
serv1# 02:43:45: recieved V3, execution time will be 3.0.....[]
serv1# 02:43:48: finished executing V3, sending response-----[]
serv1# 02:43:49: recieved V4, execution time will be 4.0.....[]
serv1# 02:43:53: finished executing V4, sending response-----[]
serv1#

```

אנו רואים שהשרת הוא מסוג וידאו (serv1, serv2), לכן אם נתבונן בבקשות מסוג וידאו או תמונה, אורך הבקשה שווה לזמן הביצוע. לעומת זאת עבור בקשות מסוג מוזיקה, זמן הביצוע יהיה פי 2 מאורך הבקשה שנשלחה.

בצורה דומה, עבור שרת מסוג מוזיקה (serv3), זמן הביצוע של בקשות מוזיקה שווה לאורך הבקשה, אולם זמן הביצוע של בקשות תמונה יהיה פי 2 מאורך הבקשה, וזמן הביצוע של בקשת וידאו יהיה פי 3 מאורך הבקשה המקורית.

נסכם זאת בטבלה המציגה פי כמה יתארך זמן הביצוע:

Server Type \ Request type	M (Music)	V (Video)	P (Picture)
VIDEO	X2	X1	X1
MUSIC	X1	X3	X2

אז מה אתם צריכים לעשות בתרגיל?

אתם צריכים להשתמש במה שלמדתם על Socket Programming, על מנת לבנות Proxy LB מהיר ככל שתוכלו. הלקוחות והשרתים פועלים בצורה סדרתית (ה-LB לא...) חופש הפעולה שלכם הוא להחליט איזו עבודה לשלוח לאיזה שרת ובאיזה תזמון. ה-LB צריך להתחבר לכל השרתים בתחילת הריצה שלו, ולהשתמש בחיבורים אלה לכל משך עבודתו (השרתים לא מוכנים לקבל חיבור נוסף) כל פעם שלקוח רוצה לשלוח בקשה, הוא יפתח חיבור חדש וישלח את מחרוזת הבקשה.

איך נמדדת המהירות של ה-LB?

זמן הביצוע ימדד בתור הזמן שלקח ללקוח האחרון לסיים לבצע את כל הבקשות שלו. חלק בלתי נפרד מהתרגיל, בנוסף למימוש הsocket, יהיה לממש מדיניות תזמון. אתם יכולים ללמוד עוד על הנושא ב:

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

זה תחום שניתן לעשות בו תואר שלם. אך ההגיון הבריא שלכם מסוגל לספק מדיניות תזמון טובה מספיק. אנחנו לא מצפים מכם לממש אלגוריתמי תזמון מסובכים.

באיזו שפה עליכם לכתוב?

אתם יכולים לכתוב בכל שפה (python, C++, C, נתמכים ע"י המכונה בלי בעיה). אם כרגע המכונה לא תומכת בשפה שבחרתם, התייעצו איתי לפני תחילת המימוש.

איך זה יתבצע טכנית?

הכפתור Run LB מריץ סקריפט bash שנקרא lb.run שנמצא בתיקיה של התרגיל. עליכם לשנות את קובץ ההרצה שנמצא שם לקובץ של ה-LB שלכם.

איך ינתן הציון?

עיקר הציון ייקבע לפי כמה מהיר ה-LB שלכם. נריץ סדרות שונות (חשבו על מקרי קצה אפשריים) של קלטים עם ה-LB שלכם (הסדרות יהיו ארוכות וירוצו כמה פעמים על מנת לקבל מיצוע טוב ולבטל כמה שניתן אי-דטרמיניזם של הריצות) וניקח את הזמן הממוצע שלקח לו לבצע את העבודות. ציון מינימאלי ינתן לexampleLB המסופק, שמבצע Round Robin. ה-LB שאתם תממשו צריך לעשות עבודה טובה לפחות כמוהו, והציון שתקבלו יהיה יחסי לביצועי ה-LB שלכם (עם דיסקרטיזציה כלשהי) ושל שאר הכיתה.

בנוסף -

מכיוון שכתובה ב-C דורשת יותר זמן משפות אחרות, בונס של 5 נקודות ינתן למי שיכתוב בשפת C.

מה עליכם להגיש?

עליכם להגיש קובץ zip המכיל:

1. קובץ **lb.run** המעודכן שלכם.
2. תיקיית **code** / המכילה את קבצי הקוד וכל הקבצים הדרושים להרצה (בינאריים וכו').
3. קובץ **lb.pdf** המפרט בקצרה את מדיניות התזמון שלכם.
4. קובץ **id.txt** המכיל את השמות שלכם ואת מספרי תז בפורמט Israel Israeli 123456789.

הערות (יתווספו הערות במהלך התרגיל – שימו לב שאתם מבקרים פה מדי פעם):

- יש הרבה טיפוסים מוכנים בהרבה מהשפות השונות ודוגמאות והסברים באינטרנט, מותר ורצוי להשתמש בהם (לדוגמא <http://beej.us/guide/bgnet>).
- הלקוחות שולחים את הבקשות והשרתים מאזינים על פורט 80, אבל הם לא משתמשים בפרוטוקול HTTP. המידע שמכילה בקשה\תשובה הוא רק המחרוזת המתוארת.
- הבדיקה והציון היא לא תחרותית (בין הסטודנטים) אלא בין המימוש שלכם למימוש אופטימלי של ה Load Balancer.
- מטרת התרגיל היא הכרה ושימוש בSocket API שהוסבר בתרגול על Socket Programming . משמעות הדבר היא שמצופה מכם לעשות שימוש מפורש בAPI (שימוש בsocket accept\connect\bind\send\recieve וכו'..). ולא שימוש ב'האקים' העוטפים את זה (למשל, שימוש בThreadedTCPServer שלמעשה מוציא הרבה מהפואנטה של התרגיל). התרגיל יבדק באופן יבש כדי לראות אם חסרים שימושים מפורשים בapi במקומות הדרושים.