תקשורת ומחשוב – פרויקט סיום

הגשה – 1.8.21

בדיקות – 8-12.8.21

הוראות כלליות

1) את המטלה יש להגיש עד התאריך המצויין בתיבת ההגשה

2) כל הקבצי המטלה (קוד, פלט תעבורה, הסבר) כולל הסברים שלכם והקלטות **wireshark** דחוסים לקובץ ZIP ששמו הוא מס' ת.ז. של המגישים עם קו תחתון בינהם ID_ID.

3) הגשה בזוגות אפשרית.

4) מותר לכם להשתמש בכל החומר שנמצא במודל כולל קוד בתרגולים. חומרים אחרים אין אפשרות. כמובן שאפשר להעזר באינטרנט להבנה של תהליכים וקוד אבל בשום פנים ואופן לא להעתיק קוד

5) אין איחורים ללא אישור מיוחד של רכז הקורס (עמית), איחור ללא אישור יגרור אפס אוטומטי

6) הגשת העבודות תתבצע דרך מערכת ה Moodle של הקורס (לא דרך האימייל).

7) יש להקפיד על כללי עיצוב הקוד שנלמדו בתואר (נא להקפיד על פלט ברור, הערות קוד במידה ושמות משתנים בעלי משמעות). קוד רץ בלבד יכול לקבל לכל היותר ציון 60, שאר 40 הנקודות זה הסברים שלכם, ידע, קוד קריא וכו.

8) ניתן להגיש תרגילים למערכת מספר בלתי מוגבל של פעמים כאשר כל הגשה דורסת את הקודמת.

9) העבודה הינה אישית של הזוג ואסור לקבל עזרה מאנשים מחוץ לאוניברסיטה או בתוכה לה. אנשים המתקשים ורוצים עזרה יכולים לפנות לצוות הקורס בשעות הקבלה או להעלות שאלה לאתר הקורס.

10) אסור להעביר קטעי קוד בין סטודנטים, להעלות פתרונות או חלקי פתרונות לאתרים ברשת האינטרנט, פורומים או בקבוצות תקשורת שונות.

11) סטודנטים שיעתיקו פתרון, יקבלו 0 בכל המטלות בקורס ונעלה דיווח לוועדת המשמעת המוסדית.

12) לפרויקט תהיה הגנה פרונטאלית צמוד לסוף תקופת בחינות שבזמן ההגנה תצטרכו לענות גם על שאלות שקשורות לתרחיש אליס ובוב.

13) לפרויקט יש מספר חלקים, **כל חלק עומד בפני עצמו כולל שפת התכנות שאתם יכולים לבחור**.

14) במספר חלקים אתם מקבלים קוד שתוכלו להעזר בו, אתם חייבים לדעת להסביר כל שורה בקוד הזה

15) הערות וערעורים על ציוני התרגיל יש להפנות למתרגל/בודק התרגילים תוך שבוע לכל המאוחר מיום פרסום הציונים.

UDP Pinger Lab

In this lab, you will learn the basics of socket programming for UDP in Python. You will learn how to send and receive datagram packets using UDP sockets and also, how to set a proper socket timeout. Throughout the lab, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate.

You will first study a simple Internet ping server written in the Python, and implement a corresponding client. The functionality provided by these programs is similar to the functionality provided by standard ping programs available in modern operating systems. However, these programs use a simpler protocol, UDP, rather than the standard Internet Control Message Protocol (ICMP) to communicate with each other. The ping protocol allows a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

You are given the complete code for the Ping server below. Your task is to write the Ping client.

Server Code

The following code fully implements a ping server. You need to compile and run this code before running your client program.

In this server code, 30% of the client's packets are simulated to be lost. You should study this code carefully, as it will help you write your ping client.

```
# UDPPingerServer.py

# We will need the following module to generate randomized lost packets

import random

from socket import *


# Create a UDP socket

# Notice the use of SOCK_DGRAM for UDP packets

serverSocket = socket(AF_INET, SOCK_DGRAM)

# Assign IP address and port number to socket

serverSocket.bind(('', 12000))


while True:

      # Generate random number in the range of 0 to 10

      rand = random.randint(0, 10)

      # Receive the client packet along with the address it is coming from

      message, address = serverSocket.recvfrom(1024)

      # Capitalize the message from the client
```

```
        message = message.upper()

        # If rand is less is than 4, we consider the packet lost and do not
respond

        if rand < 4:

                continue

        # Otherwise, the server responds

        serverSocket.sendto(message, address)
```

The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if a randomized integer is greater than or equal to 4, the server simply capitalizes the encapsulated data and sends it back to the client.

Packet Loss

UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons. Because packet loss is rare or even non-existent in typical campus networks, the server in this lab injects artificial loss to simulate the effects of network packet loss. The server creates a variable randomized integer which determines whether a particular incoming packet is lost or not.

Client Code (Three parts)

You need to implement the following client program.

The client should send 10 pings to the server. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should get the client wait up to one second for a reply; if no reply is received within one second, your client program should assume that the packet was lost during transmission across the network. You will need to look up the Python documentation to find out how to set the timeout value on a datagram socket.

Specifically, your client program should

(1) send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.)

(2) print the response message from server, if any

(3) calculate and print the round trip time (RTT), in seconds, of each packet, if server responses

(4) otherwise, print “Request timed out”

During development, you should run the `UDPPingerServer.py` on your machine, and test your client by sending packets to *localhost* (or, 127.0.0.1). After you have fully debugged your code, you should see how your application communicates across the network with the ping server and ping client running on different machines.

Message Format

The ping messages in this lab are formatted in a simple way. The client message is one line, consisting of ASCII characters in the following format:

Ping *sequence_number time*

where *sequence_number* starts at 1 and progresses to 10 for each successive ping message sent by the client, and *time* is the time when the client sends the message.

Currently, the program calculates the round-trip time for each packet and prints it out individually. **Modify this to correspond to the way the standard ping program works**. You will need to report the minimum, maximum, and average RTTs at the end of all pings from the client. In addition, calculate the packet loss rate (in percentage).

Another similar application to the UDP Ping would be the UDP Heartbeat. The Heartbeat can be used to check if an application is up and running and to report one-way packet loss. The client sends a sequence number and current timestamp in the UDP packet to the server, which is listening for the Heartbeat (i.e., the UDP packets) of the client. Upon receiving the packets, the server calculates the time difference and reports any lost packets. If the Heartbeat packets are missing for some specified period of time, we can assume that the client application has stopped. Implement the UDP Heartbeat (both client and server). **You will need to modify the given `UDPPingerServer.py`, and your UDP ping client.**

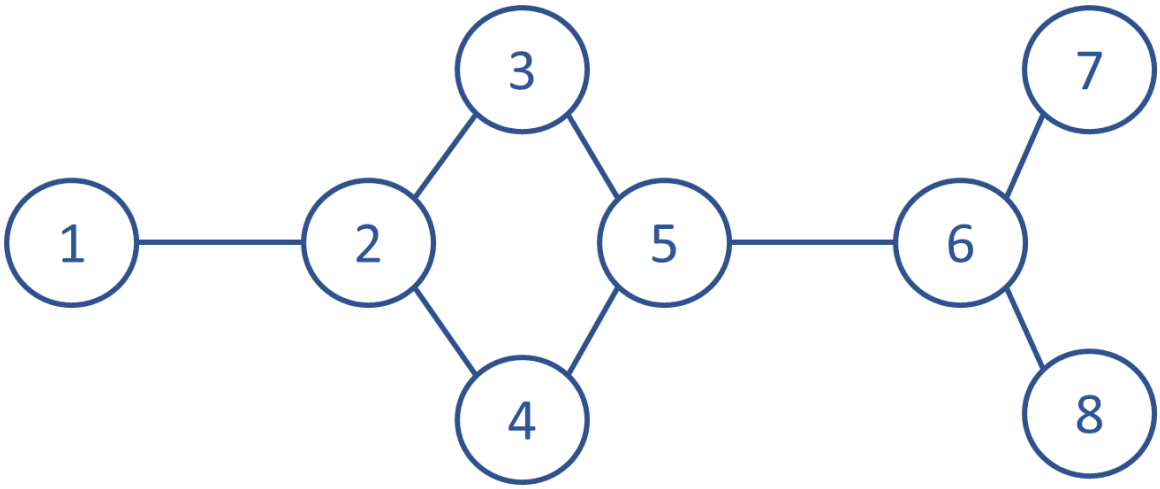
חלק שני: שפת תכנות C or C++ לבחירתכם

1) Open, read and understand Udp-Client.cpp and Udp-Server.cpp
2) Build and run the client and the server
3) Run netstat -a and see the sockets
4) Run the udp-server andre-run netstat -a, observe the newly added UDP-socket at port 5060.
5) Running the udp-server and run Wireshark to capture, see the UDP-datagrams passed and look at Application data passed UDP and IPv4 headers.
6) Convert the program to UDP over IPv6
7) Explain what are the main difference between IPv4 and IPv6 headers.

חלק שלישי : שפת תכנות C or C++

עליכם לפתח צומת (node) ברשת שמטרתה לאפשר תקשורת בין צמתים שונים. הצמתים לא בהכרח מתקשרים ישרות האחד לשני, והם יכולים לתקשר באמצעות צמתים אחרים.

לדוגמה, נתונה הרשת הבאה המורכבת משמונה צמתים:



לדוגמא צומת מס' 2, מחוברת לשלושה צמתים: 1,3 ו-4.

אם צומת מס' 2 מעוניין לשלוח הודעה לצומת מס' 6 הוא יכול לעשות את זה דרך אחד מהמסלולים הבאים: 2← 3 ←5←6 או 2←4←5←6. בחירת המסלול בו ישתמש הצומת יוגדר בהמשך.

פקודות משתמש בסיסיות

כל צומת מקבל פקודות מהמשתמש שאותה הצומת קורא באמצעות הפונקציה fgets.

הפקודות אותם משתמש יכול לתת לצומת מתומצתות בטבלה הבאה:

תיאור	מבנה הפקודה	דוגמה	תיאור הדוגמה
קביעת מזהה לצומת	setid,<*id*>	setid,7	קביעת מזהה הצומת למספר 7
התחברות לצומת חדש (יכול להגיע	connect,<*ip*>:<*port*>	connect,127.0.0.1:1234	בקשת התחברות לצומת בכתובת 127.0.0.1 בפורט 1234
שליחת הודעה לצומת	send,<*id*>,<*len*>,<*message*>	send,3,11,hello world	שליחת ההודעה "hello world" לצומת מספר 3. 11 מציין שמספר התווים לשליחה הוא 11.
הדפס מסלול לצומת	route,<*id*>	route,3	בקשה להדפיס את המסלול מהצומת הנוכחי לצומת 3
הדפס רשימת צמתים מחוברים	Peers	peers	בקשה להדפיס מי הקבצים המחוברים לצומת הנוכחי

לכל הפקודות הצומת עונה ack אם הצליח או nack אם נכשל. אם יש מידע נוסף, הוא יודפס לאחר מכן.

פרוטוקול התקשורת בין הצמתים

התקשורת בין הצמתים מבוססת על הודעות במבנה ידוע ומוגדר מראש כפי שיוגדר להלן. מבנה ורצף ההודעות מרכיבים את פרוטוקול התקשורת בין הצמתים.

כל צומת מבצע ארבע פעולות בסיסיות מול צמתים אחרים:

1. **התחברות** הפעולה הבסיסית שהצומת עושה הוא להתחבר לפחות לצומת אחד ברשת. צומת יכול להיות מחובר ליותר מצומת אחד.
   שימו לב: כל צומת צריך להיות מסוגל גם לקבל התחברויות מצמתים אחרים.
2. **חיפוש מסלול לצומת**. בכדי לשלוח הודעה לצומת יש להכיר את המסלול אל הצומת. במידה וצומת היעד מחובר ישירות לצומת המקור, ניתן לשלוח ישירות את הודעה, אחרת יש לאתר את המסלול אל הצומת.
3. **ממסור (relay) של הודעה**. כאשר צומת מקור איננו מחובר ישירות לצומת היעד, הוא יכול לבקש מצמתים במסלול לממסר את ההודעה אל צומת היעד.
4. **שליחת הודעה**. שליחת הודעה היא כאשר היא מגיעה לצומת היעד.

הפרוטוקול הוא פרוטוקול בינארי מורכב מהודעת (messages) בגודל קבוע של 512 בתים. מבנה כל הודעה בפרוטוקול מתואר בטבלה הבאה:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)

משמעות השדות:

1. Msg ID – מזהה ייחודי להודעה
2. Source ID – מזהה הצומת השולח
3. Destination ID – מזהה הצומת המקבל
4. # Trailing Msg – מספר החלקים שנשארו להודעה הנוכחית
5. Function ID - מזהה הפונקציה
6. Payload

רשימת הפונקציות הבסיסיות

מספר הפונקציה	שם הפונקציה	פרמטרים	פונקציית תגובה
1	Ack	מספר ההודעה עליה נשלח ה-ack (4 בתים ראשונים ב-payload)	
2	Nack	מספר ההודעה עליה נשלח ה-nack (4 בתים ראשונים ב-payload)	
4	Connect	ללא	ack או nack
8	Discover	צומת היעד (4 בתים ראשונים ב-payload)	route
16	Route	1. מספר הודעת ה-discover המקורית (4 בתים ראשונים ב-payload) 2. אורך התשובה (4 בתים שניים ב-payload). במידה	

		ולא נמצא מסלול אורך התשובה יהיה 0. 3. הצמתים לפי סדר השליחה (4 בתים לכל צומת)	
32	Send	1. אורך ההודעה (4 בתים ראשונים ב-payload של ההודעה הראשונה בלבד) 2. ההודעה עצמה (החל מהבית החמישי)	ack או nack
64	Relay	1. מה הצומת הבא במסלול (4 בתים ראשונים ב-payload) 2. מספר ההודעות העוקבות אותם יש לממסר (4 בתים שניים ב-payload)	ack או nack

סדר הפעולות בשליחת הודעה

התחברות

פעולה זו מתבצעת כאשר מתקבלת פקודה מהמשתמש להתחברות לצומת חדש.

תנאי מקדים: על צומת להכיר את המזהה שלו, לפני שהוא מתחבר לצומת החדש.

רצף הפעולות:

1. פתיחת TCP socket לכתובת (IP, port) שהתקבלו מהמשתמש.
2. במידה והפתיחה נכשלה, יש לשלוח הודעת nack למשתמש
3. שליחת הודעה עם connect עם Source ID מאותחל למזהה הצומת ו-0 כמזהה הצומת השני
4. הצומת השני צריך להחזיר הודעת ack עם מזהה הצומת שלו ב-Source ID, מזהה הצומת הנוכחי ב-Destination ID, ומספר הודעת ה-connect ב-payload
5. במידה ואחד המזהים לא נכון, יש להחזיר הודעת nack למשתמש.
6. במידה וכל המזהים נכונים, יש לשמור את מזהה הצומת, כתובת ה-IP וה-port במערך. יש לוודא שאורך המערך הוא באורך מספר הצמתים המחוברים לצומת הנוכחי. יש להדפיס את מזהה הצומת למשתמש

חיפוש מסלול

פעולה זו מתבצעת כאשר מתקבלת פקודה מהמשתמש לשלוח הודעה לצומת אחר, אבל לא ידוע מסלול לצומת המבוקש.

תנאי מקדים: הצומת מחובר לפחות לצומת אחד נוסף.

אופן הפעולה מבוסס על ביצוע הצפה של הודעת discover ברשת וקבלת המסלול בחזרה.

1. אם צומת היעד הוא צומת מחובר חזור מיד הודעת ack.
2. הצומת יעביר הודעת discover לכל הצמתים השכנים שעוד לא קיבלו ההודעה. אם אין צומת כזה, יש להחזיר nack.
3. לכל צומת שכן, בצע את הפעולות החל משלב 1.
4. אם מכל השכנים התקבלה הודעת nack החזר nack.

5. לכל השכנים שהחזירו route, בחר את המסלול הקצר ביותר. מבין המסלולים הקצרים בחר את המסלול שהסדר הלקסיקוגרפי שלו הוא הנמוך יותר.
6. למסלול שנבחר הוסף את מזהה הצומת הנוכחי בראש הרשימה תוך התאמת כלל השדות ושלח הודעת route למקור הבקשה. יש להחזיר הודעת route על כל אחד מהחיבורים מהם הגיע בקשת discover.

את רשימת המסלולים יש לשמור במערך שאורכו כמספר המסלולים הידועים, שמכיל מצביע לרשימת הצמתים על המסלול.

ממסור

כאשר יש לשלוח הודעה לצומת יעד, אבל צומת היעד איננו מחובר ישירות לצומת המקור, יש לממסר את ההודעה לצומת היעד. פעולת הממסור היא פקודה לצומת מחוברת להעביר אוסף של הודעות לצומת שהיא מחוברת אליה.

את רצף הצמתים הנדרש ניתן לקבל מבקשת *חיפוש מסלול*. להודעה שצריכה להגיע מצומת n_1 לצומת n_i על מסלול באורך i צמתים, מצומת $n_1 \to n_2 \to \cdots \to n_{i-1} \to n_i$, יש לשלוח $i-2$ הודעות relay אחת בתוך השנייה ובסוף הודעת send.

כל צומת שמקבל הודעת relay ממסר את מספר ההודעות **העוקבות** שהוא התבקש לצומת שאליה הוא התבקש לממסר את המידע.

אם צומת מתבקש לממסר הודעות לצומת שאיננו מחובר אליו הוא צריך להוציא הודעת nack.

לדוגמה, לפי הציור לעיל, כדי לשלוח הודעה מצומת 1 לצומת 5, על המסלול $1 \to 2 \to 3 \to 5$ צומת 1 תשלח שרשור של שתי הודעות relay ובתוכם הודעת send (ראו להלן).

לאחר שהצומת יקבל ack או nack על ממסור מהצומת הבא הוא יוציא הודעת ack או nack לצומת ששלח אליו את ההודעה בהתאמה.

שליחה

צומת מקור ששולח הודעה לצומת יעד ישלח הודעת send (לאחר שמומסרה כראוי) בה תוכן ההודעה.

1. במידה וצומת מקבל הודעת send שאיננה מיועדת אליו הוא יוציא חזרה הודעת nack למי ששלח לו את ההודעה.
2. במידה והודעת ה-send מיועדת לצומת הנוכחי, הוא ידפיס את ההודעה למשתמש ויחזיר הודעת ack לצומת ששלח אליו את ההודעה.

חומרי עזר

לתרגיל מצורף קוד שמאפשר להאזין בו זמנית על מספר socket ועל ה-stdin (מקלדת). הדבר נדרש כדי שניתן יהיה לקבל קלט מהמשתמש (מקלדת) או מכל אחד מהצמתים שמחוברים לצומת הנוכחי. במידה ומתקבל קלט כאמור יש לפעול על פי ההגדרות.

בקוד המצורף יש שני פונקציות – אחת להוסיף file descriptor (מזהה של ה-socket) בשם add_fd_to_monitoring. הפונקציה מקבל כקלט את מספר ה-file descriptor ומוסיפה אותו לרשימת ה- file descriptors שאותם היא מנטרת. הקוד מנטר אוטומטית גם קלט שמגיע מהמשתמש

הפונקציה השנייה היא wait_for_input הממתינה עד שיהיה קלט באחד ה-file descriptors (רשימת ה-socket שהוזנה באמצעות ב-add_fd_to_monitoring או קלט מהמשתמש). כאשר יש קלט זמין כאמור, הפונקציה wait_for_input תחזיר את ה-file descriptor שממנו ניתן לקרוא. לצורך ההבהרה, file descriptor מספר 0 הוא קלט שהגיע מהמשתמש וניתן להשתמש בפונקציות סטנדרטיות לטובת קריאת הקלט מהמשתמש. שימו לב שלאחר קריאת המידע הזמין מומלץ להפעיל שוב את הפונקציה wait_for_input **לפני** ביצוע קריאה נוספת מאותו file descriptor. במידה ויש קלט זמין נוסף הפונקציה תחזור מיד.

הקוד המסופק כולל קוד לדוגמה לאופן השימוש בפונקציות.

הערות כלליות

חלק מבדיקת התרגיל תהיה אוטומטית, בחלקה או כולה. לפיכך, אין להוסיף תווים מיותרים, כולל רווחים שורות חדשות וכו'. כמו כן, אין להחסיר תווים. שימו לב שכל שורת קלט\פלט מהמשתמש\אל המשתמש מסתיימת בשורה חדשה ('\n') ללא רווחים לפניה.

דוגמאות

לצורך הפשטות אנו מניחים כי כתובת ה-IP של כל אחד מהצמתים הוא 10.0.0.x כאשר x הוא מספר הצומת. הפורט שמשתמשים בו הוא 1234. לדוגמה, צומת 5 נמצא בכתובת 10.0.0.5 בפורט 1234.

שימו לב: כתובות ב-IP והפורטים בבדיקה יתקבלו מהמשתמש ולא יהיו בהכרח כמו בדוגמה.

צבעים בשימוש: רקע ירוק – קלט מהשתמש, רקע צהוב – פלט מהצומת.

חיבור צומת 3 לצמתים 2, 5

From the user blue

setid,3

ack

connect,10.0.0.5:1234

הודעת connect מצומת 3 לצומת 5:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
1	3	0	0	4	

הודעת ack מצומת 5 לצומת 3:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
10	5	3	0	1	1

Ack

5

connect,10.0.0.2:1234

הודעת connect מצומת 3 לצומת 2:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
2	3	0	0	4	

הודעת ack מצומת 2 לצומת 3:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
10	2	3	0	1	2

Ack

2

שליחת הודעה מצומת 1 לצומת 2

לפי הדוגמה, צומת 1 צמוד לצומת 2. אנו מניחים שתהליך החיבור בין הצמתים בוצע קודם להרצה הנוכחית.

send,2,12,hello world!

הודעת send מצומת 1 לצומת 2:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)	
221	1	2	0	32	4B	12B
					12	Hello world!

הודעת ack מצומת 2 לצומת 1:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
12	2	1	0	1	221

ack

שליחת הודעה מצומת 2 לצומת 5

ההנחה היא כי כל הצמתים חוברו קודם, מוצג כאן רק הודעות שצומת 1 מעורב בהם.

send,5,12,hello world!

הודעת discover מצומת 2 לצומת 1:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
3	2	1	0	8	5

הודעת discover מצומת 2 לצומת 3:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
4	2	3	0	8	5

הודעת discover מצומת 2 לצומת 4:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
5	2	4	0	8	5

הודעת nack מצומת 1 לצומת 2:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)

3	1	2	0	2	3

הודעת route מצומת 4 לצומת 2:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)			
100	4	2	0	16	4B	4B	4B	4B
					5	2	4	5

הודעת route מצומת 3 לצומת 2:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)			
321	3	2	0	16	4B	4B	4B	4B
					4	2	3	5

הודעת relay ו- send (משורשרות) מצומת 2 לצומת 5 דרך צומת 3:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)	
431	2	3	1	64	4B	4B
					5	1

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)	
432	2	5	0	32	4B	12B
					12	Hello world!

הודעת ack מצומת 3 לצומת 2:

MSG ID (4 bytes)	Source ID (4 bytes)	Destination ID (4 bytes)	# Trailing Msg (4 bytes)	Function ID (4 bytes)	Payload (492 bytes)
322	3	2	0	1	431

```
ack
route,5
ack
2->3->5
peers
ack
1,3,4
```

שאלות נפוצות

1. איך ניתן לקבל הודעות ולשלוח הודעות במקביל? כלומר, במידה ואני שולח הודעה לצומת ובאותו זמן מקבל הודעה מצומת, איך אני אמור לתפקד?
תשובה: ישנן כמה אפשרויות ליצר סוקטים שיתפקדו במקביל. אנו מציעים לכם להשתמש בקוד המצורף. ישנן שתי פונקציות שעליכם להשתמש על מנת להצליח למקבל את השימוש בסוקטים:
   א. add_fd_to_monitoring
   ב. wait_for_input

הפקודה הראשונה מקבלת כקלט file descriptor או במקרה שלכם סוקט, ומוסיפה אותו לרשימת הFDs שניתן למקבל. הפקודה השניה מכניסה את המחשב למצב בו הוא יכול למקבל בין סוקטים, וכאשר מתקבל קלט מאחד מהם ניתן לבצעו.

במקרה שלנו, הקוד כרגע ממקבל בין סוקטים כאשר בצורה דיפולטיבית הוא מקבל מידע גם משורת הפקודה(command line). פקודות אלו ישמשו אתכם בבניית השלד של הקוד, כך שתוכלו לקבל הודעות/בקשות חיבור מצמתים שונים ולבצעם, ובמקביל לבצע פעולות בצומת שלכם. לדוגמא, אם צומת 3 מנסה להתחבר לצומת 4, ובמקביל צומת 1 מנסה להתחבר לצומת 3-סוקט א' של צומת 3 יאזין לפקודות מהעולם, וסוקט ב' יאזין לפקודות הפנימיות של צומת 3. כך, פקודת החיבור לצומת 4 תופעל בסוקט ב', ובסוקט א' יופעל תתקבל הבקשה מצומת 1.

2. איך ניתן להריץ את הדוגמה?
תשובה: ניתן להריץ את הפקודה:
```
gcc example.c select.c
```
בשורת הפקודה שלכם. בדוגמא כאן, המחשב מאזין לשורת הפקודה, ולשני סוקטים בפורטים 5000 ו5001. כל קלט שתשלחו לאחד מהסוקטים/דרך שורת הפקודה, יודפס על המסך. האזנה זו תסתיים לאחר עשרה קלטים.
3. האם צריך להבין את הקוד המצורף?
תשובה: אין חובה להבין את הקוד המצורף אלא לדעת להשתמש בפקודות לעיל. כמו כן, אם אתם רוצים למקבל את הפקודות מהcommand line לבין פקודות מצמתים אחרים ע"י קוד אחר, אתם מוזמנים.
4. איך ניתן להטמיע את הקוד שנתתם בתרגיל שלנו?
תשובה: קבצי ה select השונים נותנים לכם את האפשרות להשתמש בשתי הפקודות הנ"ל על מנת למקבל את השימוש בסוקטים והקלט מהמשתמש. כאשר תכתבו את הקוד שלכם, החליפו את example.c מדוגמת הרצת ה-gcc לעיל בשם קובץ הקוד שלכם, וכך תוכלו להשתמש בפקודות לעיל.
5. מהו השימוש ב#trailing msg?
תשובה: השימוש הוא במקרה ומשרשרים הודעות. כלומר – במידה ושולחים הודעה שהיא ארוכה מהמקום הזמין ב-playload ההמשך ההודעה תופיע במנה הבאה
6. איך מייצרים msg ID?
תשובה: יש לייצר מספר בצורה שאיננה חוזרת על עצמה בגבולות גודל המשתנה(4 בתים), ניתן להשתמש במזהה אקראי (random), סידרתי או אחר, כל זמן שאיננו חוזר על עצמו.
7. האם ההודעות משורשרות, או נמצאות אחת בתוך השניה?

תשובה: הכוונה היא שהן משורשרות, כלומר-הודעות עוקבות שנשלחות יחד.