

# תוכן עניינים:

## תיאור מערכת

- 2.....תיאור כללי
- 3.....איך מריצים את התוכנה
- 4.....פונקציות משתנים והסברים
- 5.....תוצאות ופלט

## המחקר בפועל

- 6.....צילומי מסך מה**Wireshark** והסברים
- 7.....מסקנות
- 8.....מקורות ידע

תיאור מערכת :

תיאור כללי :

במטלה נדרשנו לכתוב שלושה קבצי קוד : **ping.c**, **better\_ping.c**, **watchdog.c** .

### Ping.c :

בקובץ זה השתמשנו בקובץ **myPing.c** שניתן לנו בתור קובץ עזר ואותו שינינו עפ"י דרישות המטלה.

תחילה, נדרשנו לקבל דרך המרמינל את ה**Ip** שאיתו נרצה לעבוד בעזרת הפקודה :

```
shoval@shoval-VirtualBox:~/Desktop/github/Computer-Networking-4/Computer_Networking2$ sudo ./better_ping 8.8.8.8
```

ולאחר מכן אנו בודקים שה**Ip** שקיבלנו תקין .

בנוסף, אנו יוצרים **raw socket** שדרכו נשלח את הפינג בפרוטוקול **icmp**. בהמשך התוכנית אנו שולחים בלולאה אינסופית (במקרה שלנו לשרת של גוגל) **ping** ומקבלים **pong** מהשרת.

לאחר הקבלה של ה**pong** אנו מדפיסים את ה**Ip** של הפאקטה שנשלחה (כלומר מאיפה היא הגיעה) , **sequence number** של הפאקטה (כלומר המספר הסידורי שלה) , ואת הזמן בין הבקשה לתשובה שזהו בעצם ה**RTT** .

### better\_ping.c :

בקובץ זה השתמשנו בקובץ **ping.c** ואותו שינינו עפ"י דרישות המטלה .

דרך קובץ זה הרצנו את קובץ ה**watchdog** שעליו נרחיב בהמשך.

לאחר מכן, אנו פותחים קשר **Tcp** בין ה**better\_ping** שלנו שישמש בעצם כ"לקוח" לבין ה**watchdog** שלנו שישמש כ"שרת".

במשך הלולאה שבה אנו שולחים פינג, כל פעם מחדש אנו שולחים ל**watchdog** הודעה ומעדכנים אותו ששלחנו פינג כדי שהוא יידע למדוד את הזמן שלוקח לשרת שאליו שלחנו את הפינג , לשלוח לנו בחזרה את התשובה שלו.

### Watchdog.c :

תחילה, אנו פותחים קשר **Tcp** בינו לבין הלקוח.

לאחר מכן, אנו מפעילים את המיימר שלנו ואז מקבלים בלולאה את ההודעה מהלקוח שבה הוא אומר לנו שהוא שלח פינג. אם קיבלנו את ההודעה בהצלחה אנו מאפסים את המיימר שלנו ואם לא אנו מחשבים כמה שניות לקח לשרת שאליו שלחנו את הפינג לשלוח לנו בחזרה את התשובה ואם זה עלה על עשר שניות אנו "הורגים" את התהליך וסוגרים את הסוקטים שדרכם התחברנו.

## איך מריצים את התוכנה :

בכדי להריץ את התוכנה , נצטרך במרמינל להריץ את הפקודות הבאות:

חלק ראשון-בחלק זה אנו מצפים שגוגל יפסיק לשלוח לנו תשובה בזמן מסוים ולא יקרה שום דבר.

```
shoval@shoval-VirtualBox:~/Desktop/github/Computer-Networking-4/Computer_Networking2/Computer_Networking2
$ make PartA
gcc -Wall ping.c -o PartA
shoval@shoval-VirtualBox:~/Desktop/github/Computer-Networking-4/Computer_Networking2/Computer_Networking2
$ sudo ./PartA 8.8.8.8
```

חלק שני-בחלק זה אנו מצפים שכאשר גוגל יפסיק לשלוח לנו תשובה לאחר עשר שניות התהליך יסתיים ונראה "killed".

```
shoval@shoval-VirtualBox:~/Desktop/github/Computer-Networking-4/Computer_Networking2/Computer_Networking2
$ make PartB
gcc -Wall better_ping.c -o PartB
gcc -Wall watchdog.c -o watchdog
shoval@shoval-VirtualBox:~/Desktop/github/Computer-Networking-4/Computer_Networking2/Computer_Networking2
$ sudo ./PartB 8.8.8.8
```

## פונקציות משתנים והסברים :

בחלק זה נסביר על הפונקציות החדשות שאותם למדנו במטלה זו ולא על אלו שאותם אנו כבר מכירים ממטלה קודמת .

**sendto()**-פונקציה שעולחת הודעה ואנו משתמשים בפונקציה זו מכיוון שאיתה משתמשים כאשר עובדים עם **udp**, מכיוון שאין לנו כאן חיבור עם השרת שאליו אנו שולחים את הפינג.

**recvfrom()** – זוהי פונקציה שמקבלת את ההודעה שעלחנו, ושוב אנו משתמשים בה כאשר עובדים עם **udp** כי אין לנו חיבור עם השרת שממנו אנו מקבלים את הפונג.

**fork()** – זוהי פונקציה שמפצלת תהליך אחד לשני תהליכים: אחד ייקרא "האב" שיקבל **id** אפס והשני ייקרא "הילד" וגם הוא יקבל **id** (שוונה מאפס), שני התהליכים ירוצו במקביל.

**execvp()** – זוהי פונקציה שדרכה אנו מריצים את התהליך של "הילד".

**kill()** – זוהי פונקציה שדרכה ניתן להרוג את תהליך האב.

**seconds** - משתנה שמודד את השניות שלוקח לשרת של גוגל להחזיר לנו תשובה מהרגע שעלחנו אליו פינג.

## תוצאות ופלט :

חלק ראשון- לאחר שנקמפל ונריץ את החלק הראשון נקבל :

```
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 41.293030 milliseconds (-957707 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 0 --- Time between request and replay: 41.293030 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 87.192001 milliseconds (87192 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 1 --- Time between request and replay: 87.192001 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 11.000000 milliseconds (11000 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 2 --- Time between request and replay: 11.000000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 10.301000 milliseconds (10301 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 3 --- Time between request and replay: 10.301000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 10.467000 milliseconds (10467 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 4 --- Time between request and replay: 10.467000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 19.077999 milliseconds (19078 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 5 --- Time between request and replay: 19.077999 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 14.291000 milliseconds (14291 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 6 --- Time between request and replay: 14.291000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
```

ניתן לראות כאן את ההדפסות שנדרשנו לעשות שכוללות מאיפה קיבלנו את החבילה , המספר הסידורי שלה, והRTT .

חלק שני- לאחר קימפול והרצה נצפה לקבל שלאחר שגוגל יפסיקו לשלוח לנו תשובה התהליך "ייהרג".

```
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 4.383000 milliseconds (4383 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 669 --- Time between request and replay: 4.383000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 6.052000 milliseconds (6052 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 670 --- Time between request and replay: 6.052000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 4.589000 milliseconds (4589 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 671 --- Time between request and replay: 4.589000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 4.626000 milliseconds (4626 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 672 --- Time between request and replay: 4.626000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 23.990999 milliseconds (23991 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 673 --- Time between request and replay: 23.990999 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

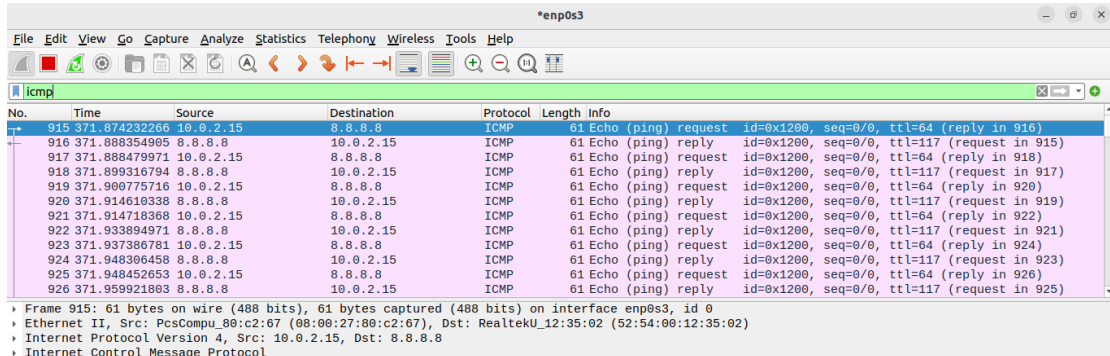
RTT: 6.731000 milliseconds (6731 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 674 --- Time between request and replay: 6.731000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
47 bytes from 8.8.8.8
Successfully received one packet with 47 bytes : data length : 19 , icmp header : 8 , ip header : 20

RTT: 10.289000 milliseconds (10289 microseconds)
Packet IP: 8.8.8.8 --- Packet Sequence Number: 675 --- Time between request and replay: 10.289000 milliseconds
Successfully sent one packet : ICMP HEADER : 27 bytes, data length : 19 , icmp header : 8
Killed
```

## המחקר בפועל :

## צילומי מסך מהWireshark והסברים :

### חלק 1:



The screenshot shows a Wireshark capture on interface enp0s3. The filter is set to 'icmp'. The packet list shows ICMP Echo (ping) requests and replies. The packet details pane shows the structure of an ICMP Echo request and reply, including fields like ID, sequence number, and TTL.

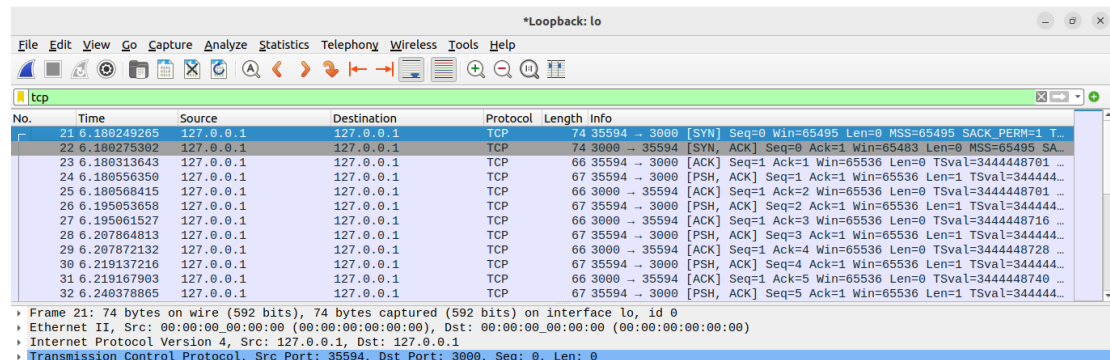
| No. | Time          | Source    | Destination | Protocol | Length | Info   |
|-----|---------------|-----------|-------------|----------|--------|--|
| 915 | 371.874232266 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 916)  |
| 916 | 371.888354905 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 915) |
| 917 | 371.888479971 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 918)  |
| 918 | 371.899316794 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 917) |
| 919 | 371.906775716 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 920)  |
| 920 | 371.914610338 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 919) |
| 921 | 371.914718368 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 922)  |
| 922 | 371.933894971 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 921) |
| 923 | 371.937386781 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 924)  |
| 924 | 371.948306458 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 923) |
| 925 | 371.948452653 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 926)  |
| 926 | 371.959921803 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 925) |

Frame 915: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface enp0s3, id 0  
Ethernet II, Src: PcsCompu\_08:c2:67 (08:00:27:80:c2:67), Dst: RealtekU\_12:35:02 (52:54:00:12:35:02)  
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8  
Internet Control Message Protocol

ניתן לראות בחלק זה שהמחשב שלנו שולח פייג לוגול בפרוטוקול **icmp** ומקבל תשובה וכך זה ממשיך עד שגוגל מפסיק לשלוח תשובה בחזרה.

### חלק 2:

## ממשק ה loopback :



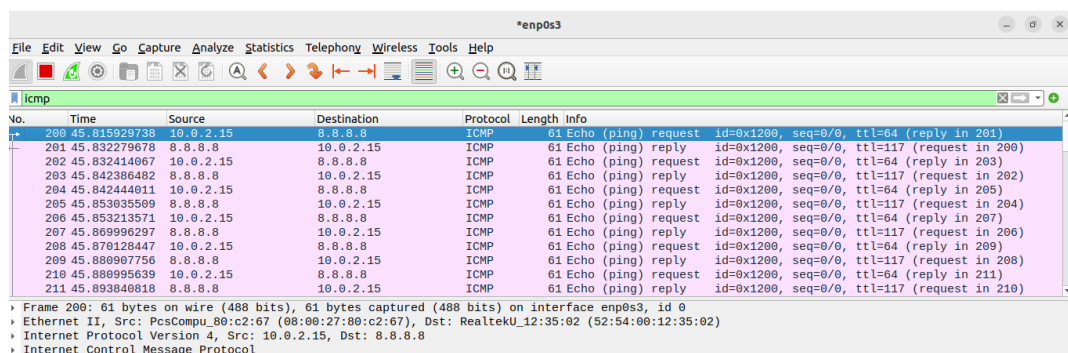
The screenshot shows a Wireshark capture on interface lo. The filter is set to 'tcp'. The packet list shows TCP SYN, ACK, and PSH packets. The packet details pane shows the structure of a TCP segment, including fields like sequence number, window size, and flags.

| No. | Time        | Source    | Destination | Protocol | Length | Info  |
|-----|-------------|-----------|-------------|----------|--------|---|
| 21  | 6.180249265 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 35594 → 3000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 T... |
| 22  | 6.180275302 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 3000 → 35594 [ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495 SA...      |
| 23  | 6.180313643 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 35594 → 3000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3444448701 ... |
| 24  | 6.180556350 | 127.0.0.1 | 127.0.0.1   | TCP      | 67     | 35594 → 3000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1 TSval=344444... |
| 25  | 6.180568415 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 3000 → 35594 [ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=3444448701 ... |
| 26  | 6.195053658 | 127.0.0.1 | 127.0.0.1   | TCP      | 67     | 35594 → 3000 [PSH, ACK] Seq=2 Ack=1 Win=65536 Len=1 TSval=344444... |
| 27  | 6.195061527 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 3000 → 35594 [ACK] Seq=1 Ack=3 Win=65536 Len=0 TSval=3444448716 ... |
| 28  | 6.207864813 | 127.0.0.1 | 127.0.0.1   | TCP      | 67     | 35594 → 3000 [PSH, ACK] Seq=3 Ack=1 Win=65536 Len=1 TSval=344444... |
| 29  | 6.207872132 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 3000 → 35594 [ACK] Seq=1 Ack=4 Win=65536 Len=0 TSval=3444448728 ... |
| 30  | 6.219137216 | 127.0.0.1 | 127.0.0.1   | TCP      | 67     | 35594 → 3000 [PSH, ACK] Seq=4 Ack=1 Win=65536 Len=1 TSval=344444... |
| 31  | 6.219167903 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 3000 → 35594 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=3444448740 ... |
| 32  | 6.240378805 | 127.0.0.1 | 127.0.0.1   | TCP      | 67     | 35594 → 3000 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=1 TSval=344444... |

Frame 21: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface lo, id 0  
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)  
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
Transmission Control Protocol, Src Port: 35594, Dst Port: 3000, Seq: 0, Len: 0

ניתן לראות כאן שנפתח קשר **tcp** בין ה **better\_ping** ל **watchdog** וכך הם מתקשרים ביניהם כאשר הפייג שולח הודעה כאשר הוא שלח פייג לשרת של גוגל וה **watchdog** מקבל את ההודעה.

## ממשק ה Wi-Fi :



The screenshot shows a Wireshark capture on interface enp0s3. The filter is set to 'icmp'. The packet list shows ICMP Echo (ping) requests and replies. The packet details pane shows the structure of an ICMP Echo request and reply, including fields like ID, sequence number, and TTL.

| No. | Time         | Source    | Destination | Protocol | Length | Info   |
|-----|--------------|-----------|-------------|----------|--------|--|
| 200 | 45.815929738 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 201)  |
| 201 | 45.832279678 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 200) |
| 202 | 45.832414067 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 203)  |
| 203 | 45.842386482 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 202) |
| 204 | 45.842444011 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 205)  |
| 205 | 45.853035509 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 204) |
| 206 | 45.853213571 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 207)  |
| 207 | 45.869996297 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 206) |
| 208 | 45.870128447 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 209)  |
| 209 | 45.880907756 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 208) |
| 210 | 45.880955639 | 10.0.2.15 | 8.8.8.8     | ICMP     | 61     | Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 211)  |
| 211 | 45.893840818 | 8.8.8.8   | 10.0.2.15   | ICMP     | 61     | Echo (ping) reply id=0x1200, seq=0/0, ttl=117 (request in 210) |

Frame 200: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface enp0s3, id 0  
Ethernet II, Src: PcsCompu\_08:c2:67 (08:00:27:80:c2:67), Dst: RealtekU\_12:35:02 (52:54:00:12:35:02)  
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8  
Internet Control Message Protocol

ניתן לראות בחלק זה שליחה ותשובה של פייג לשרת של גוגל בפרוטוקול **icmp** .

## מסקנות :

במטלה זו נדרשנו לכתוב **ping** בעצמנו . **ping** הוא בעצם כלי נפוץ למציאת הסטטוס של מכשיר ברשת.

בכדי לעשות זאת השתמשנו ב**raw socket** שמאפשר תקשורת ישירה בין תוכנית למקור חיצוני ללא התערבות של מערכת ההפעלה הראשית על המחשב.

בנוסף , השתמשנו בפרוטוקול **icmp** שהוא פרוטוקול שממשמש ל**raw socket** מכיוון שאין לנו חיבור לשרת שאליו אנו שולחים את הפינג.

ההבדל העיקרי בין פרוטוקול זה לפרוטוקולים אחרים שלמדנו כמו **tcp,udp** הוא שבפרוטוקול זה אין צורך להתחבר לצד השני בכדי לשלוח מידע .

בנוסף, במטלה זו למדנו להשתמש ב**fork** שהיא פונקציה שמחלקת את התהליך הראשי שלנו לשני תהליכים שירוצו במקביל וככה בעצם הרצנו את הפינג שלנו ואת **watchdog** שרץ ברקע ומשמש לנו כטיימר כדי לדעת מתי עברו עשר שניות שבהם לא קיבלנו שום תשובה ולעצור את התהליך .

**מקורות ידע :**

**raw-socket** מצגת תרגול

**<https://www.howtouselinux.com/post/ping-icmp>**

**<https://www.easytechjunkie.com/what-is-a-raw-socket.htm>**