Universitatea Tehnică „Gheorghe Asachi” Iași

Facultatea de Automatică și Calculatoare

TRANSFER FIȘIERE – FEREASTRĂ GLISANTĂ

DOCUMENTAȚIE PROIECT REȚELE DE CALCULATOARE

Disciplină: Rețele de calculatoare – proiect

Student: Chihalău Adrian-Ioan

Profesor coordonator:

Ș.l.dr.ing. Nicolae-Alexandru Botezatu

1. Protocolul de transfer fișiere fereastra glisantă(Sliding window protocol)

1.1. Considerații generale

Acest protocol constă în ideea de a transmite de la un sender (emițător) un pachet (un șir de informații), împreună cu alte date care asigură receiver-ului (receptorului) primirea corectă al pachetului (de exemplu, un checksum, un bloc de date pentru detecția erorilor în timpul transmiterii datelor). Cand receiver-ul verifică datele, trimite un semnal de confirmare(ACK) înapoi la sender pentru a putea transmite următorul pachet. Folosind protocolul ARQ(automatic repeat request), sender-ul se oprește pentru fiecare pachet si așteaptă de la receiver semnalul ACK. Acest lucru asigură trimiterea pachetelor în ordinea corectă, deoarece doar unul poate fi trimis odată.

Totuși timpul în care este primit semnalul ACK poate fi mai mare decât timpul necesar trimiterii unui pachet, iar în consecință, cantitatea de pachete procesate este mult mai mică decât ar trebui posibilă. Astfel protocolul sliding window permite selecția a unui număr de pachete, o „fereastră", pentru a fi trimisă fără a mai aștepta un semnal ACK. Fiecare pachet primește un număr de secvență, iar semnalele ACK vor trimite înapoi numărul. Protocolul ține evidența pachetelor care au fost confirmate, iar când ele sunt primite, sunt trimise alte pachete. În acest mod, fereastra „glisează" printr-un șir de pachete care realizează transferul, și asigură ca evitarea supraaglomerării rețelei.

1.2. Metoda selective repeat fără Nack

Este cel mai general caz de protocol fereastră glisantă, care necesită un receiver mult mai capabil. Atât sender-ul, cât și receiver-ul vor avea aceeași dimensiune a ferestrei de pachete (în mod obișnuit, este aleasă dimensiunea 2, însă în aplicația proietului va putea fi mărită dimensiunea ferestrei). Sender-ul va trimite câte un pachet spre receiver, pornindu-se în același timp un timer de expirare (timeout). Dacă nu este primit un semnal de ACK de la receiver înaintea expirării timpului, se trimite din nou pachetul și se resetează timpul timeout.

Când un pachet ajunge în receiver, se va trimite un semnal ACK către sender, iar fereastra receiver-ului se va muta cu o poziție la dreapta; astfel se va asigura primirea pachetelor de receiver. Cand semnalul de ACK ajunge în sender înainte de expirarea timpului timeout, sender-ul va considera că pachetul a fost preluat cu succes de receiver si va muta fereastra lui cu o poziție la dreapta.

2. Descriere proiect

Proiectul constă în realizarea unei aplicații care va simula transferul de fișiere între 2 calculatoare conectate la aceeași rețea, prin intermediul protocolului sliding window, folosind metoda selective repeat fără Nack. Va avea 2 view-uri: una pentru transmisie, și una pentru recepție, iar comunicația va fi implementată prin diagrame UDP. View-ul pentru recepție va permite „pierderea" voită a pachetelor pentru a putea demonstra funcționalitatea mecanismului de retransmitere a pachetelor. De asemenea, se vor putea configura parametrii de funcționare (dimensiune fereastră. temporizări etc.).

Aplicația va fi scrisă în limbajul de programare Python, care are la dispoziție biblioteca sockets pentru comunicarea între calculatoare.

2.1. Caracteristici pachet

În general, **pachetul conține 3 informații**:

- **tipul pachetului** - dacă este un pachet conține doar un text(info) sau este de confirmare(ack);
- **informația stocată** – ce informație s-a transmis prin intermediul pachetului ; în cazul tipului info, va fi un șir de 30 caractere(30 octeți), care nu conține caracterul end-of-line, în timp ce pentru tipul ack, va fi 1 pentru valoarea adevărat(pachetul a fost primit și prelucrat de receiver);
- **poziția secvenței din propoziție** – pentru că o propoziție se va împărții în fragmente de câte 30 caractere pentru a putea transmite pas cu pas pachetele din sender către receiver, se mai transmite și poziția unui astfel de fragment, pentru a se asigura că, la final, receiver-ul va primi toate acestea în ordine corectă. De asemenea, se va știi exact și ce pachete au fost primite sau nu, în consecință ce pachete cu numarul de secvență aferent vor primi ack de la receiver.

2.2. Comunicația între mașini

Conexiunea se va realiza prin datagrame UDP. Acestea reprezintă un protocol de comunicație pentru calculatoare, făcând posibilă livrarea mesajelor într-o rețea. UDP nu dispune de mecanisme de verificare a siguranței datelor, nici pentru ordinea de sosire corectă a mesajelor sau de detectare a duplicatelor, dar dispune de sume de control pentru verificarea integrității datelor sau informații privind numărul portului pentru adresarea diferitelor funcții la sursa/destinație.

Caracteristici UDP:

- **orientat către tranzacții** (util în aplicații simple de tip întrebare-răspuns);
- **foarte util în aplicații de configurări** (ex. DHCP);
- **lipsa întârzierilor de retransmisie**, pentru aplicații în timp real (VoIP, jocuri online);
- **lucrează excelent în medii de comunicații unidirecționale** precum furnizarea de informații broadcast, sau în partajarea de informații către alte noduri.

Antetul UDP este alcătuit din 4 câmpuri fiecare având lungimea de 2 octeți:

Biti	0 - 15	16 - 32
0	Portul sursa	Portul destinație
32	Lungime	Suma de control
64	Date	

- **Portul sursa** - în adresarea bazata pe IPv4 acest câmp este opțional. Daca nu este utilizat acest câmp, are valoarea zero; când reprezintă informație semnificativa, el va indica portul inițiator al procesului de transmisie a datagramelor.
- **Portul destinație** - spre deosebire de portul sursa, câmpul este obligatoriu și indica portul de recepție
- **Lungime** - acest câmp indica lungimea în octeți a datagramei: antet plus secțiune de date (valoarea minima a lungimii este 8).
- **Suma de control** - asigura imunitatea la erori; se calculează ca fiind complementul fata de 1 (pe 16 biți) a pseudo-antetului cu informații extrase din antetul IP, antetului UDP și a câmpului de date, eventual se completează cu zerouri pentru a atinge lungimea stabilita.

Aplicația aferentă va avea acces la stiva de comunicație a rețelei folosind biblioteca sockets din Python.

2.3. Vizualizare transmisie/recepție

După cerința proiectului, vizualizarea transmisiei și a recepției se va realiza prin 2 view-uri separate. Acestea vor prelua de asemenea toate informațiile despre transmisia/recepția dintre calculatoare (portul sursa, portul destinație, număr pachet, informația transmisă, data, ora) și vor fi scrise în 2 casete ale interfetei, cate una pentru transmisie și recepție.

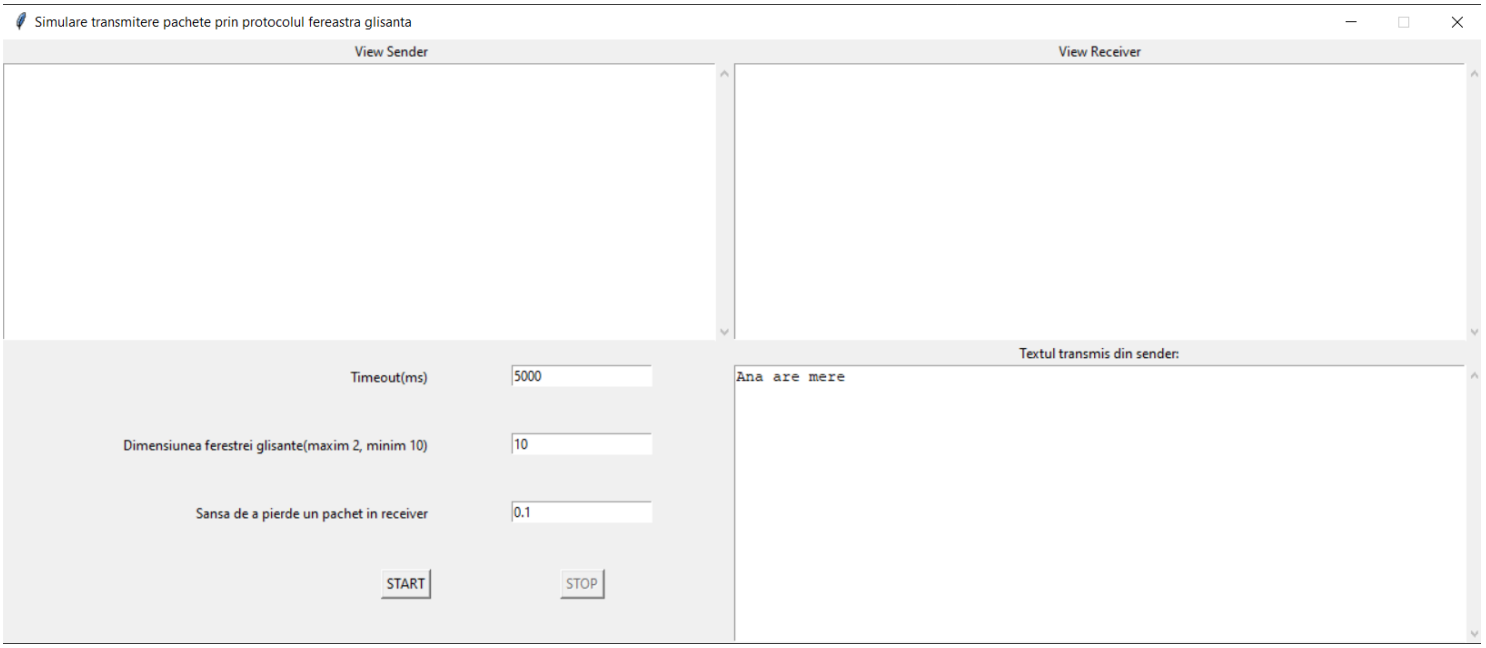
În interfață, se vor putea **modifica** următorii **parametrii de funcționare**:

- **timeout** (timpul de expirare pentru așteptarea unui răspuns de ACK de la receiver către sender – ms, implicit 5000 ms);
- **window size** (dimensiunea ferestrelor de transmisie și recepție, minim 2, maxim 10, implicit 2);
- **packet failure chance** (șansa de a „pierde” un pachet, un număr cuprins în intervalul [0,1], implicit 0,1).

Ultimul parametru de funcționare va permite „pierderea” voită a unor pachete în receiver pentru demonstrarea funcționării mecanismului.

2.4. Interfața grafică

Interfața grafică pentru care va fi permisă demonstrarea tuturor modurilor de funcționare este realizată folosind pachetul pentru construirea interfețelor în Python, **Tkinter**.



Interfața grafică a programului

Cele 2 casete-text de mai sus reprezintă cele **2 view-uri ale procesului de transmisie a pachetelor**: **pentru Sender(stânga) și pentru Receiver(dreapta)**. Acestea nu pot fi editate, doar citite(au fost setate pe read-only).

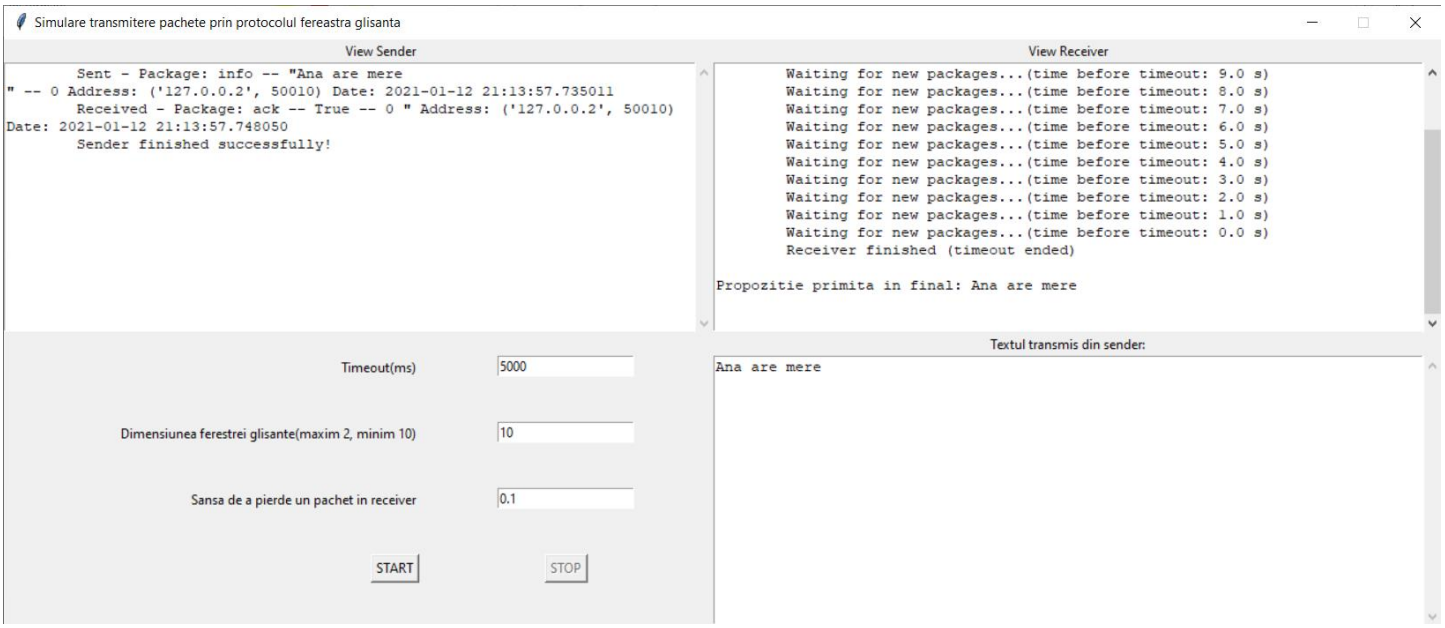
Caseta-text din dreapta jos este în schimb editabilă, pentru că servește drept **intrare a Sender-ului**.

Stânga jos se pot edita în **casete de intrare datele principale ale simulării transmisiei de date**: timpul pana când sender-ul mai trimite un pachet inca neconfirmat, dimensiunea ferestrelor glisante din sender și receiver și sansa „pierderii” unui pachet în receiver.

La apăsarea **butonului Start**, se va prelua textul scris în caseta dreapta jos, iar sender-ul va procesa acest test, transmințându-l mai departe în receiver prin intermediul protocolului de fereastră glisantă.

Acest proces poate fi oprit doar în timpul rulării acesteia prin intermediul **butonului Stop**, care nu va mai perminte sender-ului sa trimita alte pachete, iar receiver-ul sa mai astepte alte pachete noi venite.

Modul de așezare al elementelor de interfață (sau widget-uri) - butoane, casete-text, label-uri – s-a realizat prin **grid** (poziționarea elementelor ca într-o tabelă), fiind un mod mai ușor de a așeza aceste elemente în interfață.



Interfața după pornirea și terminarea simulării

3. Implementare

3.1. Definirea pachetelor și bucăților de fereastră

Acestea sunt piesele de bază în realizarea simulării, fiind modul de transmisie și păstrare a informațiilor între sender și receiver. Ele sunt **definite în modulul package** din cadrul proiectului, descris în scriptul package.py.

```python
import pickle


class package():
    def __init__(self, info_type, info, seq_num
        self.type = info_type
        self.info = info
        self.seq_num = seq_num

    def dump_pack(self):
        rez = pickle.dumps(self)
        return rez

    def load_pack(self, info):
        rez = pickle.loads(info)
        self.type = rez.type
        self.info = rez.info
        self.seq_num = rez.seq_num


class frame():
    def __init__(self, info, is_ack, seq_num):
        self.info = info
        self.is_ack = is_ack
        self.seq_num = seq_num
```

Clasa Package reprezintă pachetul ce va fi transmis și primit, cu diferențele precizate anterior, de sender și receiver. **Conține trei membrii**, specifici caracteristicilor pachetului:

- type – tipul pachetului;
- info – informația păstrată;
- seq_num – poziția pachetului în secvența de pachete

Constructorul clasei se ocupă cu alocarea celor 3 membrii, cu informații exacte; de aceea acesta are 3 argumente, pe lângă cel implicit din Python(self): tipul, informația păstrată și numărul de secvență.

Pentru că prin socket-uri se pot transmite și primi informații doar în format binar, clasa mai are la dispoziție 2 metode de împachetare/despachetare, folosindu-se de pachetul pickle din Python:

- dump_pack(self) – transformă obiectul de tip Package în șir de biți prin funcția din pachetul pickle dumps, pentru putea fi transmis prin socket-uri;
- load_pack(self, info) – prin argumentul info, care păstrează reprezentarea binară a unui pachet venită dintr-un socket, este tradus prin funcția din pachetul pickle loads, a cărui informații specifice unui pachet sunt transmise fiecărui membru aferent din obiectul prezent.

Clasa Frame implementează o parte din fereastră sau un frame. Acesta are tot 3 membri:

- info – informația păstrată, mai exact un text de maxim 30 caractere, fiind o parte din tot textul trimis din sender
- is_ack – un boolean ce marchează dacă partea a fost primită prin sau nu prin pachete de receiver
- seq_num – numărul care sugerează poziția curentă a frame-ului din toate fragmentele de propoziție; acest număr va ajuta mult la implementarea glisării ferestrei prin frame-uri atunci când primele

dintre acestea au fost confirmate și pot fi înlăturate, pentru a face loc altor noi, cu textul extras desigur din fraza rămasă de transmis.

La fel ca la clasa Package, există un constructor cu parametrii specifici atribuirii celor 3 membrii.

3.2. Implementarea interfeței

Interfața este creată prin scriptul interfața.py, folosind modulul tkinter din Python.

```
root = Tk()
root.title("Simulare transmitere pachete prin protocolul fereastra glisanta")
root.resizable(0, 0)

labelSenderView = Label(root, text="View Sender")
entrySenderView = Text(root, state=DISABLED, height=15, wrap=WORD)
scrollbarSenderView = Scrollbar(root, command=entrySenderView.yview)
entrySenderView.config(yscrollcommand=scrollbarSenderView.set)

labelReceiverView = Label(root, text="View Receiver")
entryReceiverView = Text(root, state=DISABLED, height=15, wrap=WORD)
scrollbarReceiverView = Scrollbar(root, command=entryReceiverView.yview)
entryReceiverView.config(yscrollcommand=scrollbarReceiverView.set)

labelSenderView.grid(row=0, column=0, sticky='E')
entrySenderView.grid(row=1, column=0, columnspan=2, sticky='N')
scrollbarSenderView.grid(row=1, column=1, ipady=97, sticky='NE')

labelReceiverView.grid(row=0, column=2, sticky='N')
entryReceiverView.grid(row=1, column=2, sticky='N')
scrollbarReceiverView.grid(row=1, column=3, ipady=97, sticky='N')

labelTxt = Label(root, text="Textul transmis din sender:")
entryTxt = Text(root, wrap=WORD, height=15)
scrollbarTxt = Scrollbar(root, command=entryTxt.yview)
entryTxt.config(yscrollcommand=scrollbarTxt.set)
```

Crearea interfeței începe cu definirea unui ferestre principale, numită în program root. Se foloseste constructorul Tk() pentru a defini variabila ca fereastră în Tkinter. Este setat titlul și modul de a nu fi redimensionat(fereastra are dimensiune fixă).

Urmează apoi crearea celorlalte elemente ale ferestrei: label-urile, casetele text, scrollbar-uri. Butoanele sunt create la final, odată ce au fost definite funcțiile pe care le declanșează.

Fiecare casetă text are înălțimea de 15 caractere, iar modul de vizualizare al cuvintelor este WORD (un cuvânt care nu încape la sfârșit de rând, este pus pe următoarea linie din casetă). Toate acestea au atașate și scrollbar-uri, pentru a putea vizualiza și texte mai lungi. Casetele text pentru view-urile de sender

si receiver(entrySenderView și entryReceiverView) au modul state pe DISABLED, adică acestea nu pot fi editate, ci doar citite. Numai caseta text pentru scrierea frazei de trimis în sender(entryText) poate fi editat.

Odată ce elementele de interfață de până acum au fost create, acestea sunt amplasate corespunzător pe fereastra root prin metoda grid: amplasarea widget-urilor ca intr-un tabel, precizând linia și coloana unde vor fi amplasate. Proprietatea sticky ajută la amplasarea mai estetică a widget-urilor în cazul în care coloana sau linia pe care sunt amplasate au o dimensiune mai mare decât elementul de interfață.

```
47      labelTimeout = Label(root, text="Timeout(ms)")
48      entryTimeout = Entry(root)
49
50      labelWinSize = Label(root, text="Dimensiunea ferestrei glisante(maxim 2, minim 10)")
51      entryWinSize = Entry(root)
52
53      labelFailure = Label(root, text="Sansa de a pierde un pachet in receiver")
54      entryFailure = Entry(root)
55
56      labelTimeout.grid(row=3, column=0, sticky='EN')
57      entryTimeout.grid(row=3, column=1, padx=20, sticky='N')
58      entryTimeout.insert('end', "5000")
59
60      labelWinSize.grid(row=4, column=0, sticky='EN')
61      entryWinSize.grid(row=4, column=1, sticky='N')
62      entryWinSize.insert('end', "10")
63
64      labelFailure.grid(row=5, column=0, sticky='EN')
65      entryFailure.grid(row=5, column=1, sticky='N')
66      entryFailure.insert('end', "0.1")
67
68      labelTxt.grid(row=2, column=2)
69      entryTxt.grid(row=3, column=2, rowspan=4, sticky='EN')
70      entryTxt.insert('end', "Ana are mere")
71      scrollbarTxt.grid(row=3, column=3, rowspan=4, ipady=97, sticky='WN')
72      scrollbarTxt.config(command=entryTxt.yview)
73
```

```
75  def insertViewSender(sentence):
76      entrySenderView.configure(state='normal')
77      entrySenderView.insert('end', "\t" + sentence)
78      entrySenderView.configure(state='disabled')
79      entrySenderView.see("end")
80
81
82  def insertViewReceiver(sentence):
83      entryReceiverView.configure(state='normal')
84      entryReceiverView.insert('end', "\t" + sentence)
85      entryReceiverView.configure(state='disabled')
86      entryReceiverView.see("end")
87
```

Următoarele elemente definite în interfață sunt label-urile și casetele-text pentru introducerea celorlalte 3 informații din simulare: timpul de așteptare a unui pachet până când primește pachetul, dimensiunea ferestrelor sender-ului și receiver-ului și șansa de a pierde un pachet.

Cele doua funcții din stânga vor ajuta la inserarea în view-urile sender și receiver a tuturor informațiilor despre transmisia/rececpția pachetelor din acestea. De precizat că pentru a putea edita cele 2 casete, temporar cele 2 casete sunt puse pe modul editabil pentru a pune fraza dorită.

```
517     buttonStart = Button(root, text="START", command=startSimulation)
518     buttonStart.grid(row=6, column=0, sticky='EN')
519
520     buttonStop = Button(root, text="STOP", command=stopSimulation, state=DISABLED)
521     buttonStop.grid(row=6, column=1, sticky='N')
522
523     root.mainloop()
524
```

Butoanele sunt create și poziționate pe fereastră de abia la final, pentru că înainte sunt definite funcțiile pentru pornirea/oprirea simulării de transmitere a frazei din sender în receiver. Inițial, butonul de stop este dezactivat pentru că simularea încă nu este pornită inițial. De asemenea, pentru a fi afișată fereastra la pornirea programului, variabila root specifică acesteia este setată ca bucla principală(astfel încât să ruleze încontinuu până la închiderea programului).

```
454 def startSimulation():
455     global stop_signal
456
457     error_msg = ""
458     error_num = 0
459     timeEnd = None
460     winSize = None
461     failChance = None
462     try:
463         timeEnd = int(entryTimeout.get())
464     except ValueError:
465         error_num += 1
466         error_msg += f"{error_num}: Timpul de timeEnd trebuie sa fie un numar intreg pozitiv!\n"
467
468     try:
469         winSize = int(entryWinSize.get())
470
471         if winSize < 2 or winSize > 10:
472             error_num += 1
473             error_msg += f"{error_num}: Dimensiunea ferestrei trebuie sa fie cuprinsa intre 2 si 10!\n"
474
475     except ValueError:
476         error_num += 1
477         error_msg += f"{error_num}: Dimensiunea ferestrei trebuie sa fie un numar intreg pozitiv, intre 2 si 10!\n"
478
479     try:
480         failChance = float(entryFailure.get())
481
482         if failChance < 0 or failChance >= 1:
483             error_num += 1
484             error_msg += f"{error_num}: Sansa de pierdere a unui pachet trebuie sa fie in intervalul [0,1)!\n"
485     except ValueError:
486         error_num += 1
```

Butonul de start, odată apăsat, declanșează funcția startSimulation(), ce realizează pornirea simulării. Mai întâi se verifică dacă informațiile introduse sunt corecte: timpul de timeout sa fie un număr întreg, dimensiunea ferestrei un număr cuprins între 2 și 10, iar șansa de pierdere a unui pachet sa fie un număr real, cuprins în intervalul [0, 1). Dacă nu apar erori, se poate porni simularea, altfel se afișează un mesaj de eroare ce cuprinde toate erorile apărute din textele scrise în casete.

```
        if failChance < 0 or failChance >= 1:
            error_num += 1
            error_msg += f"{error_num}: Sansa de pierdere a unui pachet trebuie sa fie in intervalul [0,1)!\n"
    except ValueError:
        error_num += 1
        error_msg += f"{error_num}: Sansa de pierdere a unui pachet trebuie sa fie un numar real pozitiv, intre [0," \
                     f"1)!\n"

    if error_msg != "":
        messagebox.showerror("Eroare la introducerea valorilor pentru simulare!", error_msg)
    else:
        entrySenderView.config(state=NORMAL)
        entrySenderView.delete('1.0', 'end')
        entrySenderView.config(state=DISABLED)

        entryReceiverView.config(state=NORMAL)
        entryReceiverView.delete('1.0', 'end')
        entryReceiverView.config(state=DISABLED)

        buttonStart.config(state=DISABLED)
        buttonStop.config(state=NORMAL)
        stop_signal = False
        textInEntry = entryTxt.get('1.0', END)

        sender_thread = threading.Thread(target=sender, args=(textInEntry, timeEnd, winSize,))
        receiver_thread = threading.Thread(target=receiver, args=(winSize, timeEnd, failChance,))
        receiver_thread.start()
        sender_thread.start()


def stopSimulation():
    global stop_signal
    stop_signal = True
```

Odată ce nu mai există erori la inserarea datelor, se șterge eventual tot textul din View-urile de sender și receiver, butonul de start este activat și butonul de stop este pornit. Este preluat textul de transmis din caseta text aferentă(entryTxt), și sunt pregătite 2 thread-uri pentru pornirea funcțiilor de sender și receiver. Implementarea acestora urmează să fie discutate imediat.

De-a lungul programului, a fost precizat o variabilă numită stop_signal. Aceasta are **rolul de a opri execuția funcțiilor din sender și receiver de îndată ce se apasă butonul stop**. Acesta declanșează funcția stopSimulation, care setează valoarea stop_signal pe true. Acest semnal este apoi transmis ca variabilă globală în funcțiile reprezentative celor 2 entități.

3.3. Implementarea sender-ului și receiver-ului

```
S_HOST = '127.0.0.1'  # adresa host al sender-ului
R_HOST = '127.0.0.2'  # adresa host al receiver-ului
S_PORT = 50000  # port sender
R_PORT = 50010  # port receiver

# adresa sender-ului, respectiv receiver-ului
S_ADDR = (S_HOST, S_PORT)
R_ADDR = (R_HOST, R_PORT)

```

Înainte de a descrie funcțiile specifice funcționării sender-ului și receiver-ului, în stânga sunt ilustrate adresele host și porturile folosite de sender, respectiv receiver. Adresa host și portul devin adresa efectivă a socket-urilor specifice celor 2 entități.

Sender-ul nu este definit ca obiectul unei clase, ci printr-o funcție care aplică funcționalitatea acestuia. **Funcția se numește sender** și este situată în scriptul interfață.py.

```
92  def sender(sentence, timeEnd, winSize):
93      global stop_signal
94      global S_HOST
95      global S_PORT
96      global S_ADDR
97
98      # s_sender va fi socket-ul de comunicatie de la sender catre receiver; se va transmite prin datagrame UDP
99      s_sender = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
100
101     # fiind portul principal de transmitere, va fi conectat la un PORT, la adresa ip S_HOST
102     # se gaseste un port liber dintre cele disponibile
103     while True:
104         try:
105             s_sender.bind(S_ADDR)
106             break
107         except socket.error as e:
108             if e.errno == errno.EADDRINUSE:
109                 print(f"<SENDER> Portul {S_ADDR[1]} este deja folosit!")
110                 S_PORT += 10
111                 S_ADDR = (S_HOST, S_PORT)
112             else:
113                 print(e)
114                 break
115
```

Funcția are 3 parametrii, ale căror valori sunt obținute din casetele de completat din interfață:

- sentence – fraza de trimis către receiver; este preluată din caseta text entryTxt;
- timeEnd – timpul de așteptare al confirmării unui pachet; când expiră timpul, se mai trimite încă o dată același pachet; provine din caseta text entryTimeout;
- winSize – dimensiunea ferestrei sender-ului; informația este furnizată din caseta text entryWinSize.

La început, este creat socket-ul de comunicație al sender-ului, iar pentru a putea ca acesta să transmită pachete către receiver, este setat pe adresa efectivă S_ADDR.

Pentru a facilita utilizarea multiplă a aplicației în mai multe instanțe, se încearcă setarea pe adresa inițială. În cazul în care aceasta este folosită deja, mai exact dacă portul este deja folosit, se schimbă acesta pe alt port posibil. Se continuă procesul până când se găsește un port neocupat.

```
116     timeout = timeEnd  # timpul, in milisecunde, in care socket-ul asteapta un pachet
117     window_size = winSize  # dimensiunea ferestrei glisante a transmitatorului
118     pack_size = 30  # dimensiunea pachetului (lungimea maxima a sirului de caractere din pachetul sender-ului)
119
120     pack_s = package('info', '', 0)  # pachetul trimis de sender este de tip informatie(un sir de caractere)
121     pack_r = package('ack', 1, 0)  # pachetul primit de la receiver va fi de tip ack
122     window_s = []  # buffer-ul sender-ului (va contine window_size siruri de caractere de lungime pack_size)
123     seq_num = 0  # pozitia sirurului de caractere din vectorul de propozitii
124     # util in buffer, pentru a simula glisarea ferestrei
125
126     timeout_threads = []  # vector de thread-uri pentru fiecare element din buffer
127
128     # un thread de aici este utilizat pentru trimiterea unui nou pachet cu informatii cand timer-ul timeout expira
```

Odată realizată setarea socket-ului pentru sender, se inițializează restul parametrilor importanți în procesul de funcționare: dimensiunea pachetului, forma pachetelor pentru sender și receiver, fereastra glisantă și numărul de secvențe de 30 caractere(inițial 0).

În fereastra glisantă vor fi frame-uri definite în pachetul package descris anterior, în număr egal cu maxim dimensiunea ferestrei. Dimensiunea ferestrei va varia în funcție de numărul de pachete încă neconfirmate. Practic frame-urile reprezintă modul de a confirma dacă un pachet transmis de la poziția lui a fost primit sau nu de receiver.

Variabila seq_num va ajuta la simularea glisării ferestrei, adică de fiecare dată când se adaugă un nou frame în fereastră, acesta are numărul secvenței curent, iar când se vor elimina frame-uri confirmate de la prima poziție, cele cu numărul de secvență mai mic vor fi eliminate, ca și cum fereastra ar „glisa” la stânga.

Vectorul timeout_threads are rolul de a păstra thread-urile de timeout, mai exact Timer-e. Acestea sunt thread-uri care, după un timp stabilit ca prim parametru al constructorului, va executa funcția dată ca al doilea parametru. Acestea sunt folosite pentru a putea aplica ideea de retransmitere a unui pachet în cazul în care nu frame-ul specific acestuia nu primește confirmare.

```
130     # functie de trimitere a unui pachet din sender spre receiver
131     def timeout_send(pack: package):
132
133         data_snd = pack.dump_pack()
134         s_sender.sendto(data_snd, R_ADDR)
135
136         insertViewSender(f'TIMEOUT ENDED: Sent - Package: {pack_s.type} -- "{pack_s.info}" -- {pack_s.seq_num} ' +
137                          f'Address: {R_ADDR} Date: {datetime.datetime.now()}\n')
138
```

Funcția de mai sus este chiar funcția care va fi executată de un Timer. Aceasta transformă pachetul din sender primit ca parametru în format binar, iar apoi prin socket-ul lui este transmis către adresa efectivă a receiver-ului. La final, este anunțat în view-ul sender-ului din interfață transmiterea pachetului, prin inserarea unei linii, cu toate informațiile despre pachet și data și ora la care a fost trimis pachetul.

Odată inițializată fereastra, este momentul să fie introduse frame-uri noi, ce conțin câte 30 caractere sau mai puțin(ultima parte din frază). Este decupată fraza cu primele 30 caractere și introdusă în fereastră cu numărul de secvență curent atât timp cât permite fereastra și dacă după decupare, mai sunt caractere în propoziție. Dacă fraza rămasă încape complet în propoziție, atunci se pune și ultimul frame în fereastră dacă mai este loc. Pentru fiecare frame introdus, se adaugă un timer nou în vectorul de thread-uri de timeout, primul parametru (măsurat în secunde) fiind timpul de timeout in ms/1000, al doilea funcția timeout_send și ultimul argumentul funcției: pachetul sender-ului în care s-au pus deja textul și numărul de secvență al frame-ului. Inițial, pachetele introduse nu sunt confirmate(au membrul is_ack false).

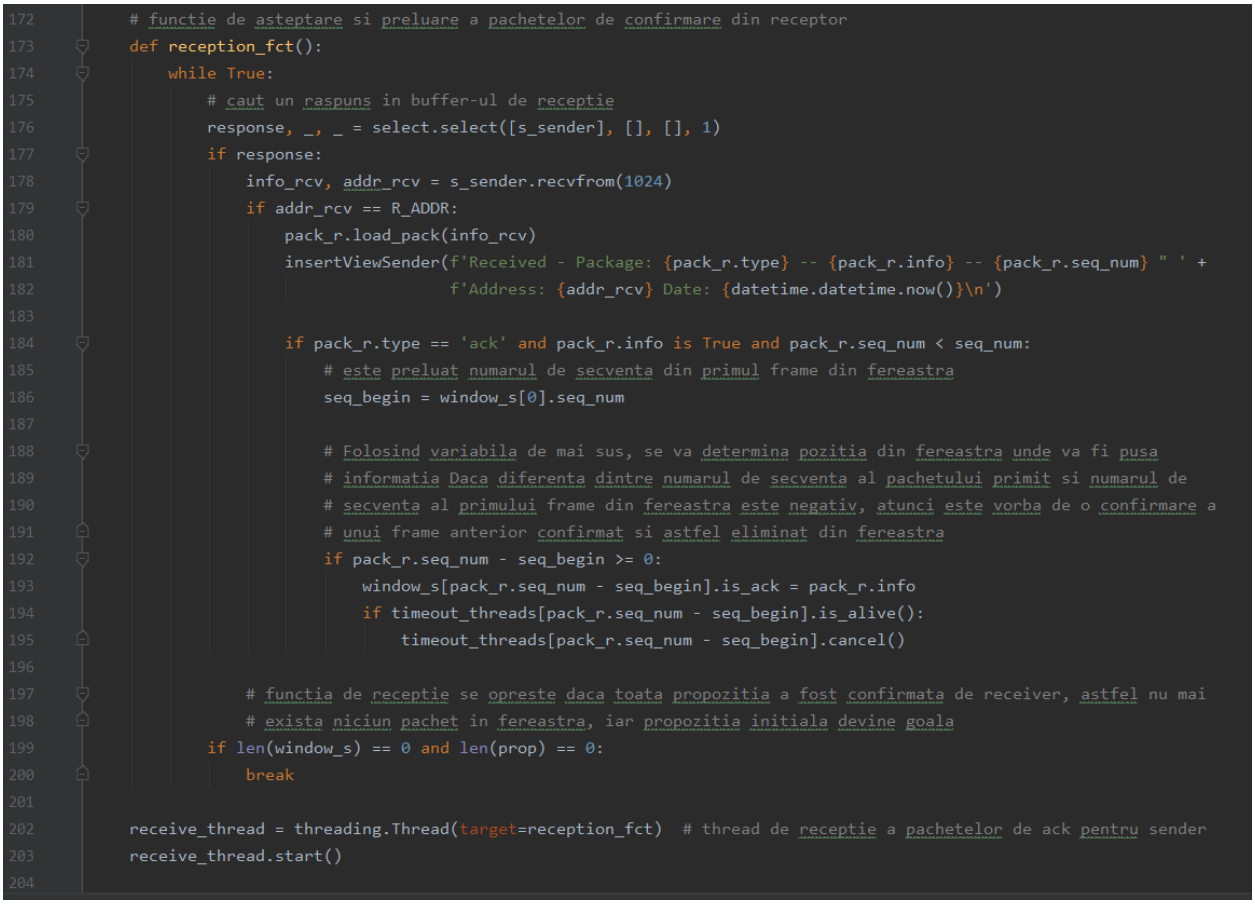
```
140         prop = sentence
141
142         # punem propozitia in buffer pana cand acesta este plin(dimensiunea lui egala cu dimensiunea ferestrei) sau pana
143         # cand nu mai avem ce pune din propozitie
144         while len(window_s) != window_size and len(prop) != 0:
145
146             # daca propozitia nu poate incapea complet intr-o zona din buffer
147             if len(prop) > pack_size:
148
149                 # punem cate o parte din propozitie intr-un frame cat ne permite zona de buffer(maxim pack_size octeti pe
150                 # zona)
151                 window_s.append(frame(prop[:pack_size], False, seq_num))
152                 # Frame-ul consta intr-un sir de caractere, un boolean ce va determina daca frame-ul a fost primit in
153                 # receiver si pozitia in secventa de trimitere a sirurilor spre receiver
154                 prop = prop[pack_size:]
155
156             # daca propozitia incape complet in zona de buffer
157             else:
158                 window_s.append(frame(prop, False, seq_num))
159                 prop = ''
160
161             # pentru fiecare frame introdus este atribuit un thread de timeout(atunci cand se trimite frame-ul sub forma
162             # de pachet catre receiver, daca nu primeste de la acesta in timp de timeout/1000 secunde un pachet de ack
163             # corespunzator frame-ului, atunci se va trimite din nou acelasi pachet catre receiver, iar thread-ul se
164             # incheie)
165             pack_s.info = window_s[-1].info
166             pack_s.seq_num = window_s[-1].seq_num
167             timeout_threads.append(threading.Timer(timeout / 1000, timeout_send, args=(pack_s,)))
168
169             # marim numarul de secvente introduse in total in fereastra
170             seq_num += 1
171
```

Procedeul de inserare inițială a frame-urilor a ferestrei din sender

Pentru a fi receptate pachete de confirmare(ack) pentru frame-urile din fereastră, este realizată funcția reception_fct(). Aceasta așteaptă încontinuu astfel de pachete până când fereastra este goală și când nu mai este nimic de trimis din frază, considerându-se în consecință că toate pachetele au fost confirmate.

Când în buffer-ul socket-ului se găsește reprezentarea binară a unu pachet ack, acesta este recepționat, iar dacă acesta vine de la adresa receiver-ului, este convertit din binar în pachet. După transcrierea în view-ul sender-ului a recepționării pachetului, se verifică dacă pachetul primit este cu adevărat tip ack, dacă informația spune true și dacă numărul de secvență specifică pachetului nu depășește numărul curent de secvențe(dacă nu a fost deja confirmat). Dacă da, atunci frame-ul cu numărul de secvență corespunzător pachetului este confirmat, iar timer-ul de timeout specific frame-ului este oprit.

Subprogramul va funcționa în paralel cu trimiterea de pachete din sender către receiver, deci este creat și pornit un thread de recepție al pachetelor ack.



```
    # functie de asteptare si preluare a pachetelor de confirmare din receptor
    def reception_fct():
        while True:
            # caut un raspuns in buffer-ul de receptie
            response, _, _ = select.select([s_sender], [], [], 1)
            if response:
                info_rcv, addr_rcv = s_sender.recvfrom(1024)
                if addr_rcv == R_ADDR:
                    pack_r.load_pack(info_rcv)
                    insertViewSender(f'Received - Package: {pack_r.type} -- {pack_r.info} -- {pack_r.seq_num} " ' +
                                     f'Address: {addr_rcv} Date: {datetime.datetime.now()}\n')

                    if pack_r.type == 'ack' and pack_r.info is True and pack_r.seq_num < seq_num:
                        # este preluat numarul de secventa din primul frame din fereastra
                        seq_begin = window_s[0].seq_num

                        # Folosind variabila de mai sus, se va determina pozitia din fereastra unde va fi pusa
                        # informatia Daca diferenta dintre numarul de secventa al pachetului primit si numarul de
                        # secventa al primului frame din fereastra este negativ, atunci este vorba de o confirmare a
                        # unui frame anterior confirmat si astfel eliminat din fereastra
                        if pack_r.seq_num - seq_begin >= 0:
                            window_s[pack_r.seq_num - seq_begin].is_ack = pack_r.info
                            if timeout_threads[pack_r.seq_num - seq_begin].is_alive():
                                timeout_threads[pack_r.seq_num - seq_begin].cancel()

                # functia de receptie se opreste daca toata propozitia a fost confirmata de receiver, astfel nu mai
                # exista niciun pachet in fereastra, iar propozitia initiala devine goala
                if len(window_s) == 0 and len(prop) == 0:
                    break

    receive_thread = threading.Thread(target=reception_fct)  # thread de receptie a pachetelor de ack pentru sender
    receive_thread.start()
```

Funcția de recepție a pachetelor ack din receiver și pornirea thread-ului aferent

Imediat începe procesul de trimitere a pachetelor folosind frame-urile din fereastră. Acesta continuă cât timp fereastra nu este goală sau încă mai sunt de trimis fragmente din frază sau dacă semnalul de stop provenit de la apăsarea butonului stop este inactiv(are valoarea false). Ultima condiție se verifică întotdeauna la începutul iterației, iar în cazul în care nu mai este validă, se opresc toate timer-ele active, iar thread-ul de recepție a pachetelor ack se așteaptă să primească ultimele confirmări.

Pentru fiecare frame din fereastră, este trimis pachetul și este pornit apoi timer-ul specific poziției frame-ului dacă nu a fost primită încă confirmarea pentru acesta și dacă nu a fost deja pornit timer-ul. În cazul în care frame-ul curent deja a primit confirmarea, iar timer-ul este încă activ, acesta este oprit.

După parcurgerea frame-urilor, se șterge din fereastră prima poziție cu pachetul confirmat de receiver, până când în prima poziție se găsește un pachet încă neconfirmat. Astfel se păstrează fără probleme ordinea de transmisie a frazelor, iar în receiver se va găsi în ordine aceasta.

Pe urmă se introduc frame-uri noi exact ca la inserția inițială a acestora în fereastra de sender, ținându-se seama încă o dată de capacitatea curentă a ferestrei, de cât a mai rămas de trimis din frază și de numărul de secvență curent al frame-ului de introdus următor.

```
    while True:
        try:
            # in cazul in care este oprit sender-ul din afara(prin intermediul butonului stop din interfata)
            if stop_signal:
                for x in timeout_threads:
                    if x.is_alive():
                        x.cancel()
                receive_thread.join()
                break

            # De fiecare data in fereastra se trimite pentru fiecare frame un pachet de tip info, ce contine atat
            # informatia stocata in frame, cat si numarul de secventa a ferestrei.
            for i in range(len(window_s)):
                # Daca nu a pornit inca thread-ul de timeout(Timer-ul) sau timpul de asteptare al pachetului de
                # confirmare specific Timer-ului a expirat, mai intai este trimis din nou pachetul catre receiver,
                # apoi Timer-ul este pornit pentru pachetul trimis
                if not timeout_threads[i].is_alive() and window_s[i].is_ack is False:
                    pack_s.info = window_s[i].info
                    pack_s.seq_num = window_s[i].seq_num
                    dumped_pack_snd = pack_s.dump_pack()
                    s_sender.sendto(dumped_pack_snd, R_ADDR)

                    # fisier log pentru verificarea transmisiei/receptiei pachetelor pentru sender
                    insertViewSender(f'Sent - Package: {pack_s.type} -- "{pack_s.info}" -- {pack_s.seq_num} ' +
                                     f'Address: {R_ADDR} Date: {datetime.datetime.now()}\n')
                    try:
                        timeout_threads[i].start()
                    except RuntimeError:
                        pass

                # Daca frame-ul a primit confirmarea prin primirea pachetului de ack ca a fost preluat de receiver,
                # oprim thread-ul de timeout doar daca acesta inca lucreaza
                if window_s[i].is_ack is True and timeout_threads[i].is_alive():
                    timeout_threads[i].cancel()
```

Parcurgerea frame-urilor din fereastră și pornirea sau, după caz, dezactivarea timer-elor

```
            # Scoatem din fereastra, incepand de la prima pozitie, toate frame-urile confirmate, impreuna cu
            # thread-urile de asteptare aferente, pana cand gasim un frame inca in asteptare
            while len(window_s) != 0 and window_s[0].is_ack is True:
                window_s.pop(0)
                if timeout_threads[0].is_alive():
                    timeout_threads[0].cancel()
                timeout_threads.pop(0)

            # umplem fereastra cu frame-uri ce contin bucati ramase din informatia prop pana cand toata este pusa sau
            # buffer-ul este plin
            while len(window_s) != window_size and len(prop) != 0:
                # daca propozitia nu poate incapea complet intr-o zona din buffer
                if len(prop) > pack_size:
                    # punem cate o parte din propozitie intr-un frame cat ne permite zona de buffer(maxim pack_size
                    # octeti pe zona)
                    window_s.append(frame(prop[:pack_size], False, seq_num))
                    prop = prop[pack_size:]

                # daca propozitia incape complet in zona de buffer
                else:
                    window_s.append(frame(prop, False, seq_num))
                    prop = ''

                # pentru noul frame pus in fereastra, este adaugat un thread de timeout pentru acesta
                pack_s.info = window_s[-1].info
                pack_s.seq_num = window_s[-1].seq_num
                timeout_threads.append(threading.Timer(timeout / 1000, timeout_send, args=(pack_s,)))

                # marim numarul de secvente introduse in total in fereastra
                seq_num += 1
```

Ștergerea de la început al frame-urilor confirmate și reumplerea ferestrei cu alte frame-uri

```
274                 if len(window_s) == 0 and len(prop) == 0:
275                     for x in timeout_threads:
276                         if x.is_alive():
277                             x.cancel()
278                     receive_thread.join()
279                     insertViewSender("Sender finished successfully!")
280                     break
281 
282             except KeyboardInterrupt:
283                 break
284         s_sender.close()
285 
```

La finalul funcției sender, se încheie iterația de transmitere de pachete după primele 2 condiții precizate anterior, iar în caz afirmativ, se procedează ca și atunci când semnalul de stop_signal este true. După finalizarea iterației, se închide și socket-ul folosit de sender.

Receiver-ul și implementarea acestuia sunt oferite de **funcția receiver** din același script ca și sender-ul. Funcția are aceeași 2 parametrii ca și funcția precedentă: dimensiunea ferestrei glisante și timpul de timeout, dar al treilea parametru reprezintă șansa de a pierde pachetul recepționat. La fel ca la entitatea descrisă anterior, se începe prin a crea un socket pentru receiver, căruia se caută un port liber pentru a-i putea fi atribuit socket-ului adresa efectivă.

```
287 def receiver(winSize, timeEnd, failChance):
288     global stop_signal
289     global R_HOST
290     global R_PORT
291     global R_ADDR
292 
293     # socket-ul receiver-ului
294     s_receiver = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
295     # se conecteaza socketul la adresa cu portul disponibil(pentru a putea trimite packete de ack)
296     while True:
297         try:
298             s_receiver.bind(R_ADDR)
299             break
300         except socket.error as e:
301             if e.errno == errno.EADDRINUSE:
302                 print(f"<RECEIVER> Portul {R_ADDR[1]} este deja folosit!")
303                 R_PORT += 10
304                 R_ADDR = (R_HOST, R_PORT)
305             else:
306                 print(e)
307                 break
308 
```

Inițializarea parametrilor simulării receiver-ului sunt asemănători cu cei de la sender, dar în plus se adaugă un timp de timeout pentru receiver (rcv_timeout), exprimat în secunde, fiind dublul timpului de timeout obișnuit pentru un pachet din sender. Astfel receiver-ul așteaptă dublul timpului de timeout un

pachet dacă nu ajunge altul nou în buffer-ul socket-ului, asigurându-se că are timp măcar un pachet să ajungă în acesta.

Se mai adaugă și un vector de bucăți primite de receiver de-a lungul simulării(sentence_pcs) și fraza finală primită(sentence_rcv), care va fi formată, după încheierea simulării, din toate fragmentele din sentence_pcs. Fiecare poziție a fragmentelor din sentence_pcs corespunde numărului de secvență a frame-ului din fereastră.

```
309     window_size = winSize  # dimensiunea ferestrei glisante a receptorului
310     failure_chance = failChance  # sansa ca un pachet sa nu ajunga la receptor ( 0.1 = 10% )
311     pack_size = 30  # dimensiunea pachetului (lungimea maxima a sirului de caractere din pachetul sender-ului)
312     timeout = timeEnd  # timpul, in milisecunde, in care socket-ul asteapta un pachet(pentru sender)
313     rcv_timeout = timeout * 2 / 1000  # timpul de asteptare in secunde a unui nou pachet pentru receiver pana se inchide
314
315     pack_s = package('info', '', 0)  # pachetul primit de la sender este de tip informatie(un sir de caractere)
316     pack_r = package('ack', True, 0)  # pachetul trimis de receiver va fi de tip ack
317     window_r = []  # buffer-ul(fereastra) receiver-ului
318     sentence_pcs = []  # vectorul in care se va stoca parti din informatia transmisa
319     sentence_rcv = ''  # propozitia finala, compusa dupa primirea tuturor partilor (si cand buffer-ul este gol)
320     seq_num = 0  # pozitia sirurului de caractere din vectorul de propozitii
321     # util in buffer, pentru a simula glisarea ferestrei
322
```

Inițializarea parametrilor simulării receiver-ului

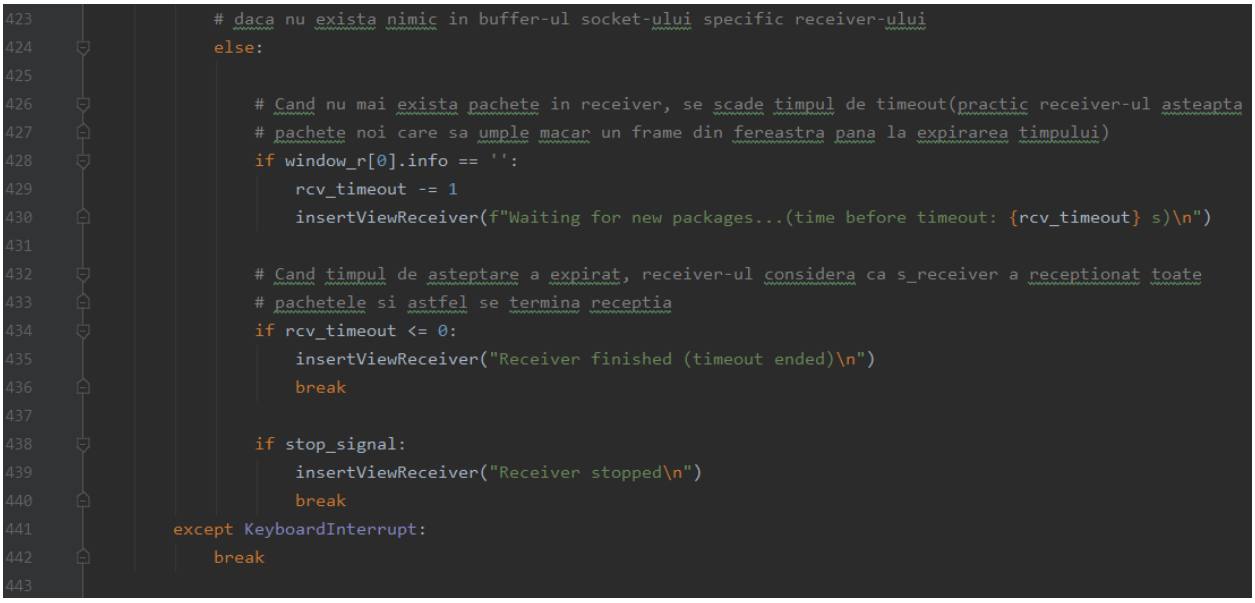
Fereastra va fi umplută 10 frame-uri care nu conțin nici un text, neconfirmate și cu numărul de secvență curent, care se incrementează la fiecare frame introdus.

```
323         # initial fereastra va fi umpluta cu frame-uri cu informatii nule si fara confirmare cu ack
324         # iar vectorul de parti ale propozitiei primite va fi umplut cu valori nule
325         for i in range(window_size):
326             window_r.append(frame('', False, seq_num))
327             seq_num += 1
328             sentence_pcs.append('')
329
```

Urmează iterația de funcționare a receiver-ului, care începe prin verificarea existenței unui răspuns din buffer-ul socket-ului, corespunzător existenței unui pachet în acesta. Dacă acesta nu găsește nici un răspuns în receiver, se scade timpul de timeout cu o unitate(secundă) dacă și fereastra conține doar frame-uri cu texte goale(nu a primit prin pachete până atunci nici un text din sender). Iterația chiar se încheie dacă timpul de așteptare al receiver-ului a expirat sau dacă semnalul de stop venit de la apăsarea butonului stop este activ.

```
330     while True:
331         # in cazul in care este oprit receiver-ul din afara(prin intermediul butonului stop din interfata)
332         # se combina propozitiile din pachetele primite pana acum si se afiseaza rezultatul in view-ul receiver-ului
333
334         try:
335             # caut un raspuns in buffer-ul de receptie
336             response, _, _ = select.select([s_receiver], [], [], 1)
337
```

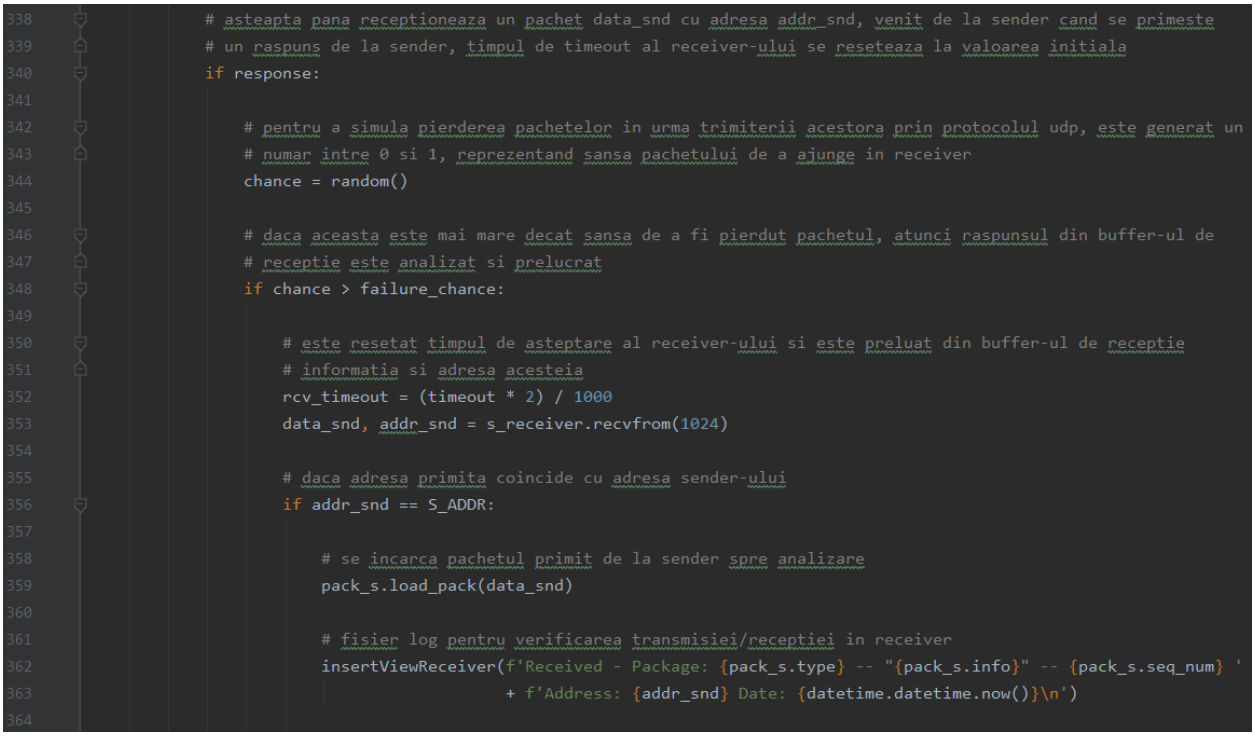
Preluarea răspunsului din buffer-ul socket-ului



```
            # daca nu exista nimic in buffer-ul socket-ului specific receiver-ului
            else:

                # Cand nu mai exista pachete in receiver, se scade timpul de timeout(practic receiver-ul asteapta
                # pachete noi care sa umple macar un frame din fereastra pana la expirarea timpului)
                if window_r[0].info == '':
                    rcv_timeout -= 1
                    insertViewReceiver(f"Waiting for new packages...(time before timeout: {rcv_timeout} s)\n")

                # Cand timpul de asteptare a expirat, receiver-ul considera ca s_receiver a receptionat toate
                # pachetele si astfel se termina receptia
                if rcv_timeout <= 0:
                    insertViewReceiver("Receiver finished (timeout ended)\n")
                    break

                if stop_signal:
                    insertViewReceiver("Receiver stopped\n")
                    break
        except KeyboardInterrupt:
            break

```

Cazul în care nu există răspuns în buffer-ul socket-ului

Dacă există răspuns în buffer, se va determina dacă pachetul s-a „pierdut” prin generarea unui număr între 0 și 1. Dacă numărul generat este mai mare decât șansa de a pierde pachetul, înseamnă că a ajuns cu succes un pachet, iar dacă acest pachet provine de la adresa efectivă a receiver-ului, atunci este și prelucrat, și timpul de timeout al receiver-ului se resetează la valoarea inițială.



```
        # asteapta pana receptioneaza un pachet data_snd cu adresa addr_snd, venit de la sender cand se primeste
        # un raspuns de la sender, timpul de timeout al receiver-ului se reseteaza la valoarea initiala
        if response:

            # pentru a simula pierderea pachetelor in urma trimiterii acestora prin protocolul udp, este generat un
            # numar intre 0 si 1, reprezentand sansa pachetului de a ajunge in receiver
            chance = random()

            # daca aceasta este mai mare decat sansa de a fi pierdut pachetul, atunci raspunsul din buffer-ul de
            # receptie este analizat si prelucrat
            if chance > failure_chance:

                # este resetat timpul de asteptare al receiver-ului si este preluat din buffer-ul de receptie
                # informatia si adresa acesteia
                rcv_timeout = (timeout * 2) / 1000
                data_snd, addr_snd = s_receiver.recvfrom(1024)

                # daca adresa primita coincide cu adresa sender-ului
                if addr_snd == S_ADDR:

                    # se incarca pachetul primit de la sender spre analizare
                    pack_s.load_pack(data_snd)

                    # fisier log pentru verificarea transmisiei/receptiei in receiver
                    insertViewReceiver(f'Received - Package: {pack_s.type} -- "{pack_s.info}" -- {pack_s.seq_num} '
                                       + f'Address: {addr_snd} Date: {datetime.datetime.now()}\n')

```

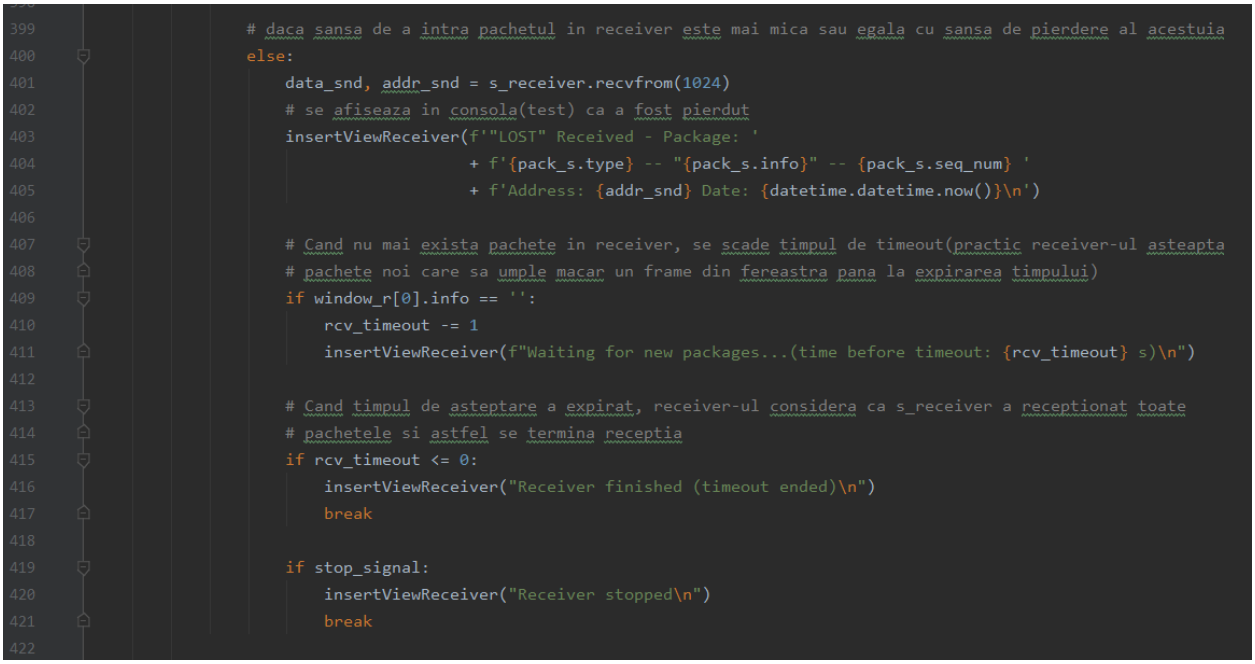
Odată convertit pachetul din forma binară în forma concretă, se verifică dacă pachetul primit este de tip text(info), dimensiunea textului din pachet este mai mică sau egală cu 30 caractere și dacă numărul de secvență se încadrează în fereastra receiver-ului. Dacă nu se încadrează, ori este trimis prea devreme, ori frame-ul care a trimis pachetul deja a primit confirmarea.

În cazul încadrării în fereastră, sunt introduse în frame-ul din fereastră și în poziția fragmentului de frază textul din pachet la numărul de secvență dat de acesta, iar frame-ul este de asemenea confirmat. Imediat este creat un pachet de ack, cu numărul de secvență al pachetului primit și acceptat, și este convertit în format binar și trimis prin socket către adresa efectivă a sender-ului.

Elementul de confirmare al frame-ului va ajuta la eliminarea primelor frame-uri până când primul rămas încă mai are de primit. Tot atunci se inserează o nou frame confirmat și un fragment gol de propoziție, cu numărul de secvență curent, incrementat la fiecare nou frame.

```
                # daca pachetul contine un fragment de propozitie( type = info )
                # si secventa de unde provine nu depaseste numarul maxim curent de secvente(seq_num)
                if pack_s.type == 'info' and pack_s.seq_num < seq_num and pack_size >= len(pack_s.info):

                    # este preluat numarul de secventa din primul frame din fereastra
                    seq_begin = window_r[0].seq_num

                    # folosind variabila de mai sus, se va determina pozitia din fereastra unde va fi pusa
                    # informatia, iar acolo este confirmat ca a fost primit, daca numarul de secventa al
                    # pachetului nu este inainte de numarul de secventa al primului frame din fereastra
                    if pack_s.seq_num - seq_begin >= 0:
                        window_r[pack_s.seq_num - seq_begin].info = pack_s.info
                        window_r[pack_s.seq_num - seq_begin].is_ack = True
                        sentence_pcs[pack_s.seq_num] = pack_s.info

                        # odata pusa informatia in vectorul de parti de date, se transmite catre sender un
                        # pachet de ack la secventa de unde a venit partea
                        pack_r.seq_num = pack_s.seq_num
                        dumped_pack_rcv = pack_r.dump_pack()
                        s_receiver.sendto(dumped_pack_rcv, S_ADDR)

                        insertViewReceiver(f'Sent - Package: '
                                           + f'{pack_r.type} -- {pack_r.info} -- {pack_r.seq_num} '
                                           + f'Address: {S_ADDR} Date: {datetime.datetime.now()}\n')

                # Scoatem din fereastra, incepand de la prima pozitie, toate frame-urile confirmate pana cand
                # gasim un frame inca in asteptare. De asemenea, adaugam in locul lor, la finalul ferestrei,
                # un nou frame cu informatie goala
                while window_r[0].is_ack:
                    window_r.pop(0)
                    window_r.append(frame('', False, seq_num))
                    sentence_pcs.append('')
                    seq_num += 1
```

Dacă în schimb șansa de ajunge pachetul deja aflat în buffer-ul socket-ului este mai mică sau egală cu șansa de a pierde pachetul în timpul transmisiei, se preia pachetul doar pentru a marca în view-ul receiver-ului că acesta a fost pierdut la transmisie, iar timpul de timeout al receiver-ului, cât și momentul de oprire al programului se tratează la fel ca în lipsa unui răspuns în buffer.



```
            # daca sansa de a intra pachetul in receiver este mai mica sau egala cu sansa de pierdere al acestuia
            else:
                data_snd, addr_snd = s_receiver.recvfrom(1024)
                # se afiseaza in consola(test) ca a fost pierdut
                insertViewReceiver(f'"LOST" Received - Package: '
                                   + f'{pack_s.type} -- "{pack_s.info}" -- {pack_s.seq_num} '
                                   + f'Address: {addr_snd} Date: {datetime.datetime.now()}\n')

                # Cand nu mai exista pachete in receiver, se scade timpul de timeout(practic receiver-ul asteapta
                # pachete noi care sa umple macar un frame din fereastra pana la expirarea timpului)
                if window_r[0].info == '':
                    rcv_timeout -= 1
                    insertViewReceiver(f"Waiting for new packages...(time before timeout: {rcv_timeout} s)\n")

                # Cand timpul de asteptare a expirat, receiver-ul considera ca s_receiver a receptionat toate
                # pachetele si astfel se termina receptia
                if rcv_timeout <= 0:
                    insertViewReceiver("Receiver finished (timeout ended)\n")
                    break

                if stop_signal:
                    insertViewReceiver("Receiver stopped\n")
                    break
```

Tratarea cazului de „pierdere” a pachetului în timpul transmisiei