

Rețele de Calculatoare

-Selector de noduri-

Student: Mătiș Oana-Antonia

Grupa: 30238

Cuprins

1. Enunțul problemei	3
2. Soluția propusă	4
3. Proiectare și implementare	6
4. Rezultate obținute.....	9
5. Analiza WireShark	12
6. Concluzii	16

1. Enunțul problemei

3.2 Selector de noduri (Node selector)

- Există trei noduri în topologie: N1, N2, N3 (Figura 8.9)
- N1 crește o valoare de 100 de ori și după fiecare incrementare trimite valoarea fie către N2 sau către N3, noduri care sunt selectate aleatoriu pentru transmitere
- Când N2 primește o valoare întreagă care este un multiplu de 3, va trimite înapoi un pachet ACK către N1
- Când N3 primește o valoare întreagă care este un multiplu de 5, va trimite înapoi un pachet ACK către N1
- Sugestii de implementare:
 - Implementați o singură clasă pentru N2 și N3 care este instanțiată cu diferiți parametri de comunicare (reutilizați codul și nu-l duplicați pentru fiecare instanță)
 - Toată comunicarea folosește socket-uri UDP (opțional)

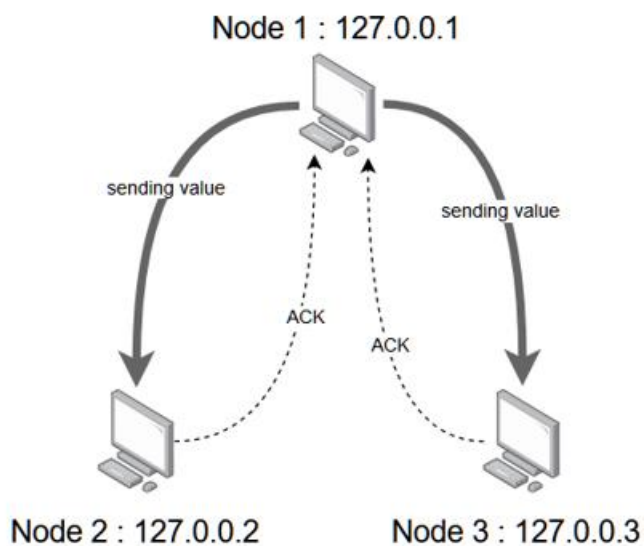


Figura 8.9 Topologia rețelei pentru selector de noduri

Se cere: Implementarea simulării unei rețele pentru un selector de noduri în cadrul căruia comunicarea are loc între trei noduri: N1, N2, N3. Nodul N1 va trimite valori către N2 și N3, aleatoriu. Dacă N2 primește un multiplu de 3, va trimite un ACK la N1, iar N3 va trimite un ACK la N1 pentru multiplii de 5. Se utilizează o singură clasă pentru N2 și N3 și se folosește socket-ul UDP pentru comunicare. Testarea se face cu Wireshark și se prezintă o captură care

arată comunicarea între adresele IP ale nodurilor. Analizarea raportului de trafic și a lungimii pachetelor se realizează prin utilizarea statisticilor WireShark sau a inspecției manuale.

2. Soluția propusă

În ceea ce privește limbajul de programare pentru realizarea cerinței, am ales Java deoarece:

- Java oferă o bibliotecă standard puternică pentru lucrul cu rețelele și socket-urile. În cod am utilizat clasele "DatagramSocket" și "DatagramPacket" pentru a permite comunicarea între noduri prin UDP. Aceste clase abstractizează detalii de nivel scăzut ale rețelei.
- Java este un limbaj orientat pe obiecte, iar codul este structurat în clase și obiecte pentru a organiza și a abstractiza funcționalitățile. Clasele "NodeSelector" și "NodeACK" sunt definite pentru a reprezenta nodurile din rețea și pentru a gestiona ACK-urile.

```
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, destinationAddress, destinationPort);  
  
this.datagramSocket = new DatagramSocket(port);
```

Ca protocol de comunicare, voi utiliza UDP.

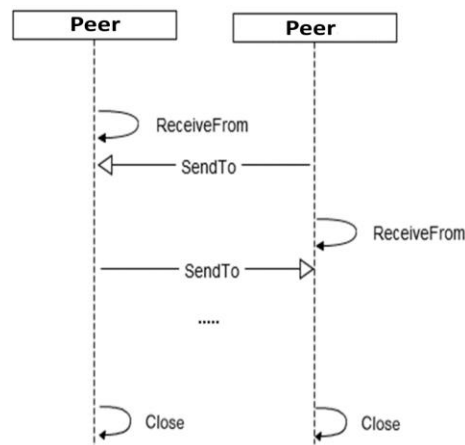
UDP (User Datagram Protocol) este un protocol de nivelul transportului în cadrul suitei de protocoale de internet (TCP/IP). UDP este un protocol simplu, fără conexiune și fără stări, care oferă un serviciu de bază de trimitere a datagramelor între aplicații pe o rețea IP.

Funcționare:

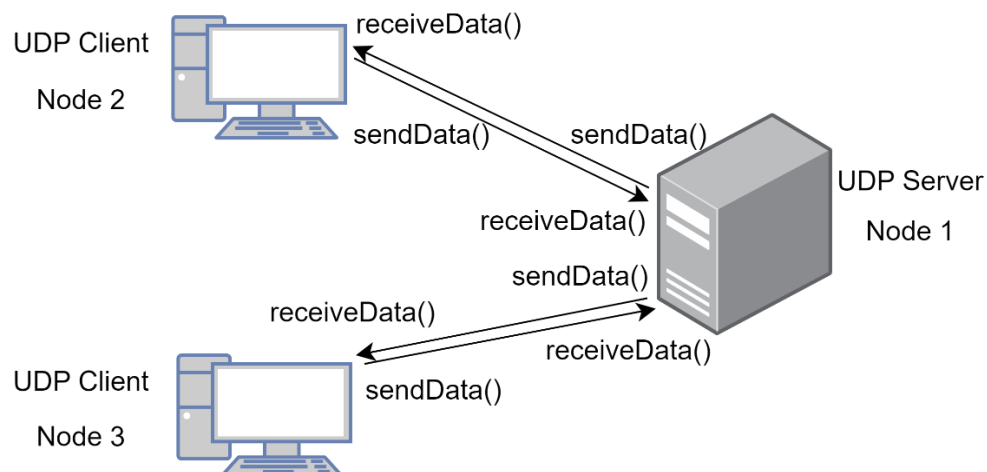
- **Fără conexiune:** UDP este un protocol fără conexiune, ceea ce înseamnă că nu este necesară stabilirea unei conexiuni prealabile între sender și receiver înainte de trimiterea datelor. Fiecare pachet de date este tratat independent și nu se menține o stare de conexiune între părți.
- **Serviciu de bază:** UDP furnizează un serviciu simplu de transmitere a datagramelor fără a garanta livrarea lor sau ordinea în care sunt livrate. Acest lucru înseamnă că pachetele pot fi pierdute sau pot ajunge la destinație în altă ordine decât au fost trimise.
- **Fără confirmare de primire (ACK):** UDP nu oferă confirmări de primire a datelor trimise (ACK), deci nu există o confirmare explicită a faptului că datele au fost livrate cu succes la destinație.

De ce este folosit UDP la selectarea nodurilor:

- **Eficiență și viteză:** UDP este mai rapid decât TCP, deoarece nu necesită stabilirea și menținerea unei conexiuni. Acest lucru este important într-o simulare în timp real a unei rețele, unde viteza de transmitere a datelor este crucială.
- **Simplitate:** UDP este mai simplu și mai ușor de implementat decât TCP, ceea ce face mai ușoară dezvoltarea și testarea unei simulări de rețea.
- **Trimiterea de pachete în mod neordonat:** În simularea selectării nodurilor, nu este strict necesar să se asigure că fiecare pachet de date este livrat la destinație sau că ajunge în ordinea corectă, deoarece comportamentul simulat poate implica unele pierderi de date sau întârzieri în rețea. Folosirea UDP permite simularea acestui comportament fără a adăuga complexitatea suplimentară a protocolului TCP.



Am realizat următoarea diagramă pentru a reprezenta comunicarea UDP specifică rețelei pentru selectorul de noduri:



Pentru a implementa corect un `NodeSelector` în cadrul unei simulări de rețea, acesta trebuie să conțină o instanță a unui server de socket, care să fie capabil să primească și să trimită date pe rețea. În cazul implementării comunicării între noduri folosind UDP, este necesar un server UDP.

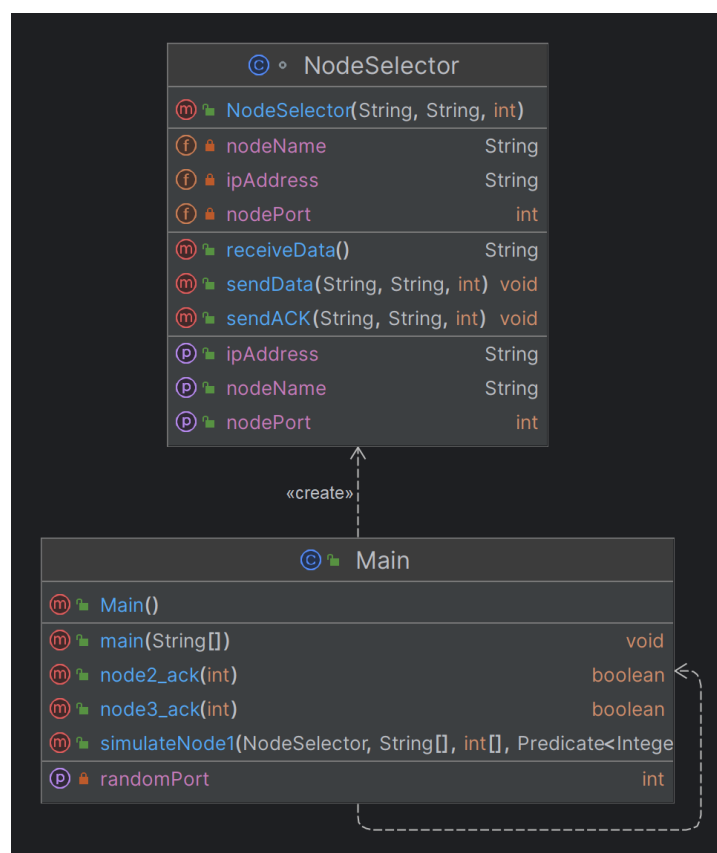
Astfel, un `NodeSelector` ar trebui să conțină o instanță a clasei `DatagramSocket` pentru a permite trimiterea și primirea datagramelor. Această instanță de socket va fi folosită pentru a comunica cu alte noduri din rețea. De asemenea, este important să se stabilească portul pe care serverul va asculta pentru a primi date și portul destinatarului la care vor fi trimise datele.

În rezumat, un `NodeSelector` ar trebui să includă:


- o instanță a clasei `DatagramSocket`.
- Setarea portului pentru ascultarea datelor primite
- Metode pentru trimiterea și primirea datelor prin socket-ul datagram.
- Informații despre numele nodului, adresa IP și alte detalii relevante pentru identificarea și gestionarea nodului în cadrul rețelei simulate.

3. Proiectare și implementare

Diagrama UML de clase reprezentativă programului scris în Java.



Prezentarea claselor

 **Clasa NodeSelector** este responsabilă pentru gestionarea funcționalității de trimitere și primire a datelor prin socket-uri UDP, precum și pentru stocarea informațiilor despre un nod, cum ar fi numele, adresa IP și portul.

Variabilele clasei:

- **nodeName**: Un șir de caractere care reprezintă numele nodului.
- **ipAddress**: Un șir de caractere care reprezintă adresa IP a nodului.
- **datagramSocket**: Obiectul DatagramSocket utilizat pentru trimiterea și primirea datelor prin socket-ul UDP.
- **nodePort**: Numărul portului asociat nodului.


Constructorul clasei:

Constructorul primește trei parametri: **nodeName**, **ipAddress** și **port**. În cadrul constructorului, un nou obiect DatagramSocket este creat pentru nod, folosind portul specificat. De asemenea, se setează un timeout pentru recepția datelor.

Metodele clasei:

- **getNodeName()**: Returnează numele nodului.
- **getIpAddress()**: Returnează adresa IP a nodului.
- **getNodePort()**: Returnează portul asociat nodului.
- **sendData(String data, String destinationIp, int destinationPort)**: Trimite date către o destinație specificată prin adresă IP și port. Datele sunt codificate folosind UTF-8 înainte de trimitere.
- **sendACK(String data, String destinationIp, int destinationPort)**: Trimite un ACK către o destinație specificată prin adresă IP și port. ACK-ul este codificat folosind UTF-8 înainte de trimitere.
- **receiveData()**: Primește date de la un alt nod. Datele primite sunt decodate folosind UTF-8 și returnate sub formă de șir de caractere. Această metodă poate returna null dacă nu se primesc date în intervalul de timeout specificat.

Această clasă este esențială pentru comunicarea între noduri în cadrul rețelei simulate și asigură gestionarea corectă a trimiterii și recepției datelor prin socket-uri UDP. De asemenea, utilizează codificarea UTF-8 pentru a asigura compatibilitatea cu alte platforme de comunicație.

 **Clasa Main** este punctul de intrare al programului și conține metodele principale pentru simularea comunicației între noduri.

Metodele clasei Main:

- **simulateNode1**: Această metodă simulează comportamentul nodului 1. Primește ca parametri un obiect NodeSelector care reprezintă nodul 1, adresele IP ale destinațiilor, porturile destinațiilor, condițiile pentru trimiterea ACK-urilor, porturile ACK-urilor și

adresele IP de destinație ale ACK-urilor. Într-un ciclu, se generează valori de date, se trimit către destinații și se trimit ACK-uri în funcție de condițiile specificate.

- **main:** Această metodă este punctul de intrare al programului. Aici sunt create instanțe ale nodurilor 1, 2 și 3 folosind clasa NodeSelector. Se stabilesc adresele IP și porturile pentru fiecare nod, apoi se inițializează variabilele necesare pentru simulare. Se apelează metoda simulateNode1 pentru nodul 1 și se așteaptă primirea ACK-urilor.
- **node2_ack și node3_ack:** Acestea sunt predicții care decid dacă un ACK trebuie trimis în funcție de valoarea dată. Pentru nodul 2, ACK-ul este trimis când valoarea este divizibilă cu 3, iar pentru nodul 3, ACK-ul este trimis când valoarea este divizibilă cu 5.
- **getRandomPort:** Această metodă generează un port aleator între 1024 și 65535.

Clasa NodeSelector este folosită pentru a gestiona funcționalitatea de trimitere și primire a datelor prin socket-urile UDP și pentru a stoca informații despre noduri, cum ar fi numele, adresa IP și portul.

Această implementare utilizează codificarea UTF-8 pentru trimiterea și primirea datelor, iar ACK-urile sunt trimise ca șiruri de caractere "ACK". De asemenea, porturile sunt generate aleator pentru a simula comportamentul real al rețelei.

La finalul simulării și compilării, programul va rula simularea comportamentului nodului 1 în rețea. Acesta va trimite date către alte noduri și va primi ACK-uri în funcție de condițiile specificate. După finalizarea simulării, programul va aștepta primirea ACK-urilor și se va opri după ce va primi primul ACK sau după ce va trece un număr de iterații specificat. Dacă programul primește un ACK pentru valoarea 100, va afișa mesajul "The program stopped after receiving ACK for value 100.", iar apoi se va încheia execuția.

4. Rezultate obținute

În urma rulării programului, voi prezenta rezultatele afișate în consolă

```
Node 1 sent 1 to 127.0.0.2:14825 (received by N2)
Node 1 sent 2 to 127.0.0.2:14825 (received by N2)
Node 1 sent 3 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 3
Node 1 sent 4 to 127.0.0.2:14825 (received by N2)
Node 1 sent 5 to 127.0.0.2:14825 (received by N2)
Node 1 sent 6 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 6
Node 1 sent 7 to 127.0.0.2:14825 (received by N2)
Node 1 sent 8 to 127.0.0.2:14825 (received by N2)
Node 1 sent 9 to 127.0.0.3:54808 (received by N3)
Node 1 sent 10 to 127.0.0.2:14825 (received by N2)
Node 1 sent 11 to 127.0.0.3:54808 (received by N3)
Node 1 sent 12 to 127.0.0.3:54808 (received by N3)
Node 1 sent 13 to 127.0.0.3:54808 (received by N3)
Node 1 sent 14 to 127.0.0.2:14825 (received by N2)
Node 1 sent 15 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 15
Node 1 sent 16 to 127.0.0.3:54808 (received by N3)
Node 1 sent 17 to 127.0.0.2:14825 (received by N2)
Node 1 sent 18 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 18
Node 1 sent 19 to 127.0.0.3:54808 (received by N3)
Node 1 sent 20 to 127.0.0.2:14825 (received by N2)
Node 1 sent 21 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 21
N2 sent ACK to Node 1 for value 21
Node 1 sent 22 to 127.0.0.2:14825 (received by N2)
Node 1 sent 23 to 127.0.0.2:14825 (received by N2)
Node 1 sent 24 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 24
Node 1 sent 25 to 127.0.0.2:14825 (received by N2)
Node 1 sent 26 to 127.0.0.2:14825 (received by N2)
Node 1 sent 27 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 27
Node 1 sent 28 to 127.0.0.2:14825 (received by N2)
Node 1 sent 29 to 127.0.0.3:54808 (received by N3)
Node 1 sent 30 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 30
Node 1 sent 31 to 127.0.0.2:14825 (received by N2)
Node 1 sent 32 to 127.0.0.2:14825 (received by N2)
Node 1 sent 33 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 33
Node 1 sent 34 to 127.0.0.3:54808 (received by N3)
Node 1 sent 35 to 127.0.0.2:14825 (received by N2)
Node 1 sent 36 to 127.0.0.3:54808 (received by N3)
Node 1 sent 37 to 127.0.0.3:54808 (received by N3)
Node 1 sent 38 to 127.0.0.3:54808 (received by N3)
Node 1 sent 39 to 127.0.0.3:54808 (received by N3)
Node 1 sent 40 to 127.0.0.3:54808 (received by N3)
N2 sent ACK to Node 1 for value 40
```

```
Node 1 sent 40 to 127.0.0.3:54808 (received by N3)
N3 sent ACK to Node 1 for value 40
Node 1 sent 41 to 127.0.0.3:54808 (received by N3)
Node 1 sent 42 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 42
Node 1 sent 43 to 127.0.0.2:14825 (received by N2)
Node 1 sent 44 to 127.0.0.3:54808 (received by N3)
Node 1 sent 45 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 45
Node 1 sent 46 to 127.0.0.2:14825 (received by N2)
Node 1 sent 47 to 127.0.0.2:14825 (received by N2)
Node 1 sent 48 to 127.0.0.3:54808 (received by N3)
Node 1 sent 49 to 127.0.0.3:54808 (received by N3)
Node 1 sent 50 to 127.0.0.2:14825 (received by N2)
Node 1 sent 51 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 51
Node 1 sent 52 to 127.0.0.3:54808 (received by N3)
Node 1 sent 53 to 127.0.0.2:14825 (received by N2)
Node 1 sent 54 to 127.0.0.3:54808 (received by N3)
Node 1 sent 55 to 127.0.0.2:14825 (received by N2)
Node 1 sent 56 to 127.0.0.3:54808 (received by N3)
Node 1 sent 57 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 57
Node 1 sent 58 to 127.0.0.3:54808 (received by N3)
Node 1 sent 59 to 127.0.0.2:14825 (received by N2)
Node 1 sent 60 to 127.0.0.3:54808 (received by N3)
```

```
N3 sent ACK to Node 1 for value 60
Node 1 sent 61 to 127.0.0.3:54808 (received by N3)
Node 1 sent 62 to 127.0.0.3:54808 (received by N3)
Node 1 sent 63 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 63
Node 1 sent 64 to 127.0.0.2:14825 (received by N2)
Node 1 sent 65 to 127.0.0.2:14825 (received by N2)
Node 1 sent 66 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 66
Node 1 sent 67 to 127.0.0.3:54808 (received by N3)
Node 1 sent 68 to 127.0.0.3:54808 (received by N3)
Node 1 sent 69 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 69
Node 1 sent 70 to 127.0.0.3:54808 (received by N3)
N3 sent ACK to Node 1 for value 70
Node 1 sent 71 to 127.0.0.3:54808 (received by N3)
Node 1 sent 72 to 127.0.0.3:54808 (received by N3)
Node 1 sent 73 to 127.0.0.2:14825 (received by N2)
Node 1 sent 74 to 127.0.0.2:14825 (received by N2)
Node 1 sent 75 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 75
Node 1 sent 76 to 127.0.0.2:14825 (received by N2)
Node 1 sent 77 to 127.0.0.3:54808 (received by N3)
Node 1 sent 78 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 78
Node 1 sent 79 to 127.0.0.2:14825 (received by N2)
Node 1 sent 80 to 127.0.0.3:54808 (received by N3)
```

```

N2 sent ACK to Node 1 for value 78
Node 1 sent 79 to 127.0.0.2:14825 (received by N2)
Node 1 sent 80 to 127.0.0.3:54808 (received by N3)
N3 sent ACK to Node 1 for value 80
Node 1 sent 81 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 81
Node 1 sent 82 to 127.0.0.2:14825 (received by N2)
Node 1 sent 83 to 127.0.0.2:14825 (received by N2)
Node 1 sent 84 to 127.0.0.3:54808 (received by N3)
Node 1 sent 85 to 127.0.0.2:14825 (received by N2)
Node 1 sent 86 to 127.0.0.3:54808 (received by N3)
Node 1 sent 87 to 127.0.0.3:54808 (received by N3)
Node 1 sent 88 to 127.0.0.2:14825 (received by N2)
Node 1 sent 89 to 127.0.0.3:54808 (received by N3)
Node 1 sent 90 to 127.0.0.2:14825 (received by N2)
N2 sent ACK to Node 1 for value 90
Node 1 sent 91 to 127.0.0.2:14825 (received by N2)
Node 1 sent 92 to 127.0.0.2:14825 (received by N2)
Node 1 sent 93 to 127.0.0.3:54808 (received by N3)
Node 1 sent 94 to 127.0.0.3:54808 (received by N3)
Node 1 sent 95 to 127.0.0.3:54808 (received by N3)
N3 sent ACK to Node 1 for value 95
Node 1 sent 96 to 127.0.0.3:54808 (received by N3)
Node 1 sent 97 to 127.0.0.3:54808 (received by N3)
Node 1 sent 98 to 127.0.0.3:54808 (received by N3)
Node 1 sent 99 to 127.0.0.3:54808 (received by N3)
Node 1 sent 100 to 127.0.0.2:14825 (received by N2)

```

```

Node 1 sent 100 to 127.0.0.2:14825 (received by N2)
The program stopped after receiving ACK for value 100.

Process finished with exit code 0

```

De asemenea, comunicarea UDP poate fi observată și în WireShark

Time	Source	Destination	Protocol	Length	Info
1369...	1390.513845	127.0.0.1	127.0.0.2	UDP	33 5958 → 14825 Len=1
1369...	1390.514025	127.0.0.1	127.0.0.2	UDP	33 5958 → 14825 Len=1
1369...	1390.514226	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1369...	1390.520310	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1369...	1390.520375	127.0.0.1	127.0.0.2	UDP	33 5958 → 14825 Len=1
1369...	1390.520477	127.0.0.1	127.0.0.2	UDP	33 5958 → 14825 Len=1
1369...	1390.520548	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1369...	1390.520580	127.0.0.1	127.0.0.2	UDP	33 5958 → 14825 Len=1
1369...	1390.520642	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1369...	1390.520710	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1369...	1390.520746	127.0.0.1	127.0.0.2	UDP	33 5958 → 14825 Len=1
1370...	1390.520813	127.0.0.1	127.0.0.2	UDP	33 5958 → 14825 Len=1
1370...	1390.520917	127.0.0.1	127.0.0.3	UDP	33 5958 → 54808 Len=1
1370...	1390.521027	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1370...	1390.521065	127.0.0.1	127.0.0.2	UDP	34 5958 → 14825 Len=2
1370...	1390.521139	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1370...	1390.521185	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1370...	1390.521255	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1370...	1390.521323	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1370...	1390.521370	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2

1371...	1390.532844	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.532967	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.533029	127.0.0.1	127.0.0.2	UDP	35 5958 → 14825 Len=3
1371...	1390.533120	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.600170	127.0.0.1	127.0.0.1	TCP	45 54006 → 52592 [PSH, ACK] Seq=6700 Ack=3832 Win=2615808 Len=1 [TCP segment of a reassembled PDU]
1371...	1390.600206	127.0.0.1	127.0.0.1	TCP	44 52592 → 54006 [ACK] Seq=3832 Ack=6701 Win=2612992 Len=0

Se poate observa că Nodul 1 (adresa sursă 127.0.0.1) trimite date atât spre Nodul 2, căruia îi corespunde adresa destinație 127.0.0.2, cât și spre Nodul 3, căruia îi corespunde adresa destinație 127.0.0.3.

În plus, porturile corespunzătoare nodurilor sunt generate random, se poate observa corespondența dintre consola din IntelliJ și WireShark în ceea ce privește numărul corespunzător fiecărui port.

1371...	1390.531801	127.0.0.1	127.0.0.2	UDP	34 5958 → 14825 Len=2
1371...	1390.531953	127.0.0.1	127.0.0.2	UDP	34 5958 → 14825 Len=2
1371...	1390.532047	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.532203	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.532257	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.532360	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.532436	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.532520	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.532568	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.532644	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.532677	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.532726	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2

> Frame 137151: 34 bytes on wire (272 bits), 34 bytes captured (272 bits) on interface \Device\NPF_{Loopback}, id 0
 > Null/Loopback
 > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.2
 > User Datagram Protocol, Src Port: 5958, Dst Port: 14825
 > Data (2 bytes)

0000 02 00 00 00 45 00 00 1e 2b 3a 00 00 80 11 00 00 ...E...+:.....
 0010 7f 00 00 01 7f 00 00 02 17 46 39 e9 00 0a 77 75F9...wu
 0020 39 32 92

În această captură se observă transmiterea unui mesaj de lungime len = 2 bytes prin comunicarea UDP de la Nodul 1 la Nodul 2, respectiv porturile 5958 și 14825.

5. Analiza WireShark

În acest capitol se vor analiza rezultatele statistice obținute în urma capturii.

Ethernet	IPv4 · 26	IPv6 · 14	TCP · 143	UDP · 399							
Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
0.0.0.0	255.255.255.255	8	3 kB	8	3 kB	0	0 bytes	84174.356412	652.9244	33 bits/s	0 bits/s
127.0.0.1	127.0.0.1	320,756	17 MB	320,756	17 MB	0	0 bytes	0.000000	84971.5606	1591 bits/s	0 bits/s
127.0.0.1	127.0.0.2	450	15 kB	450	15 kB	0	0 bytes	15.449248	1757.0872	69 bits/s	0 bits/s
127.0.0.1	127.0.0.3	450	15 kB	450	15 kB	0	0 bytes	15.429516	1757.1068	69 bits/s	0 bits/s

Imaginea de mai sus reprezintă statistica transferurilor de pachete între componentele rețelei de comunicare. Numărul total de pachete este 1 220,756, curpinzând aproximativ 17000 + 15 + 15 = 1730 kB.

Statistics

Measurement	Captured	Displayed	Marked
Packets	324314	324314 (100.0%)	—
Time span, s	85156.604	85156.604	—
Average pps	3.8	3.8	—
Average packet size, B	54	54	—
Bytes	17622785	17622785 (100.0%)	0
Average bytes/s	206	206	—
Average bits/s	1655	1655	—

Din totalul de 324 314 de pachete capturate, 1499 sunt UDP. Acest lucru se poate observa pe figura de mai jos, pe linia selectată cu albastru.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
▼ Frame	100.0	324497	100.0	17639534	1651	0	0	0	324497
▼ Null/Loopback	100.0	324497	7.4	1297988	121	0	0	0	324497
▼ Internet Protocol Version 4	99.8	323839	36.7	6477060	606	0	0	0	323839
> Internet Control Message Protocol	0.0	12	0.0	960	0	8	676	0	12
Internet Group Management Protocol	0.0	70	0.0	608	0	70	608	0	70
> Transmission Control Protocol	98.7	320256	52.2	9201939	861	239309	6467200	605	320256
▼ User Datagram Protocol	1.1	3501	0.2	28008	2	0	0	0	3501
Data	0.5	1499	0.9	155217	14	1499	155217	14	1499
Domain Name System	0.0	3	0.0	105	0	3	105	0	3
Dynamic Host Configuration Protocol	0.0	38	0.1	11632	1	38	11632	1	38
Link-local Multicast Name Resolution	0.0	30	0.0	990	0	30	990	0	30
Malformed Packet	0.1	244	0.0	0	0	244	0	0	244
Multicast Domain Name System	0.1	236	0.1	12310	1	236	12310	1	236

Poza de mai sus reprezintă distribuția datelor (Data Distributed – Payload len).

Raportul dintre totalul de bytes livrați și traficul relevant al aplicației este reprezentat de dimensiunea totală a datelor în raport cu totalul dimensiunii frame-urilor transmise.

➡ $155\,217 / 17\,639\,354 = 0,0086294 = \text{aproximativ } 0,86\%$

În continuare, se vor analiza elementele header-ului cuprinse în captură.

1371...	1390.531404	127.0.0.1	127.0.0.2	UDP	34 5958 → 14825 Len=2
1371...	1390.531251	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.531293	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.531362	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.531432	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.531475	127.0.0.1	127.0.0.2	UDP	34 5958 → 14825 Len=2
1371...	1390.531543	127.0.0.1	127.0.0.3	UDP	34 5958 → 54808 Len=2
1371...	1390.531636	127.0.0.1	127.0.0.2	UDP	34 5958 → 14825 Len=2
1371...	1390.531702	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.531743	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.531824	127.0.0.1	127.0.0.1	UDP	35 5958 → 5958 Len=3
1371...	1390.531861	127.0.0.1	127.0.0.2	UDP	34 5958 → 14825 Len=2

> Frame 137144: 34 bytes on wire (272 bits), 34 bytes captured (272 bits) on interface \Device\NPF_{Loopback}, id	0000 02 00 00 00 45 00 00 1e 2b 37 00 00 80 11 00 00E...+7.....
> Null/Loopback	0010 7f 00 00 01 7f 00 00 02 17 46 39 e9 00 0a 78 6f-F9...xo
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.2	0020 38 38	88

0100 = Version: 4	
.... 0101 = Header Length: 20 bytes (5)	
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)	
Total Length: 30	
Identification: 0x2b37 (11063)	
> 000. = Flags: 0x0	
...0 0000 0000 0000 = Fragment Offset: 0	
Time to Live: 128	
Protocol: UDP (17)	
Header Checksum: 0x0000 [validation disabled]	

```

> Frame 137144: 34 bytes on wire (272 bits), 34 bytes captured (272
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 30
    Identification: 0x2b37 (11063)
    > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.2

```

0000	02 00 00 00 45 00 00 1e 2b 37 00 00 80 11 00 00E...+7.....
0010	7f 00 00 01 7f 00 00 02 17 46 39 e9 00 0a 78 6f-F9...xo
0020	38 38	88

Elementele header-ului UDP apar în imaginea de mai jos.

```
> Frame 137144: 34 bytes on wire (272 bits), 34 bytes captured (272 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
√ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 30
        Identification: 0x2b37 (11063)
    √ 000. .... = Flags: 0x0
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..0. .... = More fragments: Not set
        ...0 0000 0000 0000 = Fragment Offset: 0
        Time to Live: 128
        Protocol: UDP (17)
        Header Checksum: 0x0000 [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 127.0.0.1
        Destination Address: 127.0.0.2
    √ User Datagram Protocol, Src Port: 5958, Dst Port: 14825
        Source Port: 5958
        Destination Port: 14825
        Length: 10
        Checksum: 0x786f [unverified]
        [Checksum Status: Unverified]
        [Stream index: 87]
    > [Timestamps]
        UDP payload (2 bytes)
    √ Data (2 bytes)
        Data: 3838
        [Length: 2]
```

0000	02 00 00 00 45 00 00 1e	2b 37 00 00 80 11 00 00E...+7.....
0010	7f 00 00 01 7f 00 00 02	17 46 39 e9 00 0a 78 6fF9...x0
0020	38 38		88

Elemente semnificative:

- **Source port:** 5958
- **Destination port:** 14 825
- **Length:** 10
- **Checksum:** 0x786f (utilizat pentru verificarea erorilor antetului și a datelor)
- **Stream index:** 87

6. Concluzii

Acest assignment a constatat în implementarea unei simulări a unei rețele de comunicare cu trei noduri, respectând diverse cerințe specifice. Am utilizat limbajul Java pentru implementare, deoarece oferă suport nativ pentru lucrul cu socket-uri și permite gestionarea ușoară a comunicației în rețea. Implementarea a implicat crearea unei clase principale NodeSelector pentru gestionarea nodurilor din rețea, precum și o clasă suplimentară NodeACK pentru gestionarea trimiterii ACK-urilor. Am folosit protocolul UDP pentru comunicarea între noduri, deoarece este mai rapid decât TCP și este ideal pentru aplicații care necesită o comunicare rapidă și eficientă, fără a pune accent pe fiabilitatea în livrare.

Simularea a implicat trimiterea de date între noduri, verificarea anumitor condiții pentru trimiterea ACK-urilor și așteptarea primirii ACK-urilor. Am testat implementarea folosind Wireshark pentru a verifica comunicarea între noduri și pentru a inspecta traficul de rețea. Concluzionând, acest assignment a oferit oportunitatea de a înțelege și de a aplica concepte precum comunicarea în rețea, gestionarea socket-urilor, precum și de a lucra cu instrumente precum Wireshark pentru diagnosticarea și testarea rețelelor de comunicație.