ДОДАТОК А

Графічний матеріал атестаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Електронних обчислювальних машин

Модифікований квест-метод аналізу оверлейних мереж з невизначеною кількістю оверлеїв

Виконав:
студент гр. СПзм-18-2
Артьомов С.В.

Керівник:
к.т.н., ст. викладач
Ткачов В.М.

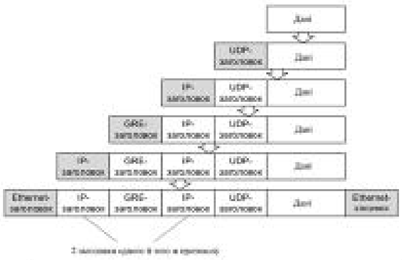
МЕТА ВИКОНАННЯ АТЕСТАЦІЙНОЇ РОБОТИ І ПОСТАНОВКА ЗАДАЧІ

З метою підвищення надійності функціонування оверлейної мережі, надбудованої над динамічно змінюваній комп`ютерній мережі, в якій кількість вузлів є нестабільним числом, необхідно модифікувати існуючий квест-метод аналізу мережі та провести відповідні дослідження ефективності його застосування з метою забезпечення анонімізації користувачів мережі.

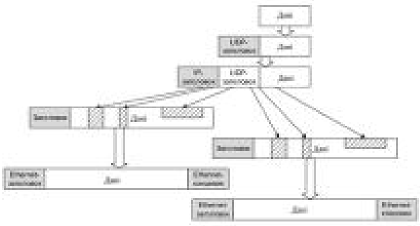
В ході виконання роботи необхідно розглянути такі питання: проаналізовати особливості побудови динамічних оверлейних мереж, розглянути методи аналізу оверлейних мереж з невизначеною кількістю оверлеїв, зокрема контурний метод та методи, що використовуються в анонімних мережах, розробити модифікований квест-метод аналізу оверлейних мереж з невизначеною кількістю оверлеїв, створити програмний продукт та провести дослідження ефективності роботи системи аналізу оверлейних мереж з невизначеною кількістю оверлеїв.

2

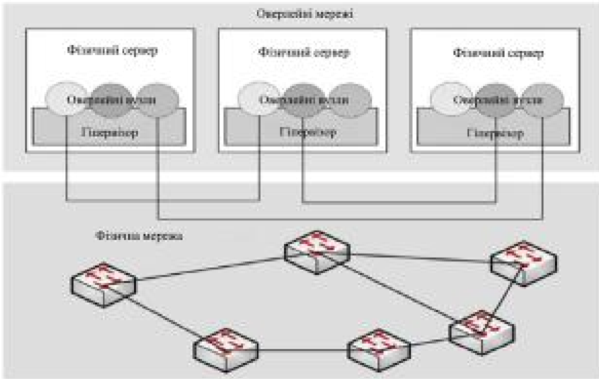
ПРИНЦИПИ ПОБУДОВИ ТА ФУНКЦІОНУВАННЯ ОВЕРЛЕЙНИХ МЕРЕЖ



Інкапсуляція пакетів при тунелюванні

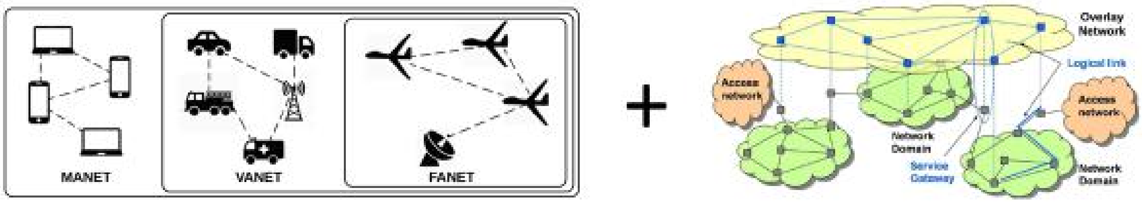


Стенографічна інкапсуляція даних в оверлейній мережі



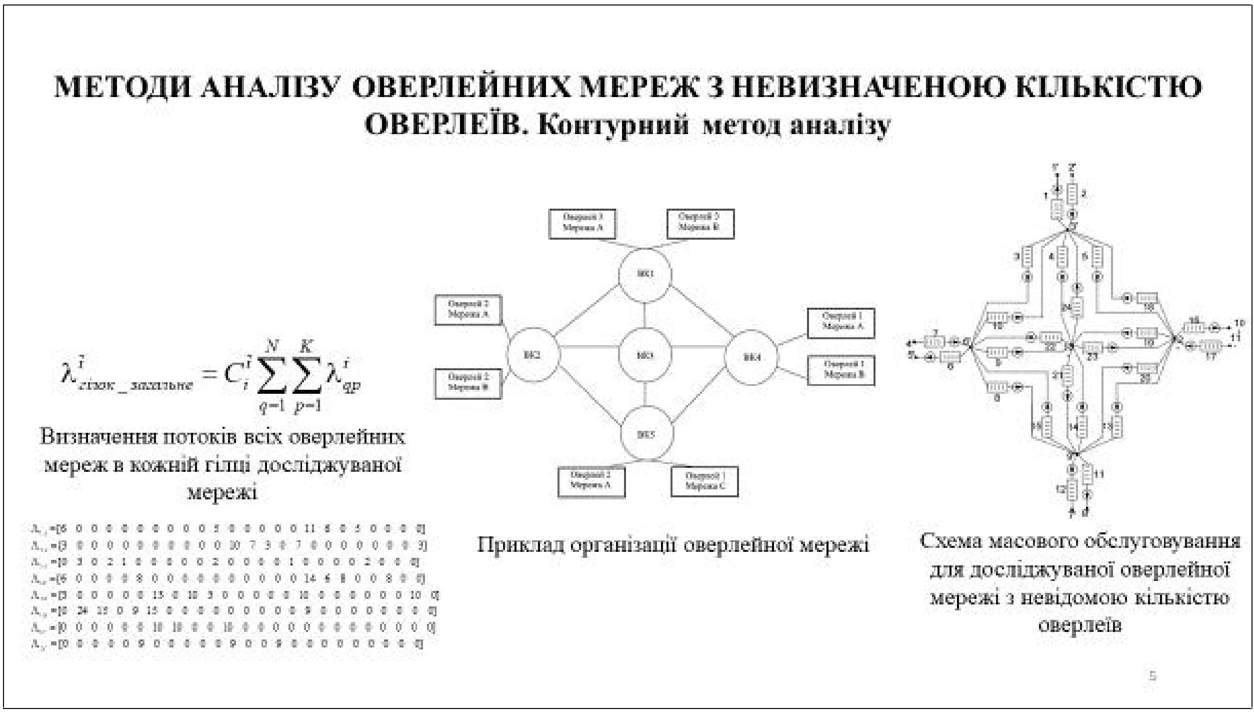
Віртуальні мережі, накладені на фізичну мережу (оверлейна інфраструктура)

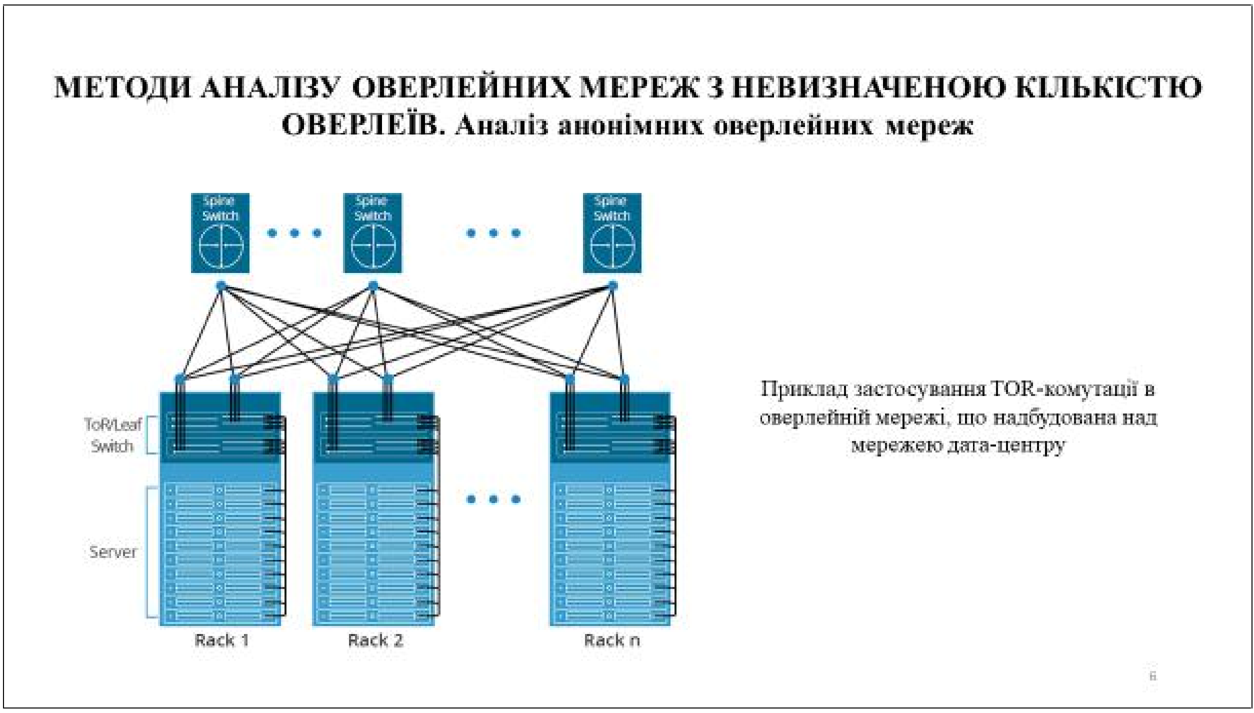
ОСОБЛИВОСТІ ДИНАМІЧНИХ ОВЕРЛЕЙНИХ МЕРЕЖ



Динамічна комп'ютерна мережа - це така мережа, яка характерізується постійною перебудова маршутів передачі даних через включення або віключення з неї учасников передачі даних. Прикладом таких мереж є MANET, FANET тощо.

Динамічна оверлейна мережа – це надбудована над динамічною комп'ютерною мережею мережа, в якій забезпечується високий рівень анонімності даних та безшовна інтеграція із зовнішніми оверлеями.


МЕТОДИ АНАЛІЗУ ОВЕРЛЕЙНИХ МЕРЕЖ З НЕВИЗНАЧЕНОЮ КІЛЬКІСТЮ ОВЕРЛЕЇВ. Контурний метод аналізу
$$\lambda^{i}_{гілок\_загальне} = C^{i}_{i} \sum_{q=1}^{N} \sum_{p=1}^{K} \lambda^{i}_{qp}$$
Визначення потоків всіх оверлейних мереж в кожній гілці досліджуваної мережі
ВК1
ВК2
ВК3
ВК4
ВК5
Приклад організації оверлейної мережі
Схема масового обслуговування для досліджуваної оверлейної мережі з невідомою кількістю оверлеїв
5


МЕТОДИ АНАЛІЗУ ОВЕРЛЕЙНИХ МЕРЕЖ З НЕВИЗНАЧЕНОЮ КІЛЬКІСТЮ ОВЕРЛЕЇВ. Аналіз анонімних оверлейних мереж
Spine Switch
Spine Switch
Spine Switch
ToR/Leaf Switch
Server
Rack 1
Rack 2
Rack n
Приклад застосування TOR-комутації в оверлейній мережі, що надбудована над мережею дата-центру
6

МОДИФІКОВАНИЙ КВЕСТ-МЕТОД АНАЛІЗУ ОВЕРЛЕЙНИХ МЕРЕЖ З НЕВИЗНАЧЕНОЮ КІЛЬКІСТЮ ОВЕРЛЕЇВ

$$\{G(\Delta t)\} = G(\Delta t_0) \to G(\Delta t_1) \to \ldots G(\Delta t_k)$$

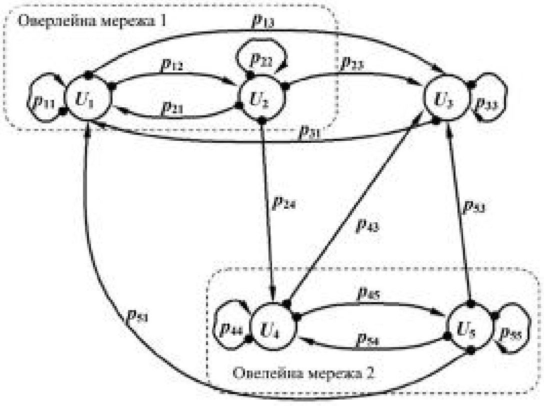
$$G(\Delta t_{k+1}) \to G(\Delta t_k) \cup \Delta G.$$

$$A(t) = \Psi\left[f(\{x_i\},\{h\},t),\xi(\{v_i\},t)\right]$$

$$f(\{x_i\},\{h\},t) = \begin{bmatrix} \{T_{1,1}^{відк}\},\{T_{1,2}^{закр}\} & \cdots & \{T_{2,m}^{відк}\},\{T_{1,m}^{закр}\} \\ \cdots & \cdots & \cdots \\ \{T_{n,1}^{відк}\},\{T_{n,2}^{закр}\} & \cdots & \{T_{n,m}^{відк}\},\{T_{n,m}^{закр}\} \end{bmatrix}$$

$$t_{закр_i} > t_{охвр_{i+i}}$$

$$t_{кор} = t_{досл} + t_{зву}$$



Послідовність подій при зміні каналу між рухомими або рухомим і стаціонарними оверлейними мережами

МОДИФІКОВАНИЙ КВЕСТ-МЕТОД АНАЛІЗУ ОВЕРЛЕЙНИХ МЕРЕЖ З НЕВИЗНАЧЕНОЮ КІЛЬКІСТЮ ОВЕРЛЕЇВ

$$p_{11} = \begin{cases} 1, P_{тек}^{i+m} > P_{пор} \\ 0 \end{cases}, \quad p_{12} = \begin{cases} 1, P_{пор} > P_{тек}^{i+m} > P_{доп} \\ 0 \end{cases}, \quad p_{13} = \begin{cases} 1, P_{тек}^{i+m} < P_{доп} \\ 0 \end{cases};$$

$$p_{22} = \begin{cases} 1, P_{тек}^{i+m} = P_{доп} \\ 0 \end{cases}, \quad p_{21} = \begin{cases} 1, P_{тек}^{i+m} \geq P_{пор} \\ 0 \end{cases}, \quad p_{23} = \begin{cases} 1, P_{тек}^{i+m} < P_{пор} \\ 0 \end{cases},$$

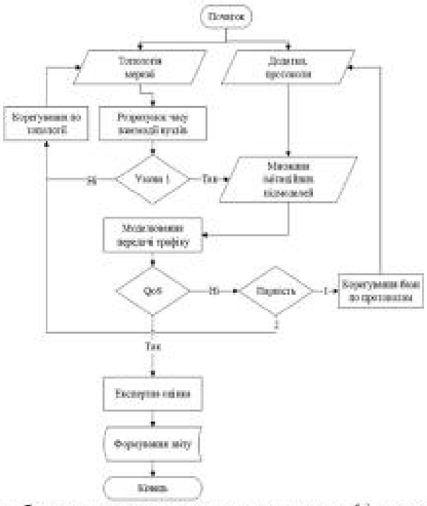
$$p_{24} = \begin{cases} 1, P_{тек}^{i+n} > P_{пор} \\ 0 \end{cases};$$

$$p_{33} = \begin{cases} 1, P_{тек}^{\Sigma i} < P_{пор} \\ 0 \end{cases};$$

$$p_{43} = \begin{cases} 1, P_{тек}^{i+n} < P_{доп} \\ 0 \end{cases}, \quad p_{44} = \begin{cases} 1, P_{тек}^{i+n} = P_{тек}^{i+m} = P_{доп} \\ 0 \end{cases}, \quad p_{45} = \begin{cases} 1, P_{пор} > P_{тек}^{i+m} \geq P_{доп} \\ 0 \end{cases};$$

$$p_{51} = \begin{cases} 1, P_{тек}^{i+m} > P_{доп} \\ 0 \end{cases}, \quad p_{53} = \begin{cases} 1, P_{тек}^{i+m} < P_{доп} \\ 0 \end{cases}, \quad p_{54} = \begin{cases} 1, P_{тек}^{i+m} > P_{пор} \\ 0 \end{cases};$$

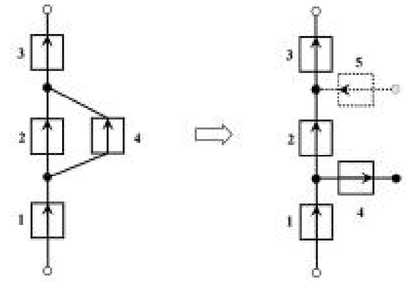
$$p_{55} = \begin{cases} 1, P_{тек}^{i+m} < P_{пор} \\ 0 \end{cases}.$$



Алгоритм роботи програмного комплексу за модифікованим квест-методом аналізу оверлейних мереж з невизначеною кількістю оверлеїв

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОБОТИ СИСТЕМИ АНАЛІЗУ ОВЕРЛЕЙНИХ МЕРЕЖ З НЕВИЗНАЧЕНОЮ КІЛЬКІСТЮ ОВЕРЛЕЇВ

Укрупнений алгоритм роботи програми



Заміна закритого контуру на відкритий в оверлейній мережі

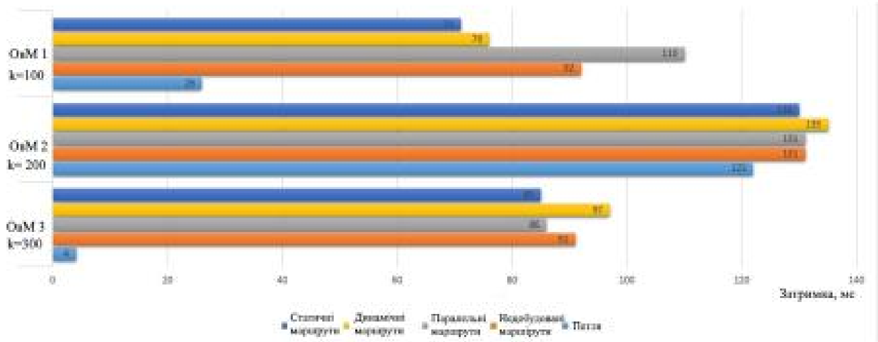
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОБОТИ СИСТЕМИ АНАЛІЗУ ОВЕРЛЕЙНИХ МЕРЕЖ З НЕВИЗНАЧЕНОЮ КІЛЬКІСТЮ ОВЕРЛЕЇВ

$$\bar{\mu}' = \begin{bmatrix} \mu'_{11} & \mu'_{12} & \cdots & \mu'_{1n} \\ \mu'_{21} & \mu'_{22} & \cdots & \mu'_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \mu'_{n1} & \mu'_{n2} & \cdots & \mu'_{nn} \end{bmatrix}$$

$$\begin{cases} \lambda'_1 = \mu'_{11}\rho'_1 + \mu'_{12}\rho'_2 + \ldots + \mu'_{1n}\rho'_n \\ \cdots \\ \lambda'_n = \mu'_{n1}\rho'_1 + \mu'_{n2}\rho'_2 + \ldots + \mu'_{nn}\rho'_n \end{cases}$$

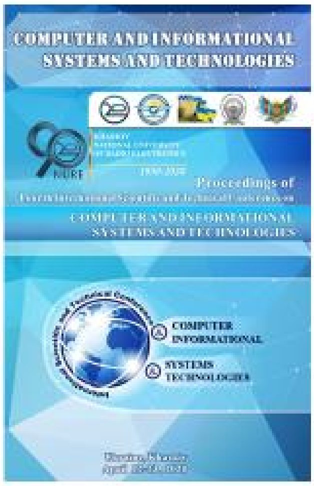
$$\begin{cases} \rho'_1 = A_{11}\rho_1 + A_{12}\rho_2 + \ldots + A_{1,n-k}\rho_{n-k} \\ \cdots \\ \rho'_1 = A_{n1}\rho_1 + A_{n2}\rho_2 + \ldots + A_{n,n-k}\rho_{n-k} \end{cases}$$

$$\bar{\rho}' = A\bar{\rho}$$



Вирахувані значення маршрутів прямування даних в оверлейних мережах з невизначеною кількістю оверлеїв

АПРОБАЦІЯ РЕЗУЛЬТАТІВ



Результати роботи апробовані на Міжнародній науково-технічній конференції "КОМП'ЮТЕРНІ ТА ІНФОРМАЦІЙНІ СИСТЕМИ І ТЕХНОЛОГІЇ" 22-23 квітня 2020 року.

Отримана публікація:

V. Tkachov, M. Hunko, M. Bondarenko, S. Artyomov Technology of Load Balancing in Anonymous Network Based on Proxy Nodes Cascade Platform / Міжнародна науково-технічна конференція "Комп'ютерні та інформаційні системи і технології". – Харків. – С. 82.

ВИСНОВКИ

В результаті реалізації модифікованого квест-методу для задачі аналізу оверлейних мереж з невизначеною кількістю оверлеїв була розроблена програмна система, яка дозволяє забезпечити можливості побудови будь-яких топологій схем і вибору моделі для кожної гілки оверлейної мережі з метою забезпечення анонімності її користувачів. При цьому результатом теоретичної роботи є розрахунок довжини черг і час затримки в гілках, а також ймовірності переповнення буферів і стаціонарні ймовірності станів для кожної гілки, обраного маршруту і всієї мережі в цілому, що впливає на ступінь анонімності мережі.

Виходячи з усього сказаного можна зробити наступні висновки: модифікований квест-метод аналізу оверлейних мереж з невизначеною кількістю оверлеїв є новим і дозволяє об'єднати практично всі існуючі послуги мереж з забезпеченням заданого рівня якості передачі даних; тензорний метод дає можливість досить просто формалізувати проектні процедури для мереж такого типу; програмна реалізація квест-методу дозволяє оцінювати необхідні показників якості при прийнятних обчислювальних витратах.

ДОДАТОК Б

Фрагменти коду програмного забезпечення для аналізу динамічної оверлейної мережі

```
//FILE: overlay/overlay.c
//
//Description: this file implements a ON process
//A ON process first connects to all the neighbors and then starts listen_to_neighbor threads each of
//which keeps receiving the incoming packets from a neighbor and forwarding the received packets to the
//SNP process. Then ON process waits for the connection from SNP process. After a SNP process is connected,
// the ON process keeps receiving sendpkt_arg_t structures from the SNP process and sending the received
//packets out to the overlay network.
//
//Date: April 28, 2020

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <signal.h>
#include <sys/utsname.h>
#include <assert.h>

#include "../common/constants.h"
#include "../common/pkt.h"
#include "overlay.h"
#include "../topology/topology.h"
#include "neighbortable.h"

//you should start the ON processes on all the overlay hosts within this period of time
#define OVERLAY_START_DELAY 25

/**************************************************************/

//declare global variables

/**************************************************************/

//declare the neighbor table as global variable

nbr_entry_t* nt;

//declare the TCP connection to SNP process as global variable

int network_conn;
```

```
/*********************************************************************/

//implementation overlay functions

/*********************************************************************/

// This thread opens a TCP port on CONNECTION_PORT and waits for the incoming
connection from all the neighbors that have a larger node ID than my nodeID,
// After all the incoming connections are established, this thread terminates
void* waitNbrs(void* arg) {
      // create TCP port on connection port
      int sd, connection;
      int ntoConnect;
      int myID, neighborNum, newnodeID;
      struct sockaddr_in myaddr, connectedaddr;
      socklen_t connaddrlen;
      connaddrlen = sizeof(connectedaddr);
      //Find number of neighbors with larger node ID
      myID = topology_getMyNodeID();
      neighborNum = topology_getNbrNum();
      ntoConnect = 0;
      for (int i = 0; i < neighborNum; i++){
            if (nt[i].nodeID > myID){
                  ntoConnect++;
            }
      }
      //Set up server address
      memset(&myaddr, 0, sizeof(myaddr));
      myaddr.sin_family = AF_INET;
      myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
      myaddr.sin_port = htons(CONNECTION_PORT);
      //Initialize socket, bind, and listen
      sd = socket(AF_INET, SOCK_STREAM, 0);
      if (sd < 0){
            printf("err: socket creation failed\n");
            return NULL;
      }
      if (bind(sd, (struct sockaddr *)&myaddr, sizeof(myaddr))<0){
            printf("err: bind failed\n");
            return NULL;
      }
      if (listen(sd, MAX_NODE_NUM) < 0){
            printf("err: listen failed\n");
            return NULL;
      }
      // Accept connections until all larger neighbors have connected
      printf("Connections to receive: %d\n", ntoConnect);
      while (ntoConnect > 0){
            connection   =   accept(sd,   (struct   sockaddr*)&connectedaddr,
&connaddrlen);
            if (connection == -1){
                  printf("err: connection failed\n");
                  continue;
            }
            newnodeID = topology_getNodeIDfromip(&connectedaddr.sin_addr);
            printf("Received conn from node = %d\n", newnodeID);
            nt_addconn(nt, newnodeID, connection);
            ntoConnect--;
      }
      return NULL;
}
```

```
// This function connects to all the neighbors that have a smaller node ID than my nodeID

// After all the outgoing connections are established, return 1, otherwise return -1

int connectNbrs() {

      int neighborNum, myID;

      int newconn;

      struct sockaddr_in servaddr;


      neighborNum = topology_getNbrNum();

      myID = topology_getMyNodeID();

      for (int i = 0; i < neighborNum; i++){

            if (nt[i].nodeID < myID){

                  //Set up server address

                  memset(&servaddr, 0, sizeof(servaddr));

                  servaddr.sin_port = htons(CONNECTION_PORT);

                  servaddr.sin_addr.s_addr = nt[i].nodeIP;

                  servaddr.sin_family = AF_INET;


                  // Set up socket and connect

                  newconn = socket(AF_INET, SOCK_STREAM, 0);

                  if (newconn < 0){

                        printf("err: can't create the socket\n");

                        return -1;

                  }


                  if (connect(newconn, (struct sockaddr*)&servaddr, sizeof(servaddr))<0){

                        printf("err: connection failed\n");

                        return -1;

                  }

                  printf("Connection est to %d\n", nt[i].nodeID);
```

```
                  //create the new connection in the table

                  nt_addconn(nt, nt[i].nodeID, newconn);

            }

      }

  return 1;

}
```

//Each listen_to_neighbor thread keeps receiving packets from a neighbor. It handles the received packets by forwarding the packets to the SNP process.

//all listen_to_neighbor threads are started after all the TCP connections to the neighbors are established

```
void* listen_to_neighbor(void* arg) {

      int index;

      int conn;

      snp_pkt_t pkt;

      // Cast arg back into an int

      index = *(int*) arg;

      //find the connection in that index

      conn = nt[index].conn;

      //Continually receive packets from neighbor and forward to SNP

      while (recvpkt(&pkt, conn) > 0){

            printf("Received packet from neighbor %d\n", nt[index].nodeID);

            if (forwardpktToSNP(&pkt, network_conn) > 0){

                  printf("Forwarded packet to SNP\n");

            }

      }

      close(conn);

      nt[index].conn = -1;

      pthread_exit(0);

}
```

/*This function opens a TCP port on OVERLAY_PORT, and waits for the

incoming connection from local SNP process. After the local SNP process is

connected, this function keeps getting sendpkt_arg_ts from SNP process, and

sends the packets to the next hop in the overlay network. If the next hop's

```
nodeID is BROADCAST_NODEID, the packet should be sent to all the neighboring
nodes.
*/
void waitNetwork() {
	int overlayserver;
	struct sockaddr_in serv_addr, snp_addr;
	socklen_t snp_addr_len;
	snp_addr_len = sizeof(snp_addr);
	snp_pkt_t pkt;
	int nextNode;
	int numNeighbors = topology_getNbrNum();
	//Create listening socket
	overlayserver = socket(AF_INET, SOCK_STREAM, 0);
	if (overlayserver < 0){
		printf("err: socket creation failed for ON\n");
		return;
	}

	//Set up server address

	memset(&serv_addr, 0, sizeof(serv_addr));
	serv_addr.sin_family = AF_INET;
	serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
	serv_addr.sin_port = htons(OVERLAY_PORT);
	//bind socket and listen for connection
	if (bind(overlayserver, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0){
		printf("err: bind failed for ON server\n");
		return;
	}
	if (listen(overlayserver, 1) < 0){
		printf("err: listen failed\n");
		return;
```

```
    }



    while (1){
        network_conn = accept(overlayserver, (struct sockaddr*)&snp_addr,
&snp_addr_len);
        if (network_conn == -1){
            printf("Connection could not be est with SNP\n");
            continue;
        }
        printf("SNP connection est\n");
        while (getpktToSend(&pkt, &nextNode, network_conn) > 0){
            printf("Received packet from SNP\n");


            // If next node is BROADCAST_NODEID send to all the
neighbors
            if (nextNode == BROADCAST_NODEID){
                for (int i = 0; i < numNeighbors; i++){
                    if (nt[i].conn != -1){
                        sendpkt(&pkt, nt[i].conn);
                        printf("Forwarded packet to %d\n",
nt[i].nodeID);
                    }
                }
            }

            // Otherwise find the next node in the table
            else {
                for (int i = 0; i < numNeighbors; i++){
                    if (nt[i].nodeID == nextNode){
                        if (sendpkt(&pkt, nt[i].conn) < 0){
                            printf("failed to forward
packet\n");
                        }
                        else{
```

```
                                        printf("forwarded packet\n");
                                    }
                                    break;
                                }
                            }
                        }
                    }
                    close(network_conn);
                    network_conn = -1;
            }
    }

    //this function stops the overlay
    //it closes all the connections and frees all the dynamically allocated memory
    //it is called when receiving a signal SIGINT
    void overlay_stop() {
            //Close all overlay connections and free neighbortable
            nt_destroy(nt);


            //Close SNP connection
            close(network_conn);
            network_conn = -1;

            exit(0);
    }


    int main() {


            //start overlay initialization
            printf("Overlay:  Node  %d  initializing...\n",topology_getMyNodeID());
```

```
        //create a neighbor table

        nt = nt_create();

        //initialize network_conn to -1, means no SNP process is connected yet

        network_conn = -1;


        //register a signal handler which is sued to terminate the process

        signal(SIGINT, overlay_stop);



        //print out all the neighbors

        int nbrNum = topology_getNbrNum();

        int i;

        for(i=0;i<nbrNum;i++) {

              printf("Overlay: neighbor %d:%d\n",i+1,nt[i].nodeID);

        }



        //start the waitNbrs thread to wait for incoming connections from
neighbors with larger node IDs
        pthread_t waitNbrs_thread;
        pthread_create(&waitNbrs_thread,NULL,waitNbrs,(void*)0);
        //wait for other nodes to start
        sleep(OVERLAY_START_DELAY);
        //connect to neighbors with smaller node IDs
        connectNbrs();
        //wait for waitNbrs thread to return
        pthread_join(waitNbrs_thread,NULL);
        //at this point, all connections to the neighbors are created
        //create threads listening to all the neighbors

        for(i=0;i<nbrNum;i++) {
              int* idx = (int*)malloc(sizeof(int));
              *idx = i;
              pthread_t nbr_listen_thread;
        pthread_create(&nbr_listen_thread,NULL,listen_to_neighbor,(void*)idx);
        }
        printf("Overlay: node initialized...\n");
        printf("Overlay: waiting for connection from SNP process...\n");
        //waiting for connection from  SNP process
        waitNetwork();
}
```

Приклад Б.1 – Структура файлу

Simple-Overlay-Network/overlay/overlay.c