CN-CA2

Mohammad Bahrami - 400243024

گزارش تمرین کامپیوتری دوم - Distance Vector Protocol Simulation :

دیتا استراکچر های مهم:

الف)Distance Table:

```
struct distance_table {
    int costs[4][4];
};
```

هر گره ساختار distance_table خود را دارد. آرایه هزینه ها فاصله این گره تا سایر گره های شبکه را ذخیره می کند. سطرها معمولاً نشان دهنده گره مبدا و ستون ها نشان دهنده گره های مقصد هستند.

ب)Packets :

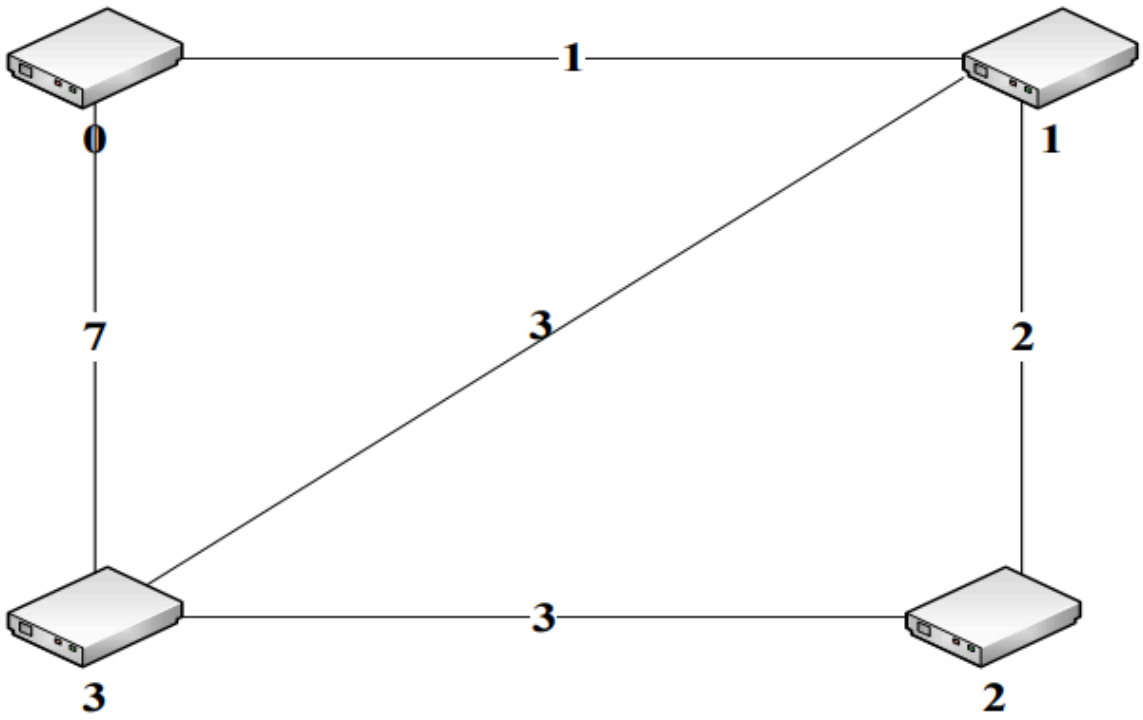
```
struct rtpkt {
    int sourceid;
    int destid;
    int mincost[4];
};
```

پکت ها برای تبادل اطلاعات مسیریابی بین گره ها استفاده می شوند. ساختار یک پکت در فایل هدر DV.h تعریف شده است.

ساختار `rtpkt`:

- `int sourceid`: شناسه گره ارسال‌کننده بسته.
- `int destid`: شناسه گره دریافت‌کننده بسته (باید همسایه مستقیم باشد).
- `int mincost[4]`: کمترین هزینه به گره‌های 0 تا 3.

توپولوژی شبکه:




```c
#include "DV.h"
#include "node0.h"

#define NODE_ID 0


// Link costs to directly connected neighbours used for initializing the distance table.
int connectcosts0[4] = { 0, 1, 999, 7 };

// Minimum costs to all nodes, initially set to same as connectcosts
int mincosts0[4] = { 0, 1, 999, 7 };

struct distance_table dt0;
```

```c
#define NODE_ID 1


// Link costs to directly connected neighbours used for initializing the distance table.
int connectcosts1[4] = { 1, 0, 2, 3 };

// Minimum costs to all nodes, initially set to same as connectcosts
int mincosts1[4] = { 1, 0, 2, 3 };

struct distance_table dt1;
```

```c
#define NODE_ID 2


// Link costs to directly connected neighbours used for initializing the distance table.
int connectcosts2[4] = { 999, 2, 0, 3 };

// Minimum costs to all nodes, initially set to same as connectcosts
int mincosts2[4] = { 999, 2, 0, 3 };

struct distance_table dt2;
```

```c
#define NODE_ID 3


// Link costs to directly connected neighbours used for initializing the distance table.
int connectcosts3[4] = { 7, 3, 3, 0 };

// Minimum costs to all nodes, initially set to same as connectcosts
int mincosts3[4] = { 7, 3, 3, 0 };

struct distance_table dt3;
```

در هر نود همین ساختار را داریم و در ابتدا ID هر نود را ست کرده و توپولوژی شبکه را به عنوان یک آرایه 4 تایی به آن داده و در انتها جدول فاصله ها را تعریف میکنیم.

فانکشن های مهم:

1-rtinit:

```
/* students to write the following two routines, and maybe some others */
void rtinit0() {
    // Initialize distance table
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (i == j) {
                dt0.costs[i][j] = connectcosts0[i];
            }
            else {
                dt0.costs[i][j] = 999;
            }
        }
    }

    sendcosts0();
    printdt0();
}
```

مقداردهی اولیه: حلقه های تو در تو جدول فاصله را مقداردهی اولیه می کنند. برای هر گره i، بررسی می کند که آیا i همان j است یا خیر. اگر آنها یکسان باشند، هزینه را از گره 0 تا i به عنوان هزینه پیوند مستقیم تعیین می کند. در غیر این صورت، آن را به یک مقدار بزرگ (999) تنظیم می کند.

هزینه های ارسال: تابع sendcosts برای انتشار بردار فاصله اولیه به گره های همسایه فراخوانی می شود.

چاپ: حالت اولیه جدول فاصله برای کمک به تجسم نقطه شروع شبیه سازی چاپ می شود.

2-rtupdate:

```
void rtupdate0(struct rtpkt *rcvdpkt) {
    printf("----------\n");
    printf("rtupdate0 srcid: %i\n", rcvdpkt->sourceid);
    printf("rtupdate0 destid: %i\n", rcvdpkt->destid);
    printf("rtupdate0 mincosts: ");
    for (int i = 0; i < 4; i++) {
        printf("%i ", rcvdpkt->mincost[i]);
    }
    printf("\n");

    // Steps to take:
    // - Check srcid (use as index for 2d distance table)
    // - Iterate through mincosts array (index is destination node)
    // - If connectcosts0[sourceid] + rcvdpkt->mincosts[i] < connectcosts0[sourceid] + dt0.costs[sourceid][i] then update dt0.costs[s
    // - If update made in last step, then resend dt0 to directly connected nodes

    bool table_updated = false;

    printf("----------\n");

    for (int i = 0; i < 4; i++) {
        // Don't update the distance table with another node's link cost to itself (which will always be 0)
        if (i == rcvdpkt->sourceid) {
            printf("i %i matches sourceid %i. Skipping.\n", i, rcvdpkt->sourceid);
            continue;
        }
```

```
    // Don't update the distance table with another node's link cost back to the current node (which we already have from mincost
    if (i == NODE_ID) {
        printf("i %i matches NODE_ID %i. Skipping.\n", i, NODE_ID);
        continue;
    }

    // Ignore link costs of 999, which mean a node has not yet established a path to another node.
    if (rcvdpkt->mincost[i] == 999) {
        printf("node %i has not yet established a path to node %i. Skipping.\n", rcvdpkt->sourceid, i);
        continue;
    }

    printf("dt0.costs[i][srcid]: %i\n", dt0.costs[i][rcvdpkt->sourceid]);
    printf("connectcosts0[srcid]: %i\n", connectcosts0[rcvdpkt->sourceid]);
    printf("connectcost0[srcid] + rcvdpkt->mincost[i]: %i\n", connectcosts0[rcvdpkt->sourceid] + rcvdpkt->mincost[i]);
    if (dt0.costs[i][rcvdpkt->sourceid] == 0 || (connectcosts0[rcvdpkt->sourceid] + rcvdpkt->mincost[i]) < dt0.costs[i][rcvdpkt->
        dt0.costs[i][rcvdpkt->sourceid] = connectcosts0[rcvdpkt->sourceid] + rcvdpkt->mincost[i];
        table_updated = true;
    }
}

printf("----------\n");

if (table_updated == true) {
    printf("dt0 was updated. New table below.\n\n");
    printdt0();

    findmincosts0();
```

```
    printf("----------\n");

    if (table_updated == true) {
        printf("dt0 was updated. New table below.\n\n");
        printdt0();

        findmincosts0();

        printf("dt0update mincosts0: ");
        for (int i = 0; i < 4; i++) {
            printf("%i ", mincosts0[i]);
        }
        printf("\n");

        printf("dt0update sending out new min costs.\n");
        sendcosts0();
    }
}
```

تابع rtupdate0() مسئول رسیدگی به دریافت بسته های برداری فاصله از گره های همسایه و به روز رسانی جدول فاصله برای گره 0 است. این تابع در پروتکل مسیریابی بردار فاصله مرکزی است زیرا اطلاعات دریافتی را پردازش می کند و جدول مسیریابی را به روز می کند تا توپولوژی شبکه فعلی را منعکس کند.این ساختار تضمین می کند که هر گره در شبکه به طور مداوم جدول فاصله خود را بر اساس آخرین اطلاعات همسایگان خود به روز می کند و در نهایت به اطلاعات مسیریابی کوتاه ترین مسیر همگرا می شود.

بقیه توابع آپدیت نیز مراحل یکسانی را دنبال می‌کنند: دریافت بسته، استخراج شناسه منبع و بردار فاصله، به‌روزرسانی جدول فاصله در صورت یافتن مسیر کوتاه‌تر، و ارسال بردارهای فاصله جدید در صورت به‌روزرسانی.

3-findmincost:

```
void findmincosts0() {
    // Update mincosts if distance table was updated.
    for (int i = 0; i < 4; i++) {
        // Ignore link cost to ourself
        if (i == NODE_ID) {
            continue;
        }

        for (int j = 0; j < 4; j++) {
            // Ignore routes to another node via ourself, which we already have from connectcosts[]
            if (j == NODE_ID) {
                continue;
            }

            // Ignore uninitialized link costs
            if (dt0.costs[i][j] == 0) {
                continue;
            }

            // Update mincost[i] with lowest cost for node i from each of the options of dt0.costs[i][j]
            if (dt0.costs[i][j] < mincosts0[i]) {
                mincosts0[i] = dt0.costs[i][j];
            }
        }
    }
}
```

این تابع برای به‌روزرسانی آرایه `mincosts0` بر اساس جدول مسافت‌ها (`distance table`) استفاده می‌شود، در صورتی که جدول مسافت‌ها به‌روزرسانی شده باشد.
اگر شناسه گره همان شناسه گره جاری (`NODE_ID`) باشد، ادامه می‌دهد و هزینه لینک به خود را نادیده می‌گیرد.

4-sendcosts:

```c
void sendcosts0() {
    struct rtpkt cost_pkt;
    cost_pkt.sourceid = NODE_ID;
    cost_pkt.destid = 0;
    memcpy(&cost_pkt.mincost, mincosts0, 4 * sizeof(int));

    struct rtpkt *cost_pkt_ptr = &cost_pkt;

    for (int i = 0; i < 4; i++) {
        // ignore self and non directly connected nodes (999)
        if (i == NODE_ID || connectcosts0[i] == 999) {
            continue;
        }

        cost_pkt_ptr->destid = i;
        tolayer2(cost_pkt);
    }
}
```

تابع sendcosts مسئول ارسال بردار فاصله به روز شده از گره 0 به تمام همسایگان متصل مستقیم آن است. این یک گام مهم در پروتکل مسیریابی بردار فاصله است زیرا نمای فعلی شبکه را از گره 0 به همسایگانش منتشر می کند و به آنها امکان می دهد جداول فاصله خود را به روز کنند.

5-tolayer2:

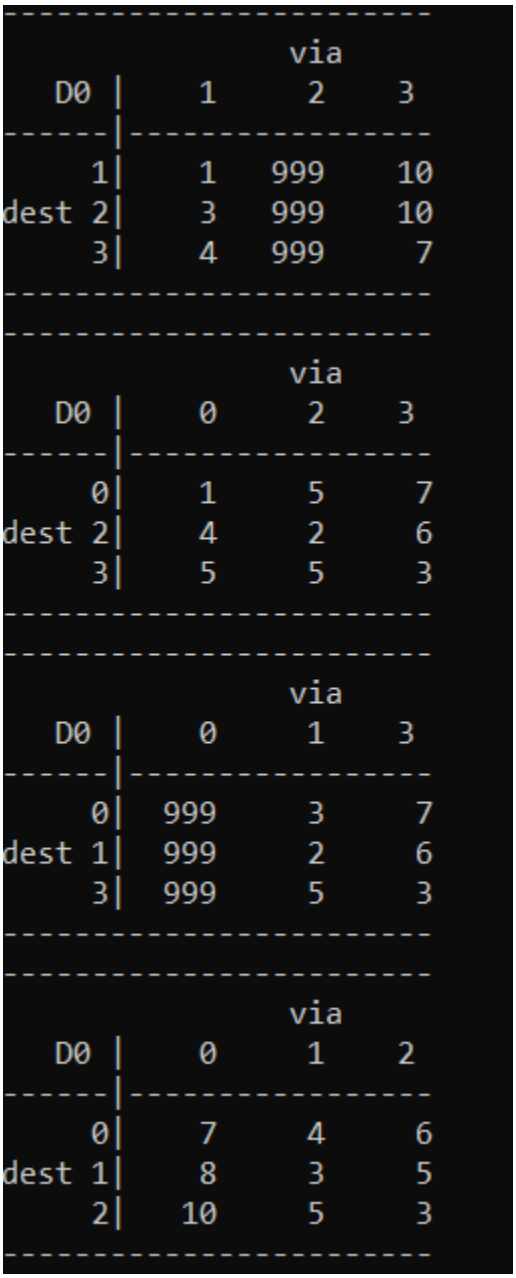
```
void tolayer2(struct rtpkt packet) {
    struct rtpkt *mypktptr;
    struct event *evptr, *q;
    float jimsrand(),lastime;
    int i;

    int connectcosts[4][4];

    /* initialize by hand since not all compilers allow array initilization */
    connectcosts[0][0]=0;  connectcosts[0][1]=1;  connectcosts[0][2]=999;
    connectcosts[0][3]=7;
    connectcosts[1][0]=1;  connectcosts[1][1]=0;  connectcosts[1][2]=2;
    connectcosts[1][3]=3;
    connectcosts[2][0]=999;  connectcosts[2][1]=2;  connectcosts[2][2]=0;
    connectcosts[2][3]=3;
    connectcosts[3][0]=7;  connectcosts[3][1]=3;  connectcosts[3][2]=3;
    connectcosts[3][3]=0;

    /* be nice: check if source and destination id's are reasonable */
```

```
  /* be nice: check if source and destination id's are reasonable */
  if (packet.sourceid<0 || packet.sourceid >3) {
      printf("WARNING: illegal source id in your packet, ignoring packet!\n");
      return;
  }

  if (packet.destid<0 || packet.destid >3) {
      printf("WARNING: illegal dest id in your packet, ignoring packet!\n");
      return;
  }

  if (packet.sourceid == packet.destid)  {
      printf("WARNING: source and destination id's the same, ignoring packet!\n");
      return;
  }

  if (connectcosts[packet.sourceid][packet.destid] == 999)  {
      printf("WARNING: source and destination not connected, ignoring packet!\n");
      return;
  }

  /* make a copy of the packet student just gave me since he/she may decide */
  /* to do something with the packet after we return back to him/her */
  mypktptr = (struct rtpkt *) malloc(sizeof(struct rtpkt));
  mypktptr->sourceid = packet.sourceid;
  mypktptr->destid = packet.destid;
```

```
    mypktptr->mincost[i] = packet.mincost[i];
}

if (TRACE > 2) {
    printf("    TOLAYER2: source: %d, dest: %d\n              costs:",
    mypktptr->sourceid, mypktptr->destid);
    for (i=0; i<4; i++) {
        printf("%d  ",mypktptr->mincost[i]);
    }
    printf("\n");
}

/* create future event for arrival of packet at the other side */
evptr = (struct event *)malloc(sizeof(struct event));
evptr->evtype =  FROM_LAYER2;     /* packet will pop out from layer3 */
evptr->eventity = packet.destid;  /* event occurs at other entity */
evptr->rtpktptr = mypktptr;       /* save ptr to my copy of packet */

/* finally, compute the arrival time of packet at the other end.
medium can not reorder, so make sure packet arrives between 1 and 10
time units after the latest arrival time of packets
currently in the medium on their way to the destination */
lastime = clocktime;

for (q=evlist; q!=NULL ; q = q->next) {
    if ( (q->evtype==FROM_LAYER2  && q->eventity==evptr->eventity) )
    lastime = q->evtime;
}

evptr->evtime = lastime + 2.*jimsrand();

if (TRACE > 2) {
    printf("    TOLAYER2: scheduling arrival on other side\n");
}

insertevent(evptr);
```

تابع tolayer2 فرآیند ارسال یک بسته از یک گره به گره دیگر در یک شبکه را شبیه سازی می کند. این نشان دهنده لایه پایینی در پشته شبکه است که در آن انتقال واقعی بسته ها اتفاق می افتد. در یک پیاده سازی واقعی شبکه، این تابع لجستیک تحویل یک بسته به گره مقصد را انجام می دهد، اما در زمینه شبیه سازی، این فرآیند را ساده و انتزاعی می کند.

خروجی نهایی:



```
----------------------
            via
  D0 |   1    2    3
-----|----------------
    1|   1  999   10
dest 2|   3  999   10
    3|   4  999    7
----------------------
----------------------
            via
  D0 |   0    2    3
-----|----------------
    0|   1    5    7
dest 2|   4    2    6
    3|   5    5    3
----------------------
----------------------
            via
  D0 |   0    1    3
-----|----------------
    0| 999    3    7
dest 1| 999    2    6
    3| 999    5    3
----------------------
----------------------
            via
  D0 |   0    1    2
-----|----------------
    0|   7    4    6
dest 1|   8    3    5
    2|  10    5    3
----------------------
```

جدول هر گره هزینه های محاسبه شده برای رسیدن به هر گره دیگر را از طریق هر گره میانی ممکن نشان می دهد.

حداقل هزینه برای هر مقصد با مقایسه مقادیر در ستون ها مشخص می شود.

این جداول وضعیت نهایی را پس از همگرا شدن الگوریتم مسیریابی بردار فاصله منعکس می کنند، به این معنی که همه گره ها بردارهای فاصله خود را به اشتراک گذاشته اند و کوتاه ترین مسیرها ایجاد شده است.

این تجزیه و تحلیل کمک می کند تا اطمینان حاصل شود که هر گره درک دقیق و به روزی از توپولوژی شبکه دارد و می تواند بسته ها را به طور موثر با استفاده از کوتاه ترین مسیرهای شناسایی شده در این جداول فاصله، مسیریابی کند.