PROTOCOALE DE COMUNICATIE: Tema #2
Simularea unui Protocol de Routare

Termen de predare: 16 APRILIE 2015

Titulari curs: *Valentin CRISTEA, Gavril GODZA, Florin POP*

Responsabili Tema: **Florin POP, Catalin VASILE, Claudia IFRIM**

Obiectivele temei

Obiectivul temei 2 de casa este simularea unei retele de routere prin implementarea unui program (in C/C++). Mai exact, routerele vor trebui sa fie capabile sa ruleze un algoritm de rutare SPF (shortest path first) minimalist, sa se adapteze schimbarilor din retea in cadrul aceluiasi algoritm si sa ruteze o serie de mesaje date ca input. Schimbarile care pot interveni in cadrul retelei pot fi de doua tipuri:

1. adaugarea unei noi legaturi in retea;
2. stergerea unei legaturi existente in retea.

Suportul temei

Tema va avea ca suport urmatoarele fisiere:

- `main.cpp`: prezinta etapele cronologice ale simularii. Orice functie care nu contine cuvantul `api` poate fi modificata (se poate modifica antetul si se pot trimite si alti parametri).
  Atentie! NU schimbati ordinea/locatia functiilor. Orice schimbare va conduce la un output gresit al simulatorului.
- `sim.h/.cpp`: contin functiile pe care trebuie sa le implementati in rezolvarea temei, le puteti modifica antetul, dar atentie, ele sunt apelate in `main.cpp`. Functii care trebuiesc implementate sunt:
  - `init_sim()`: se vor implementa toate actiunile care tin de initializarea simulatorului. Tot aici se va face si initializarea protocolului de rutare (floodarea initiala cu mesaje).
  - `trigger_events()`: va verifica lista de schimbari/evenimente; doar cele ale carui timp corespund cu timpul simulatorului vor fi procesate ("triggered").
  - `process_messages()`: va verifica lista de mesaje care trebuie pornite in retea si va fi instintat nodul aferent de la care porneste mesajul; se vor redistribui mesajele de protocol si cele rutabile. *Atentie!* Nu conteaza ordinea in care realizati aceste 2 actiuni.
  - `update_routing_table()`: fiecare ruter in parte este anuntat ca a venit timpul sa-si recalculeze tabela de rutare pe baza datelor acumulate in structura de date care reprezinta topologia.

Important!!! O recomandare ar fi sa aveti cel putin 4 clase/fisiere: una din ele este cea existenta si anume `sim.cpp` care va gestiona ruterele o clasa pentru rutere o clasa pentru tabela de rutare si o clasa pentru topologia pe care vor actiona ruterele.

Important!!! Fiecare router are propria lui topologie si propria lui tabela de rutare. Nu este o recomandare, este o cerinta obligatorie. Nerespectarea acestei constrangeri va duce la anularea temei.

Etapele protocolului de rutare

Fiecare ruter isi va mentine o structura de date care simbolizeaza topologia retelei. In schimbul de date cu celelalte rutere el are voie sa **porneasca** mesaje care contin DOAR vecinii direct conectati si costurile aferente (pot contine si alte date, dar in niciun caz o topologie intreaga, sau tabela de rutare) pentru a putea ajunge la ei; si sa faca **forward** la mesajele primite de la celelalte rutere.

1. La **initializarea simularii**, dupa ce fiecare ruter si-a primit DOAR vecinii direct conectati, ruterele vor trimite cate un mesaj la fiecare vecin direct conectat care sa contina vecinii lui si costurile pentru a ajunge la ei (`init_sim()`);
2. In continuare se va face forwarding la **mesajele** pe care le-au initiat celelalte noduri din retea (`process_messages()`);
3. In momentul in care se apeleaza `update_routing_table()` fiecare router va fi anuntat ca e timpul sa-si calculeze tabela de rutare pe baza a ceea ce a acumulat in structura de date care reprezinta topologia.

Etapele de tratare ale unui eveniment

In cazul unui eveniment care a fost `triggered` se vor lua urmatoarele actiuni:

1. Cele 2 rutere implicate vor fi anuntate de schimbare si vor incepe se floodeze reteaua cu mesaje care contin vecinii lor direct conectati si costurile aferente (din nou, nu aveti voie sa trimiteti o topologie intreaga sau tabela de rutare), `trigger_events()`.
2. Celelalte rutere vor trebui sa-si updateze topologia pe baza mesajelor primite si sa le fac forward mai departe la vecinii lor direct conectati, `process_messages()`.
3. Cand se apeleaza iar `update_routing_table()`, ruterele vor trebui sa-si recalculeze tabela de rutare pe baza topologiei proprii.

Floodare la infinit cu mesaje de gestionare

Daca nu se implementeaza un mecanism in plus, mesajele folosite pentru mentinerea algoritmului de rutare vor ajunge sa floodeze reteaua la infinit. Pentru a opri acest lucru, un algoritm SPF obisnuit obliga routerele sa-si semneze pachetele cu o versiune a datelor continute. Aceasta versiune este modificata doar de routerul caruia ii apartine informatia. Ruterele care primesc mesajele doar le inspecteaza si le face forward. Ruterul caruia ii apartine mesajul va incrementa versiunea la aparitia unui nou eveniment (aparitia sau disparitia unei legaturi).

Un ruter care inspecteaza mesajul/informatia primit/a se va uita la versiunea din mesaj si o va compara cu versiunea stocata in baza lui de date. Daca versiunea este aceeasi sau mai mica inseamna ca informatia nu ii este de folos (este prea veche sau o stie deja) => nici nu-si va updata datele pe baza mesajului si nici nu-i va face forward.

Rutare de pachete

Vor fi doua actiuni majore la rutarea de pachet:

1. Simulatorul va inspecta coada de mesaje la fiecare nou ciclu de timp si se va uita daca va gasi un mesaj care trebuie pornit la timpul curent. Daca gaseste un astfel de mesaj, ruterul in cauza este anuntat sa inceapa acest mesaj.
2. Celelalte rutere vor ruta mai departe mesajul fara sa-l modifice efectiv, `process_messages()`.

API

Pentru rezolvare temei de casa punem la dispozitie un API. Pentru a beneficia de acesta, in fiecare sursa trebuie sa includeti fisierul `api.h`.

Detaliile specifice acestui API sunt:

- `get_time()` - va intoarce "ceasul" simulatorului
- `endpoint` - pentru fiecare nod/router `i` din retea va exista o variabila `endpoint[i]` specifica lui.

Exemplu de trimitere a unui mesaj de gestionare a algoritmului de rutare:

```
endpoint[i].send_msg(&endpoint[j], message, msg_size, filename);
```

unde:

- `i` - id-ul nodului care trimite mesajul;
- `j` - id-ul nodului catre care se trimite mesajul;
- `message` - pointer catre mesajul care se trimite;
- `msg_size` - lungimea mesajului;
- `filename` - un string care poate fi folosit pentru generarea unei poze cu graful simulatorului, cu evidentierea link-ului folosit. Poate fi NULL, daca vreti sa nu se genereze un astfel de fisier.

Exemplu de trimitere a unui mesaj rutabil (care face parte din traficul normal al retelei):

```
endpoint[i].route_message(&endpoint[j], dst, tag, message, filename);
```

unde:

- `i` - id-ul nodului care trimite mesajul;
- `j` - id-ul nodului catre care se trimite mesajul (next hop-ul);
- `dst` - id-ul nodului final, la care trebuie sa ajunga mesajul;
- `tag` - un tag dat mesajului (citit din fisierul de input al simulatorului);
- `message` - un string (=> un sir de caractere terminat in `\0`) care reprezinta mesajul care va fi trimis;
- `filename` - un string care poate fi folosit pentru generarea unei poze cu graful simulatorului, cu evidentierea intregului circuit al mesajului pana in acel moment; Poate fi NULL, daca vreti sa nu se genereze un astfel de fisier. Desenarea circuitului intreg se bazeaza pe tag. Daca acesta nu e folosit corect tot timpul nu se poate preciza corectitudinea outputului.

Exemplu de primire a unui mesaj rutabil:

```
valid = endpoint[i].recv_message(&src, &dst, &tag, message);
```

unde:

- `i` - id-ul routerului care face recv pe mesaj;
- `valid` - este valoarea intoarsa de functie. Acesta va comunica daca s-a intors un mesajul valid => daca mai sunt mesaje de primit la timpul curent;
- `src` previous hop;
- `dst` - id-ul ruterului final, care trebuie sa primeasca mesajul.
- `message` - un string (=> un sir de caractere terminat in `\0`) care contine mesajul in sine.

Exemplu de primire a unui mesaj folosit la gestionarea algoritmului de rutare:

```
msg_size = endpoint[i].recv_protocol_message(message);
```

unde:

- `msg_size` - dimensiunea mesajului primit. Daca nu s-a primit nimic (si nu mai exista nimic de primit la timpul curent) acesta va fi `-1`;
- `i` - id-ul routerului care face recv pe mesaj; message = un pointer spre un string (=> un sir de caractere terminat in `\0`) in care se pot primi datele. Pentru alte date necesare gestionarii algoritmului de rutare va trebui sa va codificati voi mesajul in acest sens.

Compilare si rulare

Pentru compilarea temei aveti un `Makefile` functional.

```
make        % va compila sursele curente;
make clean  % va sterge executabilul si fisierele obiect aferente.
```

Pentru a adauga si fisierele voastre la compilare, completati in dreapta regulii simulation cu sursele voastre `*.cpp *.c`.

Rulare:

```
./simulation <topology file> <messages file> <events file>
```

Format fisier de topologie:

```
n       # numarul de noduri/rutere
0 2 1   # r1 cost r2
...     # o lista de adiacenta
8 3 5   # r1 cost r2
```

Adicentele trebuie facute in ambele directii. O linie `0 2 1` nu va asigura ca exista inca o linie de forma `1 2 0`.

Format fisier mesaje input:

```
n                                   # numarul de mesaje
0 4 15 10 Succes la tema de casa.   # <src> <dst> <time> <tag> <msg>
```

Mesajul in sine este cel dupa primele 4 valori numerice, si include si `\n`.

Format fisier evenimente:

```
m             # nr de unitati de timp care vor fi simulate de program
n             # numarul de evenimente
0 1 0 3 10    # <r1> <r2> <type> <cost> <time>
```

`type` poate avea 2 valori:

- 0 = legatura noua
- 1 = legatura intrerupta

Costul in cazul evenimentului de intrerupere de legatura este o valoare dummy (nefolosita).

Tema va fi corectata pe baza rezultatelor afisate in fisierul `results.out`. Acesta contine logari ale pachetelor rutate. Exemplu de logare:

```
[16] R[10]: 2 -> 4 =  Succes la tema de casa.
```

unde

- 16 = timpul la care s-a transmis mesajul
- 10 = tagul mesajului
- 2 = previous hop
- 4 = next hop

Restrictii

- Fiecare ruter primeste de la simulator date doar despre vecinii sai direct conectati.
- Mesajele sunt transmise strict pe baza informatiilor locale ruterului in cauza. Informatii locale sunt datele primite de la simulator si date primite prin mesaje de la vecini.

- Un mesaj poate circula intre 2 rutere doar daca exista un link intre ele. In momentul in care se termina simularea nu mai trebuie sa existe mesaje in tranzitie. Testele vor fi facute de asa natura incat sa asigure un timp suficient (Δt destul de mare, cu cel putin 5 unitati peste timpul optim de convergenta) pentru a trimite toate mesajele, inclusiv pe cele de gestionare a protocolului.
- Se va considera la fel de incorect daca studentul va arunca intentionat mesajele inainte de terminarea simularii. Tema are ca scop simularea unei retele de rutere, si NU doar trecerea unui set de teste.
- Nerespectarea acestor prevederi va duce la anularea punctajului pe tema, indiferent de punctajul afisat de checker.

Alte mentionari

- Intre initializare, mesaje si oricare eveniment vi se asigura un timp suficient a.i. reteaua sa conveaarga/trimita toate mesajele. In schimb vor exista teste care vor transmite un set de mesaje simultan. Nu conteaza ordinea in care le tratati.
- Output-ul folosit la corectarea temei (continutul fisierului `results.out` generat la sfarsitul rularii unei simulari) va fi sortat cu comanda sort inainte de a fi comparat cu un output de referinta, astfel ca ordinea in care s-au transmis mesajele rutabile la un timp `t` nu va conta. API-ul foloseste biblioteca `graphviz` pentru a genera imaginile cu traseul mesajelor. Pe distributii Debian-based puteti instala biblioteca cu urmatoarea comanda:
  `sudo apt-get install libgraphviz-dev`
- Biblioteca in sine contine bug-uri si memory-leaks, motiv pentru care resursele folosite pentru generarea imaginilor nu sunt eliberate.
- Programe precum valgrind vor vedea doar partea de memory-leaks, lasandu-va in continuare posibilitatea diagnosticarii problemelor de acces invalid la diverse zone de memorie.
- Pentru mesajele de gestionare a protocolului, API-ul mai mentine un log in fisierul `debug.out`. O intrare este de forma:
  `[3] D: 5 -> 1`
  unde:
  - 3 - timpul la care se transmite mesajul
  - 5 - sursa mesajului
  - 1 - destinatarul mesajului