Simularea unui protocol de dirijare cu starea legaturilor

Data limita pentru trimiterea temei: **Miercuri 17 aprilie 2013, ora 23:55**

Responsabil tema: Eliana-Dina Tirsa

Ultima modificare a enuntului: 9 aprilie (tratarea evenimentului adaugare de link – sectiunile colorate)

Ultima modificare a log_rutare (din exemplu): 9 aprilie (liniile 269 si 317)

Enuntul temei

Implementati (in C/C++) un protocol de dirijare cu starea legaturilor conform specificatiilor, intr-un simulator cu pasi de timp, urmand un schelet de cod dat (vezi resurse tema 2). Simulatorul are mai multe procese – unul pentru o entitate centrala (in continuare numita si simulator central sau proces central) si 10 pentru procese fiu ale simulatorului central, pentru rutere. Simulatorul central este complet implementat. De asemenea, caile de comunicatie sunt deja create. Voi aveti de implementat comportamentul ruterelor, conform listei de evenimente si mesajelor primite in retea.

Ipoteze:

- Un ruter va fi simulat printr-un proces. Pe langa programul central, aveti deja create 10 procese. Din acestea, doar cel cu id-ul 0 este initial nod in topologie. Celelalte vor adera la topologie conform unor evenimente primite in fisierul de input de programul central. O topologie poate avea maxim 10 rutere.
- Se considera ca legaturile dintre 2 rutere sunt punct la punct. (in implementare, trimiterea de mesaje intre rutere va fi mijlocita de catre simulatorul central). Deci practic toate mesajele trimise de rutere vor pleca pe aceeasi legatura(out), iar procesul central le va trimite ruterului destinatie(daca e cazul), conform campului next_hop din pachet.
- “Inundarea” cu mesaje se va face “hop-by-hop”, se presupune ca nu exista un mecanism de tip broadcast
- Link-urile au costuri, dar in simulare un mesaj al protocolului link state parcurge un link intr-un pas de timp (adica in pasul t este trimis iar in t+1 (nu mai devreme sau mai tarziu) se considera receptionat si este procesat)
- Mediul de comunicatie se presupune fiabil (nu avem nevoie de confirmari, nu se pierd mesaje pe link – dar se poate face discard de pachete la nivel de ruter, in caz de copii sau mesaje vechi)
- Un router intrat in retea nu se va considera oprit niciodata (pana la finalul simularii).
- Vi se garanteaza ca disparitia unui link sau adaugarea unui nou ruter nu vor fractiona reteaua in doua componente conexe diferite.
- Evenimentele vor avea o astfel de succesiune incat intre aparitia, suprimarea (eventual re-aparitia) unui link vor trece suficienti pasi de timp, pentru incat capetele linkului sa vada mereu aceeasi stare (up/down) pentru respectivul link.

Simulatorul va trata urmatoarele evenimente:

- adaugare ruter (eveniment tip 1)
- adaugare link intre 2 rutere (eveniment tip 2)
- suprimare link intre 2 rutere (eveniment tip 3)
- dirijare pachet (eveniment tip 4)

Se considera 4 tipuri de mesaje pentru protocolul link state si inca 6 tipuri de mesaje de control, pentru comunicarea intre procesul central si procesele care reprezinta ruterele:

Mesaje pentru protocolul link state (procesarea lor trebuie implementata de voi):

- LSA (link state advertisments) (tip 1)
- Database Request (tip 2)
- Database Reply (tip 3)
- Pachet de date (tip 4)

Mesaje de control: (R – ruter, C – proces central ; deja implementate)

- R->C: Terminare procesare mesaje la pasul curent (tip 5)
- C->R: Incepere procesare mesaje vechi din coada (tip 6)
- C->R: Anuntarea unui eveniment la ruterele direct implicate (tip 7)
- C->R: Cerere tabela de rutare de la ruterele din retea – la finalul fiecarui pas (tip 8)
- C->R: Terminare simulare (tip 9)
- R->C: Trimitere tabela de rutare la simulator central, pentru logare (tip 10)

Structura mesajului si tipuri de pachete

Aveti definita o structura de tip mesaj, care contine informatii necesare mesajelor de control, dar are si un camp in care va puteti configura voi ce alte campuri vreti (ca la tema 1). **Atentie, nu aveti voie sa modificati structura msg, definita deja in resursele temei.**

```
typedef struct {
        int type;    //tipul mesajului (1..10)
        int creator;   //id-ul ruterului care a creat mesajul
        int seq_no;  //numar de secventa; id unic al mesajului in cadrul ruterului creator
        int sender;   //indice ruter cel care trimite/forward-eaza mesajul
        int next_hop;  //indice nod unde este trimis mesajul → atentie, trebuie sa aveti NEAPARAT link
        int time;      //pasul de timp la care a fost trimis mesajul
        int add;  //e TRUE daca mesajul e de tip eveniment de adaugare link
        int len;   //la latitudinea voastra
        char payload[1400];  //la latitudinea voastra
} msg;
```

In cadrul campului payload puteti codifica orice informatii sau alte campuri de care credeti ca aveti nevoie. De exemplu (lista nu este exhaustiva, dar exista restrictii):

- lista_vecini+costuri (ai creator)
- destinatie (camp folosit doar la tipul 4)
- cost (folosit doar la tipul 4; dupa fiecare trimitere se aduna costul linkului; initial e 0)
- timp_creare (pasul de timp in care s-a construit pachetul)

(numele campurilor este orientativ, nu obligatoriu - in orice caz, folositi ceva sugestiv)

Un mesaj este unic determinat de (**creator, seq_no**). Avand doua mesaje cu acelasi creator si seq_no diferite, este mai nou cel care are seq_no mai mare. Un mesaj se considera copia altui mesaj daca au creator si numar de secventa egale. Nu putem stabili care dintre doua mesaje cu creator diferit este mai nou, uitandu-ne doar la numerele lor de secventa. De aceea folosim **time (timp_creare)**.

Fiecare ruter are un contor propriu pentru mesajele create de el. Fiecare nou mesaj creat “from scatch” (**de tipul 1,2,4**) primeste ca numar de secventa valoarea curenta a contorului, dupa care contorul este incrementat cu o unitate. (initial toate contoarele sunt 0). Numarul de secventa nu se modifica la forwardarea mesajului (**de tip 1 sau 4**).

Mesajele de **tip 3** (trimise ca raspuns la mesaje de tip 2) sunt create pe baza unor mesaje de tip 1 (vezi mai jos); mesajele de tip 3 vor lasa nemodificat numarul de secventa si campul creator;

Campurile creator si lista_vecini+costuri au semnificatii usor diferite in functie de tipul pachetului:

La tipul 1(LSA):

- creator este cel care creeaza pachetul; seq_no e setat doar la creare initiala, nu si la forwardare
- lista_vecini este lista cu toti vecinii creatorului si costuri (la momentul crearii pachetului)

La tipul 2:

- creator este creatorul mesajului (dupa caz: id-ul ruterului nou intrat, id-ul unui capat al unui nou link)
- lista_vecini contine doar legatura (creator ↔ vecin) si costul aferent

La tipul 3:

- creator este **ruterul despre care se ofera informatii noului intrat (sau celuilalt capat al unui nou link)**
- seq_no este numarul de secventa al informatiei despre creator in LSADatabase a senderului
- lista_vecini este lista cu toti vecinii ruterului creator si costuri (asa cum reiese din LSADatabase a senderului)
- Database reply(mesaj de tip 3) nu este un singur pachet, ci o succesiune de pachete: toate pachetele din LSADatabase a transmitatorului, dar cu tipul 3 in loc de 1 (cu sender si timp actualizate, dar creator, seq_no si lista_vecini nemodificate).

La tipul 4:

- creator este **ruterul sursa (nu se va modifica la ruterele de pe traseu)**
- seq_no este **setat doar la sursa (nu se va modifica la ruterele de pe traseu)**
- campul lista_vecini+costuri nu se foloseste

Restrictii:

Conform protocolului pe care trebuie sa-l implementati, **nu aveti voie sa includeti** in pachetele de tip 1-4:

- **tabela de rutare sau bucati din tabela de rutare** (exceptie next_hop)
- **topologia a unui ruter** (lista/matricea de adiacenta) (in cadrul mesajelor de tip 3 se trimit informatii pe baza carora se completeaza topologia – pachete LSA – dar **cate unul per mesaj**)

Mesajele pachet de date(tip=4)) nu contin propriu-zis date. Ele vor fi pachete rutate. Un pachet de date primit la pasul t va fi rutat conform tabelei de rutare de la sfarsitul pasului t-1.

Structuri de date:

Fiecare ruter va mentine (cel putin) urmatoarele structuri de date/variabile proprii. (E vorba de un sistem distribuit, nu exista structuri de date globale pentru rutere; un ruter nu are acces la structurile de date sau variabilele altui ruter, decat prin schimbul de mesaje)

- LSADatabase (o lista/matrice in care va tine maxim *numar_rutere* pachete LSA(de tip 1) – cel mai recent LSA despre fiecare ruter) ; se actualizeaza de oricate ori primeste un mesaj cu informatii mai **noi**
- **topologia** (lista de adiacenta sau matrice de adiacenta construita pe baza LSADatabase); se actualizeaza de cate ori se modifica si LSADatabase
- tabela de rutare (construita pe baza topologiei, cu un algoritm de shortest path); daca se gasesc cai multiple de cost egal catre aceeasi destinatie, se va alege calea in care urmatorul hop are id-ul cel mai mic; **se actualizeaza o singura data pe un pas de timp (la sfarsitul pasului de timp, dupa ce s-au procesat mesajele vechi si evenimentul, dar inainte de a trimite mesaje**

de tip 5)

- **coada de mesaje** (sau 2 cozi, una cu mesaje vechi, primite la timpul t-1, alta cu mesaje primite la timpul t)
- contor pentru mesajele create (“from scratch”) (numar de secventa pentru mesaje create)
- pasul de timp

Topologia se creeaza pe baza LSADatabase astfel. Notam cu LSA(ri) , LSA-ul avand creator pe ri. Fie t1 timpul in care a fost creat LSA(r1) si t2 timpul in care a fost creat LSA(r2) in LSADatabase a unui ruter oarecare r:

r.topologie[r1][r2] = r.topologie[r2][r1] = cost (r1,r2) conform r.LSA(r1), daca t1 >=t2

r.topologie[r1][r2] = r.topologie[r2][r1] =cost (r2,r1) conform r.LSA(r2), daca t2>t1

Daca unul din cele 2 LSA-uri nu exista, se ia in considerare cel existent. Daca nici unul din cele 2 LSA-uri nu exista, ruterul r inca nu a aflat de existenta celor 2 rutere r1 si r2.

Coada de mesaje: Daca optati pentru o singura structura gen FIFO, trebuie sa fiti atenti sa nu procesati mesaje noi (venite in pasul de timp curent). Va poate fi mai simplu sa tineti 2 cozi de mesaje si la fiecare pas de timp sa le interschimbati.

Procesare mesaje:

***Mesaj LSA(tip 1)*:** Un mesaj LSA se creeaza in cazul unor evenimente de tip 1, 2 sau 3 (atunci cand un ruter **din retea** constata ca a primit un nou vecin (ruter nou sau link nou) sau ca i-a picat un link). La primirea unui mesaj LSA, se verifica daca exista o copie sau un mesaj LSA mai **nou** avand acelasi creator – daca da, mesajul vechi sau copia se arunca (drop). Daca nu, se actualizeaza LSADatabase (si eventual si topologia), se actualizeaza campul sender si mesajul se trimite vecinilor (in afara de vecinul de la care a primit, pentru a evita – pe cat posibil - buclele).

Mesaj DatabaseReply(tip 3): Mesajele de acest tip se trimit **numai** ca raspuns la mesaje de tip 2(DatabaseRequest). Pachetele unui mesaj DatabaseReply se proceseaza la primire la fel ca cele LSA, doar ca nu se mai forwardeaza.

Procesarea mesajelor de tipuri 2-4 reiese din tratarea evenimentelor:

Atentie!

1)In tratarea evenimentelor (redata mai jos), trebuie sa tineti seama ca un mesaj din cadrul protocolului linkstate (tip 1,2,3,4) nu poate fi si trimis si procesat in acelasi pas de timp. Pentru ca procesul central nu buffer-eaza niciun pachet, el trebuie buffer-at (pus in coada) la ruterul receptor si procesat in pasul de timp urmator primirii. Mesaje de control, de tip 6, 7, 8 si 9, care se proceseaza imediat dupa primire, nu se buffer-eaza in nicio coada.

2) **Evenimentele se pastreaza intr-un mesaj special si se proceseaza la sfarsitul pasului de timp (mai exact dupa ce s-au procesat mesajele vechi din coada, dar inainte de calculul tabelei de rutare si transmiterea mesajelor de tip 5 – terminarea pasului de timp). Respectati aceasta ordine de prelucrare.**

E1. Actiuni la adaugarea unui nou ruter NewR. Fie t(s) momentul de timp al adaugarii noului ruter.

- Se trimit pachete Database Request (tip2) de la **NewR** catre toti vecinii lui
- Vecinii raspund lui NewR cu mesaje DatabaseReply
- Vecinii noului ruter construiesc pachet LSA(cu creator = id propriu) si il trimit tuturor vecinilor, inclusiv lui NewR

- Vecinii isi actualizeaza structurile de date
- **NewR isi** actualizeaza structurile de date pe baza informatiilor din pachetele DatabaseReply si LSA-urile de la vecini conform indicatiilor de mai sus

Mai sus au fost prezentate numai actiunile care sunt legate strict de aparitia noului ruter. Se vor procesa si alte mesaje in acesti pasi de timp, daca este cazul (de exemplu forwardarea pachetelor LSA si a pachetelor de rutare...).

E2. Actiuni la aparitia unui nou link (intre 2 rutere existente, intre care nu era o legatura in prealabil)

- Capetele link-ului creeaza si isi trimit unul altuia cate un pachet Database Request (si actualizeaza topologia proprie cu noul link) (update 9 aprilie)
- Fiecare capat raspunde celuilalt cu mesaje DatabaseReply
- Capetele linkului construiesc pachet LSA(cu creator = id propriu) si il trimit tuturor vecinilor directi, inclusiv celuilalt capat de link
- Capetele linkului isi actualizeaza LSADatabase(update 9 aprilie) pe baza evenimentului (cu noul LSA)
- Se proceseaza mesajele Database Reply primite de la celalalt capat al linkului, cu actualizarea structurilor de date daca e cazul

E3. Actiuni la suprimarea unui link (intre 2 rutere existente, intre care era o legatura in prealabil).

- Capetele link-ului sterg din topologia proprie linkul picat si construiesc pachet LSA(cu creator = id propriu) si il trimit tuturor vecinilor directi (capatul opus al linkului nu mai este vecin acum); se actualizeaza LSADatabase ale capetelor linkului, pe baza LSA-ului propriu

Observati ca informatia pentru un link picat va fi mai devreme raspandita in retea decat informatia unui link nou.

E4. Dirijare pachet

- Ruterul sursa construieste un pachet de date si-l trimite primului hop (conform tabelei de rutare)
- fiecare ruter de pe traseu dirijeaza mesajul cu un hop mai departe, conform tabelei de rutare, pana se ajunge la destinatie

Tabela de rutare a unui ruter se va calcula maxim o data pentru fiecare pas de timp (la finalul pasului de timp, dupa procesarea evenimentului dar inainte de mesajele de tip 5). Deci rutarea foloseste tabela de rutare creata la sfarsitul pasului anterior.

Indicatii pentru implementare

Se va simula activitatea din retea, conform specificatiilor din enunt, cati pasi de timp sunt necesari pentru stabilizarea retelei (pana nu mai exista niciun mesaj in cozi si s-au terminat evenimentele). Atentie, mesaje se mai trimit si dupa ce nu nu mai au loc evenimente.

Evenimentele se proceseaza in mai multi pasi de timp. Functia *procesare_eveniment()* va contine numai primul pas de timp din fiecare eveniment (primul bullet). Ceilalti pasi rezulta din procesarea mesajelor generate la un pas anterior.

Punctaj:

Pentru a lua punctaj >10/100, trebuie sa aveti implementat corect calculul tabelelor de rutare. Numai asa se poate verifica validitatea punctelor de mai jos (nu se va verifica alt output in afara fisierului log_rutare):

a) 35 puncte daca tema voastra implementeaza CORECT evenimentele de tip 1.
b) 25 puncte daca tema voastra implementeaza CORECT evenimentele de tip 2 (in conditiile in care cel putin a) este indeplinit)
c) 25 puncte daca tema voastra implementeaza CORECT evenimentele de tip 3 (in conditiile in care cel putin a) este indeplinit)
d) 15 puncte daca tema voastra implementeaza CORECT rutarea (eveniment 4) (in conditiile in care cel putin a) este indeplinit)
CORECT inseamna conform specificatiilor din enunt(si comentariilor din rutare.c), cu procesarea adecvata a tipurile de pachete si structuri de date implicate in tratarea respectivului evenimen. **Temele voastre vor fi evaluate prin inspectarea fisierului de log, log_rutare si a codului sursa (rutare.c si fisiere ajutatoare lui).** (Procesul central logheaza tabelele de rutare la fiecare pas si toate mesajele de tip 4. Folositi tabela de rutare deja declarata si initializata. Pentru pachete de tip 4, setati campurile necesare logarii, mentionate in comentariile din ruter.c).

Deci chiar daca nu implementati evenimentul de rutare, trebuie sa actualizati corect tabela de rutare ca sa va poata fi evaluata tema pentru punctele a,b,c. Punctele b,c si d se vor testa, daca e cazul - spcificati in readme, in conditiile minime (numai a indeplinit) folosind fisiere de input care contin numai o parte din tipurile de evenimente - cele pe care declarati voi in readme ca le-ati tratat corect.

Tema nu poate fi corectata daca nu aveti output corect – in log_rutare (macar pentru o parte din tipurile de evenimente).

Tema va fi testata si cu fisiere de input care nu s-au pus la dispozitia studentilor. Implementarea voastra nu trebuie sa mearga doar pe exemplul furnizat. Outputul nu se va verifica cu comanda diff (fisierele de iesire nu vor fi identice): din cauza faptului ca procesul central primeste mesaje de la mai multe procese, tabelele de rutare pot sa nu fie afisate in ordinea id-ului ruterelor active la un anumit pas. Dar continutul tabelei de rutare de la un anumit id si pas ar trebui sa semene cu exemplul oferit.

Conditii pentru depunctari:

- nerespectarea specificatiilor
- lipsa *readme sau readme nerelevat
- **makefile nefunctional (punctaj 0)
- **tema care nu se compileaza (punctaj 0)
- **tema care nu ruleaza (punctaj 0) corect – aveti la dispozitie un tutorial de debugging
- tema nu are output (corect) in log_rutare pentru respectivul eveniment
- copierea temei (0 pe tema + anularea punctajului din laborator+teme, conform regulamentului cursului de PC, atat la sursa cat si la destinatie)

*In readme nu reluati enuntul, explicati optiunile personale in implementare (cat va permit specificatiile), alegerea structurilor de date, actualizarea lor, etc...
** Se tine cont de faptul ca noi v-am furnizat in resurse tema 2 un Makefile functional si programe fara erori/warning-uri la compilare si rulare.

Format fisier de intrare / parsare mesaje de tip 7

Procesul central primeste un singur parametru la rulare, si anume un fisier de intrare in care sunt evenimentele. (initial se considera ca topologia contine numai routerul 0). Se garanteaza ca evenimentele din fisier sunt in pasi de timpi **pari** consecutivi(incepand cu pasul de timp 0: 0, 2, 4..), iar la fiecare pas **par** de timp are loc un singur event. Am ales aceasta distantare intre evenimente, ca reteaua sa se stabilizeze putin inainte de urmatorul eveniment. Atentie, simularea voastra trebuie sa

dureze mai mult decat ultimul pas de timp la care a avut loc un eveniment.

Fisierul de intrare este de forma:
Pe prima linie numarul n_ev de evenimente, iar pe urmatoarele n_ev linii:

- tip_event format_event

event=1:

- 1 id_ruter_nou nrvec vecin1 cost1 … vecin_nrvec cost_nrevec
- Exemplu: 1 6 3 1 4 2 3 4 2

(Eveniment adaugare ruter nou, id_ruter_nou=6, numar_vecini = 3, vecin1=1, cost1=4, vecin2=2, cost2=3, vecin3=4, cost3=2) Cost(i) = cost (id_nou, vecin(i))
event=2:

- 2 ruter1 ruter2 cost
- Exemplu: 2 4 5 2

(Eveniment adaugare link nou intre ruterele 4 si 5, cu cost link=2)
event=3:

- 3 ruter1 ruter2
- Exemplu: 3 2 4

(Eveniment suprimare link intre ruterele 2 si 4)
event=4:

- 4 sursa destinatie
- Exemplu: 4 6 10

(Rutare pachet de la ruter 6 la ruter 10)
Exemplu fisier de intrare: Vezi resurse tema 2