Курсова робота на тему

“Алгоритми маршрутизації в мережах”

***Зміст***

**1. Вступ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 2**

**2. Маршрутизаційні рішення \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 5**

**2.1. RIP : Алгоритм вектору відстані \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 5**

**2.2. Dual IS-IS : Алгоритм відкриття найкоротшого**

**шляху \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 8**

**2.2.1. Огляд алгоритму \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 8**

**2.2.2.** **Реалізація алгоритму відкриття найкоротшого шляху в DUAL IS-IS середовищі\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 9**

1. **Висновки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 13**
2. **Глоссарій\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 13**

**5. Список літератури\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 14**

**6. Додаток\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст 15**

**1. Вступ**

В наш час комп’ютерні мережі перебувають в стані розвитку й набувають широкого розповсюдження. Лише комп’ютерна мережа Internet в даний час розрахована на 4.294.967.296 комп’ютерів, які матимуть IP адреси. До цього числа слід додати чиленні локальні та корпоративні мережі. Всі ці комп’ютери були з’єднані з метою обміну інформацією і власники комп’ютерів жадають швидкої передачі великої кількості інформації на значні відстані.

Вимоги користувачів мережі задовольняються покращенням якісті каналів передачі даних на заміну телефонним дротам прийшлі оптично-волоконні лінії, канали передачі даних за допомогою супутникового зв’язку тощо. Але значну роль при такій кількості з’єднаних в мережу комп’ютерів відіграє якість протоколів, за допомогою яких здійснюється передача даних

між серверами, протоколів маршрутизації, алгоритмів на яких вони побудовані.

Враховуючи, що 4-байтну адресацію в мережі Internet буде замінено 8-байтною, тобто максимальна кількість комп’ютерів під’єднаних до мережі зросте у 4.294.967.296 разів, слід зазначити, що найбільшу роль відіграватиме покращення саме механізму маршрутизації пакетів даних між серверами мережі Internet.

Маршрутизація – це задача знаходження шляху між комп’ютером, що відсилає дані та комп’ютером-одержувачем, але в зв’язаній моделі IP ця задача в основному зводиться до пошуку шляхів до шлюзів між мережами. Поки пакети даних знаходяться на окремій мережі або підмережі проблеми маршрутизації вирішуються за технологією, специфічною для інтерфейсу цієї мережі. IP маршрутизація починається, коли потрібно передати дані між різними мережами з різними інтерфейсами. Якщо мережі отримувача та відправника безпосередньо зв’язані, то дані мають пройти через шлюз, що з’єднує мережі. Якщо ці мережі не зв’язані шлюзом, дані мають пройти через мережі, що знаходяться між відправником і одержувачем та шлюзами що їх з’єднують.Як тільки дані доходять до шлюзу на мережі отримувача, технологія маршрутизації цієї мережі спрямовує дані до отримувача.

Для знаходження маршруту до комп’ютера-отримувача система зберігає таблиці маршрутизації, які використовуються протоколами мережного рівня для вибору потрібного мережного інтерфейсу. Маршрутизаційна інформація зберігається у вигляді двох таблиць: перша – для маршрутів до хостів, друга – для маршрутів до мереж. Такий підхід дозволяє використовувати універсальні механізми визначення маршрутів як для мереж із розподіленим середовищем передачі даних , так і для мереж типу point-to-point. Визначаючи маршрут, модуль мережного протоколу (IP) спочатку переглядає таблиці для хостів, а потім для мереж. Якщо пошук не дає результату, то використовується маршрут по замовчуванню.

Визначення маршруту може базуватися на різноманітних показниках або комбінаціях показників. Програмні реалізації алгоритмів маршрутизації вираховують вартість маршруту для визначення оптимальних маршрутів до

пункту призначення.

В таблиці 1 наведено приклад таблиці типу пункт призначення/наступний об’єкт для пересилання пакетів.

|  |  |
| --- | --- |
| Мережа призначення | Наступний об’єкт |
| 57 | вершина С |
| 24 | вершина В |
| 26 | вершина В |
| 18 | вершина А |
| 20 | вершина С |
| 34 | вершина А |
| 28 | вершина А |

Таблиця 1 : маршрутизаційна таблиця типу пункт призначення/наступний об’єкт для пересилання пакетів

Існує багато підходів до задач пошуку оптимальних шляхів в мережі, що реалізовані в протоколах, за якими відбувається маршрутизація, таких як Interior Gateway Protocols: OSPF (Open Shortest Path First), Dual IS-IS (Intermediate System to Intermediate System), RIP (Routing Information Protocol), GGP (Gateway to Gateway Protocol); Exterior Gateway Protocols: BGP (Border Gateway Protocol), EGP (Exterior Gateway Protocol), Inter-AS Routing without Exterior Gateway; Static Routing.

Найрозповсюдженішими в Internet є реалізації алгоритмів вектору відстані та відкриття найкоротшого шляху.

Алгоритми відкриття найкоротшого маршруту, також відомі як алгоритми стану канала направляють потоки маршрутизаційної інформации до всіх вузлів об'єднаної мережі. Але кожний маршрутизатор відсилає лише ту

частину маршрутизаційної таблиці, котра описує стан його власних каналів. Алгоритми вектору відстані (також відомі, як алгоритми Белмана-Форда) вимагають від кожногo маршрутизатора відсилки всієї або частини своєї

маршрутизаційної таблиці, але лише своїм сусідам. Алгоритми відкриття найкоротшого маршруту фактично направляють невеликі корекції по всіх напрямках, в той час як алгоритми вектору відстані відсилають більш великі корекції лише до сусідніх маршрутизаторів.

Відрізняючись більш швидкою сходимістю, алгоритми відкриття найкоротшого маршруту трохи менше схильні до утворення петель маршрутизації, ніж алгоритми вектору відстані. З іншого боку, алгоритми відкриття найкоротшого маршруту характеризуються більш складними розрахунками в порівнянні з алгоритмами вектору відстані, потребуючи більшої процесорної потужності та пам’яті, ніж алгоритми вектору відстані. В зв’язку з цим, реалізація та підтримка алгоритмів відкриття найкоротшого маршруту може бути більш дорогою. Не дивлячись на їхні відмінності, обидва типи алгоритмів добре функціонують за самих різноманітних умов.

В додатку наведено текст демону Routed, який реалізований у маршрутизаційному протоколі RIP на основі алгоритму вектору відстані.

2. Маршрутизаційні рішення

**2.1. *RIP : Алгоритми вектору відстані***

Алгоритми вектору відстані основані на обміні малої кількості інформації. Кожний об’єкт (шлюз або хост), що приймає участь в маршрутизації, має тримати інформацію про всі комп’ютери системі. Кожний запис в таблиці маршрутизації включає наступний шлюз, на який дані, напрямлені до об’єкту, мають бути відправлені. До того ж він має містити значення, що характеризує загальну відстань до об’єкту. Відстань - це узагальнена характеристика, що може відображати швидкість передачі даних, грошову вартість передачі тощо. Алгоритми вектору вістані дістали свою назву від того, що вони можуть обчислити оптимальний маршрут коли змінюється лише список відстаней. Крім того, має місце обмін маршрутизаційною інформацією між безпосередньо зв’язаними об’єктами, тобто елементами спільної мережі. Записи в таблиці маршрутизації мають містити таку інформацію про комп’ютер-отримувач:

1. адреса : в IP реалізації це має бути IP адреса хоста або мережі;
2. шлюз : перший шлюз на цьому маршруті;
3. інтерфейс : інтерфейс, що має бути використований, щоб досягти першого шлюза;
4. ціна : число, що визначає відстань до комп’ютера-отримувача;
5. таймер : проміжок часу з того моменту коли інформація була востаннє оновлена.

На додаток можуть бути додані різні флаги та інша інформація. Таблиця починається з опису об’єктів, що прямо під’єднані до системи. Вона оновлюється на основі інформації, що приходить від сусідніх шлюзів. Найважливіша інформація, якою обмінюються хости та шлюзи міститься в звітах оновлення. Кожний об’єкт, що бере участь в маршрутизації посилає звіти оновлення, що описують таблиці маршрутизації в тому стані, в якому вони знаходяться на даний момент. Можливо визначити оптимальний маршрут користуючись лише інформацією отриманою від сусідніх об’єктів.

Алгоритми вектору відстані базуються на таблиці даючи найкращий маршрут з міркувань ціни (будь-якої характериски,яку можна представити у вигляді суми ребер графу) . Формально, якщо вершини i та j сусідні, тоді ціна d(i,j) рівна ціні ребра між i та j. В нормальному випадку всі об’єкти на даній мережі однакові, d(i,j) однакове для всіх об’єктів даноі мережі і визначає ціну використовування цієї мережі. Щоб дістати ціну всього маршруту, потрібно додати ціни всіх ребер, що складають цей маршрут. Для зручності припустимо, що ціна – додатнє ціле. Нехай D(i,j) визначає ціну кращого маршруту з вершини i до вершини j. Вона має бути визначена для кожного ребра. d(i,j) визначатиме ціни окремих кроків. Формально, нехай d(i,j) визначає ціну маршруту, йдучи прямо з вершини i до вершини j. Це значення дорівнює нескінченності, якщо i та j не сусідні вершини. (Слід зауважити, що d(i,i) - нескінченність. Це так, якщо не брати до уваги, що вершина може мати пряме з’єднання до самої себе). Оскільки ціни адитивні, то можна показати отримання кращого маршруту так :

D(i,i) = 0, для всіх i

D(i,j) = min [d(i,k) + D(k,j)], інакше k

і найкращий маршрут той, що починається з вершини i до тих вершин k, для яких d(i,k) + D(k,j) мінімальне.

Ми можеме обмежити друге рівняння для тих k, що є сусідами i. Для інших d(i,k) нескінченність, тому вони не можуть дати мінімального значення.На основі цього можливо обчислити відстань. Об’єкт i примушує його сусідів k прислати ціну шляху до об’єкту призначення j. Коли i отримує ціну d(k,j) від всіх k, він додає d(k,j) до ціни шляху D(i,k). Потім і порівнює значення від всіх сусідів і вибирає найменше.

Реальні реалізації алгоритму запам’ятовують найкращу ціну й ідентифікацію сусіда, що її надіслав. Інформація заміщається, коли надсилається менша ціна. Це дозволяє обраховувати мінімум, не зберігаючи інформацію від всіх сусідів. Але в випадку, коли інформація надходить від об’єкта, що був записаний в таблиці як найкращий, інформація оновлюється в будь-якому випадку. Механізм визначення найкращого маршруту передбачає крах об’єкту на ділянці цього маршруту. В зв’язку з цим встановлено, що об’єкти мають відсилати оновлену інформаціію кожні 30 секунд. Якщо об’єкт, що дає кращу ціну, не відповідає протягом 180 секунд (враховується можливість втрати пакету), ціна шляху встановлюється в дуже велике значення.

**2.2. OSPF, Dual IS-IS: Алгоритм відкриття найкоротшого шляху**

***2.2.1.* *Огляд алгоритму.***

Алгоритм відкриття найкоротшого шляху використовується як в IP, так і в OSI середовищі. Він має складність О(N2).

Основний алгоритм, що будує PATHS з нуля, починає додавання систем з найвигіднішими маршрутами з оглядом на PATHS (не може існувати коротшого маршруту до SELF ). Потім визначається TENT використовуючи локальні таблиці з відомостями про сусідні вершини.

Система не може бути розміщеною в PATHS до тих пір, поки не доведено, що не існує маршруту, коротшого за даний. Коли система N розміщується в PATHS, перевіряється ціна маршруту до кожної вершини M сусідньої до N через саму вершину N. Цей маршрут визначається як сума ціни маршруту до N та ціни ділянки NM. Якщо <M,\*,\*> розміщений в TENT та нове значення буде більшим, маршрут ігнорується.Якщо <M,\*,\*> розміщений в TENT та нове значення буде меншим, старий запис заміщується новим. Якщо <M,\*,\*> розміщений в TENT та нове значення таке ж саме як те, що вже є в TENT то набір {Adj(M)} встановлюється як поєднання старого запису (того, що міститься в TENT) та нового - {Adj(N)}. Якщо M не знаходиться в TENT, то даний маршрут додається в TENT.

Потім алгоритм знаходить триплети <N,x,{Adj(N)}> in TENT з мінімальним x.

***2.2.2. Реалізація алгоритму відкриття найкоротшого шляху в DUAL IS-IS середовищі***

Крок 0: Встановимо TENT та PATHS як пусті. Встановимо tentlength в 0.

(tentlength – це довжина шляху досліджуваних елементів TENT.)

1) Додамо <SELF,0,W> до PATHS, де SELF – початкова система, W –спеціальна величина, що визначає трафік до SELF що пройдений, включаючи внутрішній процес.

2) Тепер загрузимо TENT локальними даними шляхів (Кожен запис в TENT має бути визначений як маршрутизатор або кінцева система OSI, щоб дозволити правильну перевірку в Кроці 2).

Для всіх суміжних вершин Adj(N) на всіх можливих каналах:

d(N) = ціна маршруту, що проходить через (N)

Adj(N) = кількість вершин сусідніх N.

3) Якщо триплет <N,x,{Adj(M)}> в TENT, то

Якщо x = d(N), то {Adj(M)} := {Adj(M)} U {Adj(N)}.

4) Якщо N – маршрутизатор або кінцева система OSI, і більше не існує суміжних вершин {Adj(M)} то видалимо надлишкову вершину.

5) Якщо x < d(N), нічого.

6) Якщо x > d(N), видалити <N,x,{Adj(M)}> з TENT і додати триплет <N,d(N),{Adj(N)}>.

7) Якщо <N,x,{Adj(M)}> не в TENT, то додати <N,d(N),{Adj(N)}> в TENT.

8) Тепер додаються системи, для яких локальний маршрутизатор не має суміжних вершин, але вони згадані в сусідніх псевдовершинах LSP. Суміжність для таких систем визначається маршрутизатором.

9) Для всіх широковєщательних каналів в активному стані, знайти псевдовершину LSP для цього каналу. Якщо така існує, для всіх сусідів N, про які згадувається на цій вершині і не визначені в TENT, додати запис:

<N,d(N),{Adj(N)}> to TENT, where:

d(N) = ціна проміжку .

Adj(N) = кількість вершин, що стоять на шляху до заданого маршрутизатора.

10) Перейти в Крок 2.

Крок 1: Визначити нульовий PDU в LSP ситеми, щойно доданої в PATHS

1)dist(P,N) = d(P) + metric.k(P,N) для кожного сусіда N (як для кінцевої системи, так і для маршрутизатора) системи P.

2) Якщо dist(P,N) >максимальної ціни проміжку, нічого.

3) Якщо <N,d(N),{Adj(N)}> є в PATHS, нічого.

d(N) повинне бути меншим ніж dist(P,N), або N не повинне бути в PATHS. За бажанням можна зробити додаткову перевірку чи є d(N) меншим за dist(P,N).

4) Якщо триплет <N,x,{Adj(N)}> в TENT, тоді:

a) Якщо x = dist(P,N) тоді {Adj(N)}:= {Adj(N)} U {Adj(P)}.

b) Якщо N – маршрутизатор або кінцева система OSI, і більше не існує суміжних вершин {Adj(M)}, то видалимо надлишкову вершину.

c) Якщо x < dist(P,N), нічого.

d) Якщо x > dist(P,N), видалити <N,x,{Adj(N)}> з TENT, та додати <N,dist(P,N),{Adj(P)}>

5) Якщо <N,x,{Adj(N)}> не в TENT, додати <N,dist(P,N),{Adj(P)}> в TENT.

Крок 2: Якщо TENT пустий, зупинитися. Інакше:

1) Знайти елемент <P,x,{Adj(P)}>, з мінімальним x таким чином:

a)Якщо елемент <\*,tentlength,\*> залишився в TENT в списку для tentlength, вибрати цей елемент. Якщо в списку існує більше одного елементу, вибрати один з цих елементів для системи, що є псевдовершиною, вибрати ту, що не є псевдовершиною. Якщо більше нема елементів в списку для tentlength, збільшити tentlength і повторити Крок 2.

b)Видалити <P,tentlength,{Adj(P)}> з TENT.

c) Додати <P,d(P),{Adj(P)}> в PATHS.

d) Якщо система тільки що додана в PATHS – кінцева система, то перейти в Крок 2. Інакше : перейти в Крок 1.

Позначення:

PATHS – представляє ациклічний граф найкоротших шляхів від системи S. Він представляється як набір триплетів <N,d(N),{Adj(N)}>, де N ідентифікатор системи. d(N) загальна відстань від N до S).

{Adj(N)} –набір працюючих сусідів S, що їх можна використати N. Якщо система є в PATHS, шляхи, що відповідають цьому місцю є найкоротшими.

TENT – список триплетів у вигляді <N,d(N),{Adj(N)}>, де N, d(N) та {Adj(N)} відповідають визначеним в PATHS.

TENT може бути інтуітивно представлений як місце системи в PATHS. Іншими словами, триплет <N,x,{A}> в TENT говорить, що, якщо N є в PATHS, d(N) відповідає x, але N не може бути розміщене в PATHS поки не доведено, що не існує шляхів, коротших за x .

Так само <N,x,{A,B}> в TENT значить, що якщо N є в PATHS, тоді d(N) буде дорівнювати x для маршрутів, що проходять через суміжну вершину A або через суміжну вершину B.

Запропоновано в реальній реалізації таблиці TENT проводити сортування за характеристикою d(N).

**3. Висновки**

Маршрутизаційні алгоритми реалізовані на різних типах мереж від локальних до глобальних. Широко розповсюдженим є демон Routed з дистриутиву університету Каліфорнії в Берклі він реалізований в протоколі RIP. Також велике значення мають реалізації алгоритму відкриття найкоротшого маршруту для подвійного середовища OSI та TCP/IP в плані знаходження маршрутів між інтер-автономними системами та маршрутизаторами TCP/IP архитектури.

**Глоссарій**

OSI – мережна модель, запропонована організацією по стандартизації ISO

IS – Interautonomous system – інтеравтономна система, система, що приймає участь в маршрутизації в моделі OSI

ES - End System-кінцева система, система, що не приймає участі в маршрутизації в моделі OSI

Router – маршрутизатор, об’єкт маршрутизації

Gateway – шлюз, система, що має декілька мережних інтерфейсів

RIP (Routing Information Protocol) – маршрутизаційний інформаційний протокол

OSPF (Open Shortest Path First) – Маршрутизаційний протокол відкриття найкоротшого шляху

Література :

1. C.L. Hedrick. Routing Information Protocol. RFC 1058 Jun-01-1988.
2. D. Waitzman, C.Partridge, S.E. Deering. Distance Vector Multicast Routing Protocol. RFC 1075 Nov-01-1988.
3. R.W. Callon. Use of OSI IS-IS for routing in TCP/IP and dual environments.

RFC 1195 Dec-01-1990.

1. P. Almquist, F. Kastenholz. Towards Requirements for IP Routers. RFC 1716 November 1994.
2. J. Moy. , OSPF Version 2. RFC 2178 July 1997.
3. A. Ballardie. Core Based Trees (CBT) Multicast Routing Architecture. RFC 2201September 1997.
4. Bellman, R. E., "Dynamic Programming", Princeton University Press, Princeton, N.J., 1957.
5. Bertsekas, D. P., and Gallaher, R. G., "Data Networks",Prentice-Hall, Englewood Cliffs, N.J., 1987.
6. Braden, R., and Postel, J., "Requirements for Internet Gateways", USC/Information Sciences Institute, RFC-1009, June 1987.
7. Boggs, D. R., Shoch, J. F., Taft, E. A., and Metcalfe, R. M.,"Pup: An Internetwork Architecture", IEEE Transactions on Communications, April 1980.
8. Clark, D. D., "Fault Isolation and Recovery," MIT-LCS, RFC-816, July 1982.
9. Xerox Corp., "Internet Transport Protocols", Xerox System Integration Standard XSIS 028112, December 1981.
10. Ford, L.R. Jr., and Fulkerson, D.R.,"Flows in Networks", Princeton University Press, Princeton, N.J., 1962.
11. "Intermediate System to Intermediate System Intra-Domain Routeing Exchange Protocol for use in Conjunction with the Protocol for Providing the Connectionless-mode Network Service (ISO 8473)", ISO DP 10589, February 1990.
12. "Protocol for Providing the Connectionless-Mode Network Service", ISO 8473, March 1987.
13. ”End System to Intermediate System Routeing Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service (ISO 8473)", ISO 9542, March 1988.
14. Braden,R., and Postel,J., "Requirements for Internet Gateways", RFC 1009, June 1987.
15. Moy,J., "The OSPF Specification", RFC 1131, October 1989.
16. Postel,J., "Internetwork Protocol", RFC 791, September 1981.

Postel,J., "Internet Control Message Protocol", RFC 792, September 1981.

20. GOSIP Advanced Requirements Group, "Government Open Systems

Interconnection Profile (GOSIP) Version 2.0 [Final Text]", Federal Information Processing Standard, U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, October 1990.

21. "Standard for Local Area Networks and Metropolitan Area Networks: Overview and Architecture of Network Standards",IEEE Standard 802.1a-1990.

Додаток

#include "defs.h"

#include "pathnames.h"

#ifdef sgi

#include "math.h"

#endif

#include <signal.h>

#include <fcntl.h>

#include <sys/file.h>

pid\_t mypid;

naddr myaddr; /\* system address \*/

char myname[MAXHOSTNAMELEN+1];

int supplier; /\* supply or broadcast updates \*/

int supplier\_set;

int ipforwarding = 1; /\* kernel forwarding on \*/

int default\_gateway; /\* 1=advertise default \*/

int background = 1;

int ridhosts; /\* 1=reduce host routes \*/

int mhome; /\* 1=want multi-homed host route \*/

int advertise\_mhome; /\* 1=must continue adverising it \*/

int auth\_ok = 1; /\* 1=ignore auth if we do not care \*/

struct timeval epoch; /\* when started \*/

struct timeval clk, prev\_clk;

struct timeval now; /\* current idea of time \*/

time\_t now\_stale;

time\_t now\_expire;

time\_t now\_garbage;

struct timeval next\_bcast; /\* next general broadcast \*/

struct timeval no\_flash = {EPOCH+SUPPLY\_INTERVAL}; /\* inhibit flash update \*/

fd\_set fdbits;

int sock\_max;

int rip\_sock = -1; /\* RIP socket \*/

struct interface \*rip\_sock\_mcast; /\* current multicast interface \*/

int rt\_sock; /\* routing socket \*/

int rt\_sock\_seqno;

static int get\_rip\_sock(naddr, int);

static void timevalsub(struct timeval \*, struct timeval \*, struct timeval \*);

int

main(int argc,

char \*argv[])

{

int n, mib[4], off;

size\_t len;

char \*p, \*q;

struct timeval wtime, t2;

time\_t dt;

fd\_set ibits;

naddr p\_net, p\_mask;

struct interface \*ifp;

struct parm parm;

char \*tracename = 0;

/\* Some shells are badly broken and send SIGHUP to backgrounded

\* processes.

\*/

signal(SIGHUP, SIG\_IGN);

openlog("routed", LOG\_PID | LOG\_ODELAY, LOG\_DAEMON);

ftrace = stdout;

gettimeofday(&clk, 0);

prev\_clk = clk;

epoch = clk;

epoch.tv\_sec -= EPOCH;

now.tv\_sec = EPOCH;

now\_stale = EPOCH - STALE\_TIME;

now\_expire = EPOCH - EXPIRE\_TIME;

now\_garbage = EPOCH - GARBAGE\_TIME;

wtime.tv\_sec = 0;

(void)gethostname(myname, sizeof(myname)-1);

(void)gethost(myname, &myaddr);

while ((n = getopt(argc, argv, "sqdghmAtT:F:P:")) != -1) {

switch (n) {

case 's':

supplier = 1;

supplier\_set = 1;

break;

case 'q':

supplier = 0;

supplier\_set = 1;

break;

case 'd':

background = 0;

break;

case 'g':

bzero(&parm, sizeof(parm));

parm.parm\_d\_metric = 1;

p = check\_parms(&parm);

if (p != 0)

msglog("bad -g: %s", p);

else

default\_gateway = 1;

break;

case 'h': /\* suppress extra host routes \*/

ridhosts = 1;

break;

case 'm': /\* advertise host route \*/

mhome = 1; /\* on multi-homed hosts \*/

break;

case 'A':

/\* Ignore authentication if we do not care.

\* Crazy as it is, that is what RFC 1723 requires.

\*/

auth\_ok = 0;

break;

case 't':

new\_tracelevel++;

break;

case 'T':

tracename = optarg;

break;

case 'F': /\* minimal routes for SLIP \*/

n = FAKE\_METRIC;

p = strchr(optarg,',');

if (p && \*p != '\0') {

n = (int)strtoul(p+1, &q, 0);

if (\*q == '\0'

&& n <= HOPCNT\_INFINITY-1

&& n >= 1)

\*p = '\0';

}

if (!getnet(optarg, &p\_net, &p\_mask)) {

msglog("bad network; \"-F %s\"",

optarg);

break;

}

bzero(&parm, sizeof(parm));

parm.parm\_net = p\_net;

parm.parm\_mask = p\_mask;

parm.parm\_d\_metric = n;

p = check\_parms(&parm);

if (p != 0)

msglog("bad -F: %s", p);

break;

case 'P':

/\* handle arbirary, (usually) per-interface

\* parameters.

\*/

p = parse\_parms(optarg, 0);

if (p != 0) {

if (strcasecmp(p,optarg))

msglog("%s in \"%s\"", p, optarg);

else

msglog("bad \"-P %s\"", optarg);

}

break;

default:

goto usage;

}

}

argc -= optind;

argv += optind;

if (tracename == 0 && argc >= 1) {

tracename = \*argv++;

argc--;

}

if (tracename != 0 && tracename[0] == '\0')

goto usage;

if (argc != 0) {

usage:

logbad(0, "usage: routed [-sqdghmAt] [-T tracefile]"

" [-F net[/mask[,metric]]] [-P parms]");

}

if (geteuid() != 0)

logbad(0, "requires UID 0");

mib[0] = CTL\_NET;

mib[1] = PF\_INET;

mib[2] = IPPROTO\_IP;

mib[3] = IPCTL\_FORWARDING;

len = sizeof(ipforwarding);

if (sysctl(mib, 4, &ipforwarding, &len, 0, 0) < 0)

LOGERR("sysctl(IPCTL\_FORWARDING)");

if (!ipforwarding) {

if (supplier)

msglog("-s incompatible with ipforwarding=0");

if (default\_gateway) {

msglog("-g incompatible with ipforwarding=0");

default\_gateway = 0;

}

supplier = 0;

supplier\_set = 1;

}

if (default\_gateway) {

if (supplier\_set && !supplier) {

msglog("-g and -q incompatible");

} else {

supplier = 1;

supplier\_set = 1;

}

}

signal(SIGALRM, sigalrm);

if (!background)

signal(SIGHUP, sigterm); /\* SIGHUP fatal during debugging \*/

signal(SIGTERM, sigterm);

signal(SIGINT, sigterm);

signal(SIGUSR1, sigtrace\_on);

signal(SIGUSR2, sigtrace\_off);

/\* get into the background \*/

#ifdef sgi

if (0 > \_daemonize(background ? 0 : (\_DF\_NOCHDIR|\_DF\_NOFORK),

new\_tracelevel == 0 ? -1 : STDOUT\_FILENO,

new\_tracelevel == 0 ? -1 : STDERR\_FILENO,

-1))

BADERR(0, "\_daemonize()");

#else

if (background && daemon(0, new\_tracelevel) < 0)

BADERR(0,"daemon()");

#endif

mypid = getpid();

srandom((int)(clk.tv\_sec ^ clk.tv\_usec ^ mypid));

/\* prepare socket connected to the kernel.

\*/

rt\_sock = socket(AF\_ROUTE, SOCK\_RAW, 0);

if (rt\_sock < 0)

BADERR(1,"rt\_sock = socket()");

if (fcntl(rt\_sock, F\_SETFL, O\_NONBLOCK) == -1)

logbad(1, "fcntl(rt\_sock) O\_NONBLOCK: %s", strerror(errno));

off = 0;

if (setsockopt(rt\_sock, SOL\_SOCKET,SO\_USELOOPBACK,

&off,sizeof(off)) < 0)

LOGERR("setsockopt(SO\_USELOOPBACK,0)");

fix\_select();

if (background && new\_tracelevel == 0)

ftrace = 0;

if (tracename != 0) {

strncpy(inittracename, tracename, sizeof(inittracename)-1);

set\_tracefile(inittracename, "%s\n", -1);

} else {

tracelevel\_msg("%s\n", -1); /\* turn on tracing to stdio \*/

}

bufinit();

/\* initialize radix tree \*/

rtinit();

/\* Pick a random part of the second for our output to minimize

\* collisions.

\*

\* Start broadcasting after hearing from other routers, and

\* at a random time so a bunch of systems do not get synchronized

\* after a power failure.

\*/

intvl\_random(&next\_bcast, EPOCH+MIN\_WAITTIME, EPOCH+SUPPLY\_INTERVAL);

age\_timer.tv\_usec = next\_bcast.tv\_usec;

age\_timer.tv\_sec = EPOCH+MIN\_WAITTIME;

rdisc\_timer = next\_bcast;

ifinit\_timer.tv\_usec = next\_bcast.tv\_usec;

/\* Collect an initial view of the world by checking the interface

\* configuration and the kludge file.

\*/

gwkludge();

ifinit();

flush\_kern();

/\* Ask for routes \*/

rip\_query();

rdisc\_sol();

/\* Loop forever, listening and broadcasting.

\*/

for (;;) {

prev\_clk = clk;

gettimeofday(&clk, 0);

timevalsub(&t2, &clk, &prev\_clk);

if (t2.tv\_sec < 0

|| t2.tv\_sec > wtime.tv\_sec + 5) {

/\* Deal with time changes before other housekeeping to

\* keep everything straight.

\*/

dt = t2.tv\_sec;

if (dt > 0)

dt -= wtime.tv\_sec;

trace\_act("time changed by %d sec", dt);

epoch.tv\_sec += dt;

}

timevalsub(&now, &clk, &epoch);

now\_stale = now.tv\_sec - STALE\_TIME;

now\_expire = now.tv\_sec - EXPIRE\_TIME;

now\_garbage = now.tv\_sec - GARBAGE\_TIME;

/\* deal with signals that should affect tracing \*/

set\_tracelevel();

if (stopint != 0) {

rip\_bcast(0);

rdisc\_adv();

trace\_off("exiting with signal %d\n", stopint);

exit(stopint | 128);

}

/\* look for new or dead interfaces \*/

timevalsub(&wtime, &ifinit\_timer, &now);

if (wtime.tv\_sec <= 0) {

wtime.tv\_sec = 0;

ifinit();

rip\_query();

continue;

}

/\* If it is time, then broadcast our routes.

\*/

if (supplier || advertise\_mhome) {

timevalsub(&t2, &next\_bcast, &now);

if (t2.tv\_sec <= 0) {

/\* Synchronize the aging and broadcast

\* timers to minimize awakenings

\*/

age(0);

rip\_bcast(0);

/\* It is desirable to send routing updates

\* regularly. So schedule the next update

\* 30 seconds after the previous one was

\* secheduled, instead of 30 seconds after

\* the previous update was finished.

\* Even if we just started after discovering

\* a 2nd interface or were otherwise delayed,

\* pick a 30-second aniversary of the

\* original broadcast time.

\*/

n = 1 + (0-t2.tv\_sec)/SUPPLY\_INTERVAL;

next\_bcast.tv\_sec += n\*SUPPLY\_INTERVAL;

continue;

}

if (timercmp(&t2, &wtime, <))

wtime = t2;

}

/\* If we need a flash update, either do it now or

\* set the delay to end when it is time.

\*

\* If we are within MIN\_WAITTIME seconds of a full update,

\* do not bother.

\*/

if (need\_flash

&& supplier

&& no\_flash.tv\_sec+MIN\_WAITTIME < next\_bcast.tv\_sec) {

/\* accurate to the millisecond \*/

if (!timercmp(&no\_flash, &now, >))

rip\_bcast(1);

timevalsub(&t2, &no\_flash, &now);

if (timercmp(&t2, &wtime, <))

wtime = t2;

}

/\* trigger the main aging timer.

\*/

timevalsub(&t2, &age\_timer, &now);

if (t2.tv\_sec <= 0) {

age(0);

continue;

}

if (timercmp(&t2, &wtime, <))

wtime = t2;

/\* update the kernel routing table

\*/

timevalsub(&t2, &need\_kern, &now);

if (t2.tv\_sec <= 0) {

age(0);

continue;

}

if (timercmp(&t2, &wtime, <))

wtime = t2;

/\* take care of router discovery,

\* but do it to the millisecond

\*/

if (!timercmp(&rdisc\_timer, &now, >)) {

rdisc\_age(0);

continue;

}

timevalsub(&t2, &rdisc\_timer, &now);

if (timercmp(&t2, &wtime, <))

wtime = t2;

/\* wait for input or a timer to expire.

\*/

trace\_flush();

ibits = fdbits;

n = select(sock\_max, &ibits, 0, 0, &wtime);

if (n <= 0) {

if (n < 0 && errno != EINTR && errno != EAGAIN)

BADERR(1,"select");

continue;

}

if (FD\_ISSET(rt\_sock, &ibits)) {

read\_rt();

n--;

}

if (rdisc\_sock >= 0 && FD\_ISSET(rdisc\_sock, &ibits)) {

read\_d();

n--;

}

if (rip\_sock >= 0 && FD\_ISSET(rip\_sock, &ibits)) {

read\_rip(rip\_sock, 0);

n--;

}

for (ifp = ifnet; n > 0 && 0 != ifp; ifp = ifp->int\_next) {

if (ifp->int\_rip\_sock >= 0

&& FD\_ISSET(ifp->int\_rip\_sock, &ibits)) {

read\_rip(ifp->int\_rip\_sock, ifp);

n--;

}

}

}

}

/\* ARGSUSED \*/

void

sigalrm(int s)

{

/\* Historically, SIGALRM would cause the daemon to check for

\* new and broken interfaces.

\*/

ifinit\_timer.tv\_sec = now.tv\_sec;

trace\_act("SIGALRM");

}

/\* watch for fatal signals \*/

void

sigterm(int sig)

{

stopint = sig;

(void)signal(sig, SIG\_DFL); /\* catch it only once \*/

}

void

fix\_select(void)

{

struct interface \*ifp;

FD\_ZERO(&fdbits);

sock\_max = 0;

FD\_SET(rt\_sock, &fdbits);

if (sock\_max <= rt\_sock)

sock\_max = rt\_sock+1;

if (rip\_sock >= 0) {

FD\_SET(rip\_sock, &fdbits);

if (sock\_max <= rip\_sock)

sock\_max = rip\_sock+1;

}

for (ifp = ifnet; 0 != ifp; ifp = ifp->int\_next) {

if (ifp->int\_rip\_sock >= 0) {

FD\_SET(ifp->int\_rip\_sock, &fdbits);

if (sock\_max <= ifp->int\_rip\_sock)

sock\_max = ifp->int\_rip\_sock+1;

}

}

if (rdisc\_sock >= 0) {

FD\_SET(rdisc\_sock, &fdbits);

if (sock\_max <= rdisc\_sock)

sock\_max = rdisc\_sock+1;

}

}

void

fix\_sock(int sock,

char \*name)

{

int on;

#define MIN\_SOCKBUF (4\*1024)

static int rbuf;

if (fcntl(sock, F\_SETFL, O\_NONBLOCK) == -1)

logbad(1, "fcntl(%s) O\_NONBLOCK: %s",

name, strerror(errno));

on = 1;

if (setsockopt(sock, SOL\_SOCKET,SO\_BROADCAST, &on,sizeof(on)) < 0)

msglog("setsockopt(%s,SO\_BROADCAST): %s",

name, strerror(errno));

#ifdef USE\_PASSIFNAME

on = 1;

if (setsockopt(sock, SOL\_SOCKET, SO\_PASSIFNAME, &on,sizeof(on)) < 0)

msglog("setsockopt(%s,SO\_PASSIFNAME): %s",

name, strerror(errno));

#endif

if (rbuf >= MIN\_SOCKBUF) {

if (setsockopt(sock, SOL\_SOCKET, SO\_RCVBUF,

&rbuf, sizeof(rbuf)) < 0)

msglog("setsockopt(%s,SO\_RCVBUF=%d): %s",

name, rbuf, strerror(errno));

} else {

for (rbuf = 60\*1024; ; rbuf -= 4096) {

if (setsockopt(sock, SOL\_SOCKET, SO\_RCVBUF,

&rbuf, sizeof(rbuf)) == 0) {

trace\_act("RCVBUF=%d", rbuf);

break;

}

if (rbuf < MIN\_SOCKBUF) {

msglog("setsockopt(%s,SO\_RCVBUF = %d): %s",

name, rbuf, strerror(errno));

break;

}

}

}

}

/\* get a rip socket

\*/

static int /\* <0 or file descriptor \*/

get\_rip\_sock(naddr addr,

int serious) /\* 1=failure to bind is serious \*/

{

struct sockaddr\_in sin;

unsigned char ttl;

int s;

if ((s = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

BADERR(1,"rip\_sock = socket()");

bzero(&sin,sizeof(sin));

#ifdef \_HAVE\_SIN\_LEN

sin.sin\_len = sizeof(sin);

#endif

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(RIP\_PORT);

sin.sin\_addr.s\_addr = addr;

if (bind(s, (struct sockaddr \*)&sin,sizeof(sin)) < 0) {

if (serious)

BADERR(errno != EADDRINUSE, "bind(rip\_sock)");

return -1;

}

fix\_sock(s,"rip\_sock");

ttl = 1;

if (setsockopt(s, IPPROTO\_IP, IP\_MULTICAST\_TTL,

&ttl, sizeof(ttl)) < 0)

DBGERR(1,"rip\_sock setsockopt(IP\_MULTICAST\_TTL)");

return s;

}

/\* turn off main RIP socket \*/

void

rip\_off(void)

{

struct interface \*ifp;

register naddr addr;

if (rip\_sock >= 0 && !mhome) {

trace\_act("turn off RIP");

(void)close(rip\_sock);

rip\_sock = -1;

/\* get non-broadcast sockets to listen to queries.

\*/

for (ifp = ifnet; ifp != 0; ifp = ifp->int\_next) {

if (ifp->int\_state & IS\_REMOTE)

continue;

if (ifp->int\_rip\_sock < 0) {

addr = ((ifp->int\_if\_flags & IFF\_POINTOPOINT)

? ifp->int\_dstaddr

: ifp->int\_addr);

ifp->int\_rip\_sock = get\_rip\_sock(addr, 0);

}

}

fix\_select();

age(0);

}

}

/\* turn on RIP multicast input via an interface

\*/

static void

rip\_mcast\_on(struct interface \*ifp)

{

struct ip\_mreq m;

if (!IS\_RIP\_IN\_OFF(ifp->int\_state)

&& (ifp->int\_if\_flags & IFF\_MULTICAST)

#ifdef MCAST\_PPP\_BUG

&& !(ifp->int\_if\_flags & IFF\_POINTOPOINT)

#endif

&& !(ifp->int\_state & IS\_ALIAS)) {

m.imr\_multiaddr.s\_addr = htonl(INADDR\_RIP\_GROUP);

m.imr\_interface.s\_addr = ((ifp->int\_if\_flags & IFF\_POINTOPOINT)

? ifp->int\_dstaddr

: ifp->int\_addr);

if (setsockopt(rip\_sock,IPPROTO\_IP, IP\_ADD\_MEMBERSHIP,

&m, sizeof(m)) < 0)

LOGERR("setsockopt(IP\_ADD\_MEMBERSHIP RIP)");

}

}

/\* Prepare socket used for RIP.

\*/

void

rip\_on(struct interface \*ifp)

{

/\* If the main RIP socket is already alive, only start receiving

\* multicasts for this interface.

\*/

if (rip\_sock >= 0) {

if (ifp != 0)

rip\_mcast\_on(ifp);

return;

}

/\* If the main RIP socket is off and it makes sense to turn it on,

\* then turn it on for all of the interfaces.

\*/

if (rip\_interfaces > 0 && !rdisc\_ok) {

trace\_act("turn on RIP");

/\* Close all of the query sockets so that we can open

\* the main socket. SO\_REUSEPORT is not a solution,

\* since that would let two daemons bind to the broadcast

\* socket.

\*/

for (ifp = ifnet; ifp != 0; ifp = ifp->int\_next) {

if (ifp->int\_rip\_sock >= 0) {

(void)close(ifp->int\_rip\_sock);

ifp->int\_rip\_sock = -1;

}

}

rip\_sock = get\_rip\_sock(INADDR\_ANY, 1);

rip\_sock\_mcast = 0;

/\* Do not advertise anything until we have heard something

\*/

if (next\_bcast.tv\_sec < now.tv\_sec+MIN\_WAITTIME)

next\_bcast.tv\_sec = now.tv\_sec+MIN\_WAITTIME;

for (ifp = ifnet; ifp != 0; ifp = ifp->int\_next) {

ifp->int\_query\_time = NEVER;

rip\_mcast\_on(ifp);

}

ifinit\_timer.tv\_sec = now.tv\_sec;

} else if (ifp != 0

&& !(ifp->int\_state & IS\_REMOTE)

&& ifp->int\_rip\_sock < 0) {

/\* RIP is off, so ensure there are sockets on which

\* to listen for queries.

\*/

ifp->int\_rip\_sock = get\_rip\_sock(ifp->int\_addr, 0);

}

fix\_select();

}

/\* die if malloc(3) fails

\*/

void \*

rtmalloc(size\_t size,

char \*msg)

{

void \*p = malloc(size);

if (p == 0)

logbad(1,"malloc() failed in %s", msg);

return p;

}

/\* get a random instant in an interval

\*/

void

intvl\_random(struct timeval \*tp, /\* put value here \*/

u\_long lo, /\* value is after this second \*/

u\_long hi) /\* and before this \*/

{

tp->tv\_sec = (time\_t)(hi == lo

? lo

: (lo + random() % ((hi - lo))));

tp->tv\_usec = random() % 1000000;

}

void

timevaladd(struct timeval \*t1,

struct timeval \*t2)

{

t1->tv\_sec += t2->tv\_sec;

if ((t1->tv\_usec += t2->tv\_usec) > 1000000) {

t1->tv\_sec++;

t1->tv\_usec -= 1000000;

}

}

/\* t1 = t2 - t3

\*/

static void

timevalsub(struct timeval \*t1,

struct timeval \*t2,

struct timeval \*t3)

{

t1->tv\_sec = t2->tv\_sec - t3->tv\_sec;

if ((t1->tv\_usec = t2->tv\_usec - t3->tv\_usec) < 0) {

t1->tv\_sec--;

t1->tv\_usec += 1000000;

}

}

/\* put a message into the system log

\*/

void

msglog(char \*p, ...)

{

va\_list args;

trace\_flush();

va\_start(args, p);

vsyslog(LOG\_ERR, p, args);

if (ftrace != 0) {

if (ftrace == stdout)

(void)fputs("routed: ", ftrace);

(void)vfprintf(ftrace, p, args);

(void)fputc('\n', ftrace);

}

}

/\* Put a message about a bad system into the system log if

\* we have not complained about it recently.

\*

\* It is desirable to complain about all bad systems, but not too often.

\* In the worst case, it is not practical to keep track of all bad systems.

\* For example, there can be many systems with the wrong password.

\*/

void

msglim(struct msg\_limit \*lim, naddr addr, char \*p, ...)

{

va\_list args;

int i;

struct msg\_sub \*ms1, \*ms;

char \*p1;

va\_start(args, p);

/\* look for the oldest slot in the table

\* or the slot for the bad router.

\*/

ms = ms1 = lim->subs;

for (i = MSG\_SUBJECT\_N; ; i--, ms1++) {

if (i == 0) {

/\* Reuse a slot at most once every 10 minutes.

\*/

if (lim->reuse > now.tv\_sec) {

ms = 0;

} else {

ms = ms1;

lim->reuse = now.tv\_sec + 10\*60;

}

break;

}

if (ms->addr == addr) {

/\* Repeat a complaint about a given system at

\* most once an hour.

\*/

if (ms->until > now.tv\_sec)

ms = 0;

break;

}

if (ms->until < ms1->until)

ms = ms1;

}

if (ms != 0) {

ms->addr = addr;

ms->until = now.tv\_sec + 60\*60; /\* 60 minutes \*/

trace\_flush();

for (p1 = p; \*p1 == ' '; p1++)

continue;

vsyslog(LOG\_ERR, p1, args);

}

/\* always display the message if tracing \*/

if (ftrace != 0) {

(void)vfprintf(ftrace, p, args);

(void)fputc('\n', ftrace);

}

}

void

logbad(int dump, char \*p, ...)

{

va\_list args;

trace\_flush();

va\_start(args, p);

vsyslog(LOG\_ERR, p, args);

(void)fputs("routed: ", stderr);

(void)vfprintf(stderr, p, args);

(void)fputs("; giving up\n",stderr);

(void)fflush(stderr);

if (dump)

abort();

exit(1);

}