
Cơ sở mạng thông tin

Giáo trình dành cho sinh viên đại học ngành
Điện tử - Viễn thông

Chủ biên: Nguyễn Hữu Thanh

Khoa Điện tử Viễn Thông
Trường Đại học Bách khoa Hà nội

Các từ viết tắt

Đầy đủ	Viết tắt
Cumulative distribution function	CDF
First-come-first-server	FCFS
First in first out	FIFO
Last-come-first-serve	LCFS
Last in first out	LIFO
Probability density function	pdf
Probability distribution function	PDF

Bảng đổi chiếu thuật ngữ Anh - Việt

Tiếng Anh	Tiếng Việt
Analysis	Phân tích
Arrival process	Tiến trình tới
Base station	Trạm gốc
Biominal distribution	Phân bố nhị thức
Binomial process	Tiến trình nhị thức
Birth – Death Process	Tiến trình sinh tử
Departure process	Tiến trình đi
Evaluation	Đánh giá
Expectation	Kỳ vọng
Exponential distribution	Phân bố mũ
First in first out / First-come-first-serve	Vào trước phục vụ trước
Formal description	Mô tả hình thức
Frequency function	Hàm tần suất
Gaussian distribution	Phân bố chuẩn/phân bố Gauss
Inter-arrival time	Thời gian giữa hai sự kiện tới (?)
Last in first out / Last-come-first-serve	Vào sau phục vụ trước
Load	Tải
Model	Mô hình
Modeling	Mô hình hóa
Performance	Đặc tính/chất lượng hoạt động
Probability	Xác suất
Probability density function	Hàm mật độ xác suất
Probability distribution function	Hàm phân phối xác suất
Random experiment	Phép thử ngẫu nhiên
Random event	Sự kiện ngẫu nhiên
Random variable	Biến ngẫu nhiên
Scale parameter	Tham số tỷ lệ
Server	Trạm phục vụ/Server
Service process	Tiến trình phục vụ
Shape parameter	Tham số hình dạng
Simulation	Mô phỏng

Standard deviation	Độ lệch chuẩn
Steady state	Trạng thái ổn định
Stochastic process	Tiến trình ngẫu nhiên
Traffic intensity	Mật độ lưu lượng
Transformation	Biến đổi
Uniform distribution	Phân bố đều
Utilization	Hiệu suất kênh
Validation	Kiểm định tính chính xác
Variance	Phương sai

Mục lục

Các từ viết tắt	3
Bảng đổi chiều thuật ngữ Anh - Việt	4
Mục lục	6
Mục lục hình vẽ	9
Mục lục bảng biểu	10
Chương 1 Giới thiệu	1
1.1. Mục đích của việc mô hình hóa và đánh giá đặc tính hoạt động của hệ thống	1
1.2. Các tham số, tiêu chuẩn và phương pháp đánh giá một hệ thống thông tin	3
1.2.1. Các tham số đánh giá đặc tính hoạt động của hệ thống thông tin	3
1.2.2. Các tiêu chuẩn đánh giá	3
1.2.3. Các phương pháp đánh giá	4
Chương 2 Các tiên trình ngẫu nhiên	6
2.1. Xác suất và sự kiện	6
2.1.1. Phép thử và sự kiện ngẫu nhiên	6
2.1.2. Xác suất	7
2.2. Biến ngẫu nhiên và các hàm xác suất	9
2.2.1. Biến ngẫu nhiên	9
2.2.2. Hàm mật độ và phân phối xác suất	10
2.2.3. Hàm tần suất và phân phối xác suất của biến ngẫu nhiên rời rạc	12
2.2.4. Các tham số đặc trưng của biến ngẫu nhiên	13
2.3. Các mô hình phân bố xác suất cơ bản	16
2.3.1. Phân bố Bernoulli (Bernoulli distribution)	16
2.3.2. Phân bố nhị thức (binomial distribution)	17
2.3.3. Phân bố chuẩn (Gaussian distribution)	19
2.3.4. Phân bố mũ (exponential distribution)	20
2.3.5. Phân bố Poisson (Poisson distribution)	20
2.3.6. Phân bố Gamma (Gamma distribution)	21
2.3.7. Mối liên hệ giữa phân bố mũ và phân bố Gamma	22
2.4. Tiên trình ngẫu nhiên (stochastic process)	23
2.4.1. Khái niệm và định nghĩa	23
2.4.2. Phân loại	24
2.5. Các đặc tính thống kê của tiên trình ngẫu nhiên	25
2.5.1. Các hàm quan hệ xác suất	25
2.5.2. Các trung bình thống kê	26
2.5.3. Tính dừng	27
2.6. Các tiên trình ngẫu nhiên thường gặp	30
2.6.1. Tiên trình đếm	30
2.6.2. Tiên trình Poisson	31
2.7. Kết luận	31
Chương 3 Hệ thống hàng đợi	32
3.1. Giới thiệu	32
3.2. Mô hình hàng đợi – Ký hiệu Kendall	32
3.2.1. Mô hình hàng đợi đơn	32
3.2.2. Ký hiệu Kendall	33
3.2.3. Các tham số quan trọng để đánh giá đặc tính của hệ thống hàng đợi	35

3.3. Hệ thống hàng đợi ở trạng thái ổn định – Định lý Little	36
3.3.1. Hệ thống hàng đợi ổn định	36
3.3.2. Định lý Little	38
3.3.3. Một số đặc tính khác của hệ thống đồng, hoạt động ở trạng thái ổn định	40
3.4. Hàng đợi M/M/1/1	41
3.5. Hàng đợi M/M/1/∞	44
3.5.1. Xác suất có n yêu cầu trong hệ thống	44
3.5.2. Tính toán các tham số hiệu năng	46
3.6. Tiến trình sinh – tử (Birth – Death Process)	49
3.7. Hàng đợi M/M/1/K	49
3.7.1. Xác suất có n yêu cầu trong hệ thống	50
3.7.2. Tính toán các tham số hiệu năng	50
3.8. Hàng đợi M/M/c/∞	54
3.8.1. Xác suất có n yêu cầu trong hệ thống	55
Chương 4 Mạng hàng đợi	56
4.1. Mạng nối tiếp	56
Chương 5 Định tuyến trong mạng thông tin	57
5.1. Yêu cầu về định tuyến trong mạng thông tin	57
5.1.1. Vai trò của định tuyến trong mạng thông tin	57
5.1.2. Các khái niệm trong lý thuyết graph	57
5.2. Các mô hình định tuyến quảng bá (broadcast routing)	59
5.2.1. Lan tràn gói (flooding)	59
5.2.2. Định tuyến bước ngẫu nhiên (random walk)	60
5.2.3. Định tuyến khoai tây nóng (hot potato)	60
5.2.4. Định tuyến nguồn (source routing) và mô hình cây (spanning tree)	61
5.2.5. Duyệt cây	61
5.3. Các mô hình định tuyến thông dụng	82
5.3.1. Định tuyến ngắn nhất (Shortest path Routing)	82
5.4. Bài tập (Pending)	105
Chương 6 Điều khiển luồng và chống tắc nghẽn	106
6.1. Tổng quan	106
6.1.1. Mở đầu	106
6.1.2. Khái niệm điều khiển luồng	109
6.1.3. Khái niệm chống tắc nghẽn	110
6.1.4. Nhiệm vụ chủ yếu của điều khiển luồng và chống tắc nghẽn	110
6.1.5. Phân loại điều khiển luồng và tránh tắc nghẽn	111
6.2. Tính công bằng	112
6.2.1. Định nghĩa	112
6.2.2. Tính công bằng về mặt băng truyền	112
6.2.3. Tính công bằng về mặt bộ đệm	112
6.2.4. Cơ chế phát lại ARQ	114
6.2.5. Stop-and-Wait ARQ	115
6.2.6. Go-back-N ARQ	121
6.2.7. Selective repeat ARQ	128
6.3. Điều khiển luồng và tránh tắc nghẽn theo phương pháp cửa sổ	130
6.3.1. Điều khiển luồng theo cửa sổ (Window Flow Control)	131
6.3.2. Điều khiển tắc nghẽn sử dụng cửa sổ thích ứng (adaptive window)	136
6.4. Điều khiển luồng và chống tắc nghẽn dựa trên băng thông (rate-based flow control)	142
6.4.1. Khái niệm	142
6.4.2. Điều khiển băng thông theo thuật toán gáo rò (leaky bucket)	143
6.4.3. Thuật toán GPS (pending)	147

6.5. Bài tập (Pending)	148
<i>Chương 7 Kỹ thuật mô phỏng</i>	149
 7.1. Giới thiệu	149
 7.2. Mô phỏng dựa trên các sự kiện rời rạc và các công cụ	149
7.2.1. Phương pháp mô phỏng dựa trên sự kiện rời rạc	149
7.2.2. Các công cụ mô phỏng thông dụng dựa trên sự kiện rời rạc	152
 7.3. Công cụ mô phỏng mạng NS2	153
7.3.1. Cấu trúc	153
7.3.2. Các tiện ích trong NS hỗ trợ cho mô phỏng mạng [Pending]	155
7.3.3. Thí dụ (Pending)	155
 7.4. Kết luận (Pending)	155
 7.5. Bài tập (Pending)	156
Tài liệu tham khảo	157
Phụ lục 1	158

Mục lục hình vẽ

Hình 1-1. Các bước cơ bản trong mô hình hóa và đánh giá đặc tính hệ thống	3
Hình 2-1. Hàm mật độ xác suất.....	10
Hình 2-2.Hàm phân phối xác suất.....	11
Hình 2-3: Kết quả Thí dụ 2-5	13
Hình 3-1. Hệ thống hàng đợi với N trạm phục vụ, hàng đợi có độ lớn K	33
Hình 3-2. Round robin	34
Hình 3-3. Mạng đóng.....	36
Hình 3-4. Quan hệ giữa số yêu cầu tới và số yêu cầu được phục vụ	39
Hình 3-5.Hệ thống hàng đợi được quan sát với hai ranh giới khác nhau: (a) Hàng đợi và các trạm phục vụ; (b) Hàng đợi.....	41
Hình 3-6. Hàng đợi M/M/1	42
Hình 3-7. Khoảng thời gian xét.....	42
Hình 3-8. Hàng đợi M/M/1/ ∞	44
Hình 3-9. Cân bằng xác suất tại trạng thái i. Tiến trình sinh – tử	49
Hình 3-10. Hệ thống M/M/1/K.....	49
Hình 3-11. Hệ thống hàng đợi M/M/1/K	50
Hình 3-12. Hệ thống M/M/1/K trên quan điểm hệ thống đóng..	53
Hình 3-13 Hàng đợi M/M/c/ ∞	54
Hình 5-1. Hàng chờ bên trong router	61
Hình 5-2. Duyệt cây.....	62
Hình 5-3. Các thành phần	66
Hình 5-4. Phép tính Minimum Spanning Tree (MST).....	74

Mục lục bảng biểu

Bảng 2-1. Điểm tương ứng với kết quả tung xúc xắc 9

Bảng 2-2: Kết quả Thí dụ 2-5..... 13

Chương 1 Giới thiệu

1.1. Mục đích của việc mô hình hóa và đánh giá đặc tính hoạt động của hệ thống

Trong thực tế, khi khảo sát các hệ phục vụ nói chung, các hệ thống máy tính và viễn thông nói riêng, hoặc khi nghiên cứu đưa ra các cơ chế hoạt động mới trong các hệ thống này, một yêu cầu hàng đầu là phải khảo sát các đặc tính và hiệu năng hoạt động của các hệ thống, cơ chế đó.

Thí dụ 1-1:

Một nhà cung cấp dịch vụ điện thoại di động GSM muốn mở rộng vùng phủ sóng của mình. Với các tham số đầu vào cho trước bao gồm:

- Lưu lượng đầu vào λ , được tính bằng số yêu cầu kết nối trong một đơn vị thời gian. Tham số này được khảo sát và đo đạc thực tế tại vùng cần mở rộng.
- Thời gian trung bình của một cuộc gọi di động μ . Tham số này đã biết trước dựa trên các dữ liệu thống kê của nhà cung cấp dịch vụ.
- Tải tối đa u của một trạm gốc (base station), chính là số cuộc gọi tối đa mà trạm gốc có thể cho phép tại một thời điểm.
- Yêu cầu xác suất từ chối dịch vụ tối đa p . Đây là xác suất một yêu cầu kết nối bị từ chối do trạm gốc không đủ tài nguyên cung cấp cho cuộc gọi đó.

Từ các yêu cầu đầu vào, nhà cung cấp cần phải tính toán có bao nhiêu trạm gốc cần phải lắp đặt mới tại vùng đó, để xác suất từ chối dịch vụ nhỏ hơn p .

Thí dụ 1-2:

Mạng Ethernet có N máy tính, tổng lưu lượng đầu vào đo được là λ (Mbit/s). Kênh truyền có dung lượng là C (Mbit/s). Phải tính toán hiệu suất hoạt động của kênh truyền (tính bằng % của lưu dung lượng C), trễ trung bình (tính bằng s) của một gói tin khi được truyền từ nguồn tới đích.

Tuy nhiên, quá trình phân tích một hệ thống thực thường thường đối khó khăn và tốn kém. Để khẳng định tính chất của một hệ thống về hiệu năng hoạt động, tính kinh tế .v.v., thông thường người ta thường sử dụng các mô hình để miêu tả các hệ thống đó.

Định nghĩa 1-1: Mô hình (model) thông thường là sự đơn giản hóa một hệ thống thực bằng cách miêu tả các hoạt động và trạng thái của hệ thống đó cùng với các điều kiện khởi đầu và điều kiện bờ cho trước.

Mục đích của việc mô hình hóa là nó cho phép đánh giá đặc tính, từ đó cải thiện chất lượng hoạt động của một hệ thống. Để các thí nghiệm với mô hình cho ra kết quả chính xác và đáng tin cậy, các dữ liệu đầu vào của mô hình phải phù hợp với hệ thống thực tế.

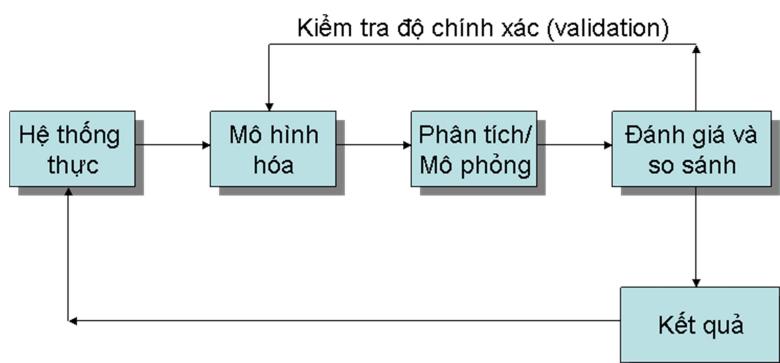
Các bước trong quá trình mô hình hóa và đánh giá đặc tính hoạt động của một hệ thống bao gồm:

- **Mô hình hóa** (modeling): Được hiểu là quá trình trừu tượng hóa và đơn giản hóa một hệ thống thực bằng cách bỏ qua các yếu tố không quan trọng và chỉ tập trung vào một tập hợp hữu hạn các thông số đáng chú ý được sử dụng để miêu tả hệ thống. Một điều khó khăn ở bước này là xác định các yếu tố có thể bỏ qua và các thông số bắt buộc phải được xem xét. Vì vậy độ chính xác của một mô hình phụ thuộc rất nhiều vào bước này.

Thông thường có hai phương pháp mô hình hóa là **mô hình toán học** và **mô hình mô phỏng** (xem 1.2.3).

- **Phân tích** (analysis): Được hiểu là quá trình tìm hiểu, khám phá các đặc điểm hoặc chức năng của hệ thống. Trong bước này, đặc điểm hoặc phản ứng của một hệ thống sẽ được nghiên cứu tương ứng với các thông số đầu vào cho trước.
- **Đánh giá và so sánh** (evaluation and comparison): Đặc tính của một hệ thống với các thông số đầu vào khác nhau sẽ được kiểm tra và so sánh, thông qua đó có thể đánh giá tính chính xác của mô hình (validation).
- **Đưa kết quả đánh giá về hệ thống thực**: Các kết quả được đưa ra từ các bước trên được đưa trở về phục vụ cho hệ thống thực (thí dụ như để cải thiện chất lượng hoạt động .v.v.).

Hình 1-1 mô tả quá trình đánh giá đặc tính hoạt động hệ thống.



Hình 1-1. Các bước cơ bản trong mô hình hóa và đánh giá đặc tính hệ thống

1.2. Các tham số, tiêu chuẩn và phương pháp đánh giá một hệ thống thông tin

1.2.1. Các tham số đánh giá đặc tính hoạt động của hệ thống thông tin

Để đánh giá đặc tính hoạt động của một hệ thống, cần phải lựa chọn các **tham số của mô hình**. Các tham số này có thể được phân loại như sau:

Các tham số liên quan đến thời gian:

- Thời gian hệ thống thực hiện một yêu cầu
- Thời gian đáp ứng của hệ thống.
- Thời gian một yêu cầu lưu lại trong hệ thống .v.v.

Các tham số liên quan đến không gian:

- Độ lớn của hàng đợi hệ thống (độ lớn bộ đệm .v.v.)
- Yêu cầu về bộ nhớ cho một chương trình .v.v.

Các tham số khác:

- Thông lượng
- Giá thành .v.v.

1.2.2. Các tiêu chuẩn đánh giá

Như chúng ta đã biết, trong quá trình mô hình hóa người ta chỉ giới hạn xem xét một số thông số quan trọng của hệ thống. Vì vậy kết luận về đặc tính hoạt động của một hệ thống nào đó bao giờ cũng có dạng:

“Hệ thống A tốt hơn hệ thống B về mặt C”

Việc kết luận chung chung theo kiểu “hệ thống A tốt hơn hệ thống B” thông thường không chính xác do khi xét theo một tiêu chuẩn đánh giá nào đó, một hệ thống có thể tối ưu nhưng khi xét theo một tiêu chuẩn khác, hệ thống đó lại có những nhược điểm đáng kể.

Thí dụ 1-3:

Sau đây là một thí dụ về việc chọn tham số mô hình, quy tắc phục vụ và các tiêu chuẩn đánh giá:

Tham số mô hình:

- Tham số lưu lượng của yêu cầu đến một hệ thống.
- Tham số đặc trưng cho thời gian phục vụ một yêu cầu.
- Số trạm phục vụ các yêu cầu

Quy tắc phục vụ:

- FIFO, LIFO
- Hàng đợi có ưu tiên

Các tiêu chuẩn đánh giá:

- Thời gian lưu lại hệ thống trung bình của một yêu cầu.
- Thời gian chờ đợi của một yêu cầu trong hàng đợi.
- Thông lượng của hệ thống.
- Tải của hệ thống.

1.2.3. Các phương pháp đánh giá

Phương pháp phân tích toán học (mathematical analysis)

Phương pháp phân tích toán học thực hiện các bước sau:

- Mô tả hình thức (formal description) một hệ thống.
- Định nghĩa các mối quan hệ trong hệ thống, mô tả chúng bằng các công thức, mô hình toán học.
- Tính toán dựa vào mô hình toán học vừa lập.

Trong quyển sách này chúng tôi chủ yếu tập trung vào phương pháp này.

Phương pháp xấp xỉ (approximation method)

Trong nhiều trường hợp, phương pháp phân tích toán học quá phức tạp để có thể mô tả một hệ thống. Phương pháp xấp xỉ cơ thể áp dụng trong những trường hợp này.

Phương pháp mô phỏng (simulative techniques)

Mô phỏng miêu tả một quá trình xảy ra trong thực tế thông qua các chương trình máy tính. Mô phỏng sử dụng cở sở xác suất thống kê để đánh giá đặc tính hoạt động của một hệ thống từ các kết quả thí nghiệm thu được, ví dụ như giá trị kỳ vọng, phương sai, các phân bố ngẫu nhiên của kết quả .v.v.

Mô phỏng có ưu điểm là việc xây dựng, phân tích và đánh giá dễ dàng hơn so với phương pháp toán học. Trong nhiều trường hợp, mô phỏng là phương pháp khả thi nhất về mặt thời gian và tài chính (không phải mua một hệ thống thực) để khảo sát và đánh giá đặc tính một hệ thống.

Phương pháp mô phỏng có những nhược điểm chính như: (1) thời gian chạy chương trình khá lớn nếu mô phỏng các hệ thống phức tạp. Do đó thông thường các hệ thống thật cũng phải đơn giản hóa đi khá nhiều khi chuyển sang mô hình mô phỏng; (2) độ chính xác kết quả của mô hình mô phỏng tương đối khó kiểm chứng. Một trong các phương pháp để kiểm tra độ chính xác của mô hình là chạy chương trình mô phỏng với các tham số đầu vào mà giá trị đầu ra đã được biết

trước, sau đó so sánh các kết quả mô phỏng so với kết quả đo đạc được trong thực tế.

Các công cụ mô phỏng mạng thông dụng được sử dụng hiện nay là *NS-2* (network simulator version 2) [5], *OMNET++* [10], *OPNET* .v.v. [8]. Các kỹ thuật mô phỏng sẽ được trình bày cụ thể trong Chương 7.

Phương pháp đo đạc (measurement techniques)

Đo đạc được sử dụng để đo các thông số cần thí nghiệm trong các hệ thống thực (như thông lượng, băng thông, trễ, tuyến đường mà một gói IP đi qua .v.v.).

Đo đạc được sử dụng để hỗ trợ cho phương pháp phân tích toán học và phương pháp mô phỏng; các tham số đầu vào sử dụng trong các mô hình toán học và mô phỏng đều được đo đạc trong thực tế. Mặt khác, đo đạc cũng được sử dụng để kiểm tra độ chính xác của một mô hình so với hệ thống thực tế.

Các công cụ đo đạc hay được sử dụng trong thực tế bao gồm: *Ethereal*, *tcpdump*, *net-snmp*, *netperf*, *dbs* .v.v. [2].

Phương pháp kết hợp (hybrid techniques)

Phương pháp này kết hợp cả phương pháp phân tích toán học và phương pháp mô phỏng ở trên. Trong phương pháp này, một hệ thống sẽ được phân tách thành các khối chức năng. Đối với từng khối, người ta có thể sử dụng phương pháp mô phỏng hoặc phân tích toán học.

Chương 2 Các tiến trình ngẫu nhiên

2.1. Xác suất và sự kiện

Trong mục này, các khái niệm cơ bản trong môn xác suất thông kê sẽ được trình bày.

2.1.1. Phép thử và sự kiện ngẫu nhiên

Định nghĩa 2-1: Một phép thử ngẫu nhiên (random experiment) là quá trình thử hay quá trình quan sát có thể được lặp đi lặp lại nhiều lần dưới cùng một điều kiện giống nhau. Kết quả của một lần thử không thể tiên đoán trước, độc lập với các phép thử trước đó và có phân bố đồng nhất.

Ví dụ:

Hành vi: ngồi quan sát giao thông trên đường Đại Cồ Việt trong 1h là một phép thử ngẫu nhiên. Kết quả của một phép thử là số người trên đường trong 1 giờ quan sát thấy

Chú ý rằng tuy kết quả của phép thử ngẫu nhiên không thể tiên đoán trước, nhưng các kết quả này phải nằm trong một tập giá trị xác định. Như vậy, nếu gọi e là kết quả của phép thử ngẫu nhiên, Ω là tập hợp các giá trị của phép thử, ta có:

$$e \in \Omega; \quad (\text{PT 2-1})$$

Thí dụ 2-1: Việc tung đồng xu là một phép thử ngẫu nhiên

$$e \in \Omega = \{\text{sấp, ngửa}\}, e \text{ lấy 2 giá trị}$$

Định nghĩa 2-2: Sự kiện ngẫu nhiên (random event) là kết quả của một phép thử ngẫu nhiên. Kết quả này nằm trong một tập các kết quả đã được xác định.

2.1.2. Xác suất

Khái niệm xác suất

Định nghĩa 2-3: Xác suất được định nghĩa là tần suất quan hệ (relative frequency) của việc xuất hiện một sự kiện ngẫu nhiên.

Nếu gọi A là sự kiện ngẫu nhiên; sự kiện A xuất hiện n_A lần trong một số lượng lớn các sự kiện n , xác suất của sự kiện A sẽ được định nghĩa như sau:

$$P(A) \equiv \frac{n_A}{n}; \quad (\text{PT 2-2})$$

Một số tính chất của xác suất

- Có thể thấy nếu A không bao giờ xuất hiện $P(A) = 0$. Mặt khác, nếu A luôn xuất hiện $P(A) = 1$. Do đó:

$$0 \leq P(A) \leq 1; \quad (\text{PT 2-3})$$

Ví dụ: nếu quan sát mặt trời mọc vào buổi sáng. Kết quả của phép thử này là sự kiện A = ngày mai trời lại sang $\rightarrow P(A) = 1$

- Nếu gọi $P(A \cup B)$ là xác suất xuất hiện của sự kiện A **hoặc** sự kiện B . Ta có, trong trường hợp biến A và B tương quan:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B); \quad (\text{PT 2-4})$$

Trong đó $P(A \cap B)$ là xác suất xuất hiện sự kiện A **và** sự kiện B .
Trong trường hợp A và B độc lập:

$$P(A \cup B) = P(A) + P(B); \quad (\text{PT 2-5})$$

- Nếu gọi $P(A/B)$ là xác suất có điều kiện (A xuất hiện với điều kiện B xuất hiện). Trong trường hợp A và B tương quan:

$$P(A \cap B) = P(A) \cdot P(B/A) = P(B) \cdot P(A/B); \quad (\text{PT 2-6})$$

Nếu có

$$P(A_i | B) = \frac{P(A_i)P(B|A_i)}{\sum_{j=1}^K P(A_j)P(B|A_j)} \quad (\text{PT 2-7})$$

Công thức trên được gọi là công thức Bayes. Mặt khác, nếu A và B là hai biến độc lập:

$$P(A \cap B) = P(A) \cdot P(B); \quad (\text{PT 2-8})$$

Bài tập 2.1.2 a/

Giả sử có 10 lá bài, 5 trong số đó màu đỏ, 5 màu xanh được đặt ngẫu nhiên với 10 phong bì trong đó có 5 phong bì đỏ, 5 phong bì xanh. Tìm xác suất để có chính xác chỉ 2 phong bì chứa lá bài cùng màu.

Bài tập 2.1.2 b/

Giả sử 2 máy 1 và 2 trong nhà máy hoạt động độc lập. A là sự kiện máy 1 không hoạt động trong 8h. B là sự kiện máy 2 không hoạt động trong 8h. cho $P(A) = 1/3$ và $P(B) = 1/4$. Tính xác suất để ít nhất 1 trong 2 máy trở nên không hoạt động trong thời gian đã đưa.

Giải:

Ta có

$$P(AB) = P(A) P(B) = 1/3 \times 1/4 = 1/12$$

Do đó xác suất mà ít nhất 1 máy không hoạt động

$$P(A \cup B) = P(A) + P(B) - P(AB) = 1/3 + 1/4 - 1/12 = 1/2$$

Bài tập 2.1.2 c/

áp dụng công thức PT-2.7

Giả sử có 1 chiếc hộp có 1 đồng xu có hai mặt hình hoa sen/đầu người và 1 đồng xu chỉ có 1 mặt hình hoa sen. Giả sử lựa chọn ngẫu nhiên 1 đồng xu và tung đồng xu đó lên 3 lần, cả 3 lần ta đều nhìn thấy hình hoa sen. Tính xác suất để đồng xu đó là đồng xu có hai mặt

Giải:

Gọi F kí hiệu cho đồng xu có hai mặt

B kí hiệu cho đồng xu có 1 mặt

H : cho hoa sen

D: cho đầu người

Tập sự kiện ngẫu nhiên

$$\Omega = \{F, B\} \times \{H, D\}^3$$

$$\Omega = \{c.x_1.x_2.x_3 : c \in \{F, B\}; x_i \in \{H, D\} \text{ với mỗi } i\}$$

Xác suất xuất hiện đồng F và có mặt hoa sen trong 3 lần

$$P[F, HHH] = P(F).P(HHH) = 1/2 \times (1/2)^3 = 1/16$$

Xác suất xuất hiện đồng B và cả 3 lần là mặt hoa sen:

$$P[B, HHH] = P(B).P(HHH)$$

Xác suất sự kiện đồng xu nhặt được trong phép thử trên là đồng xu 2 mặt F với điều kiện 3 mặt hoa sen HHH là một phần trong xác suất tổng có cùng điều kiện 3 mặt hoa sen HHH.

để tính xác suất của sự kiện nhặt được đồng 2 mặt dựa trên điều kiện cho trước là 3 lần nhặt lén thấy mặt hoa sen, áp dụng công thức PT-2.7, ta có:

$$P(F | HHH) = \frac{P(F)P(HHH || F)}{P(F)P(HHH || F) + P(B)P(HHH || B)} = \frac{(1/2)(1/2)^3}{(1/2)(1/2)^3 + (1/2).1} = \frac{1}{9}$$

2.2. Biến ngẫu nhiên và các hàm xác suất

2.2.1. Biến ngẫu nhiên

Định nghĩa 2-4: Biến ngẫu nhiên được định nghĩa là đại lượng biến thiên phụ thuộc vào kết cục của một phép thử ngẫu nhiên nào đó.

Như vậy, nếu gọi X là một biến ngẫu nhiên phụ thuộc vào sự kiện ngẫu nhiên e , Ω là tập hợp các giá trị của phép thử, ta có:

$$X \equiv X(e), e \in \Omega; \quad (\text{PT 2-9})$$

Thí dụ 2-2: Trò chơi tung xúc xắc tính điểm với luật chơi sau:

Bảng 2-1. Điểm tương ứng với kết quả tung xúc xắc

Kết quả tung	Điểm
1	200
2	500
3	600
4	1000
5	1200
6	1500

Trên Bảng 2-1, kết quả tung chính là sự kiện ngẫu nhiên (kết quả của phép thử), điểm là biến ngẫu nhiên tương ứng với kết quả đó.

Có hai loại biến ngẫu nhiên là **biến ngẫu nhiên rời rạc** và **biến ngẫu nhiên liên tục**.

Biến ngẫu nhiên rời rạc

Định nghĩa 2-5: Biến ngẫu nhiên rời rạc (discrete random variable) là biến mà tập giá trị của nó là một tập hữu hạn hoặc vô hạn đếm được các phần tử.

Nói một cách khác, miền giá trị của một biến ngẫu nhiên rời rạc là một dãy số: $x_1, x_2, \dots, x_n, \dots$ có thể hữu hạn hoặc vô hạn.

Thí dụ 2-3: Điểm thi của một sinh viên (từ 1 đến 10); Số cuộc gọi của một tổng đài trong một đơn vị thời gian.

Biến ngẫu nhiên liên tục

Định nghĩa 2-6: Biến ngẫu nhiên liên tục (continuous random variable) là biến ngẫu nhiên mà tập giá trị của nó lấp kín một khoảng trên trục số

Nói một cách khác, số phần tử của tập giá trị là vô hạn, không đếm được theo lý thuyết số. Miền giá trị của một biến ngẫu nhiên liên tục sẽ là một đoạn $[a, b] \in \mathbb{R}$.

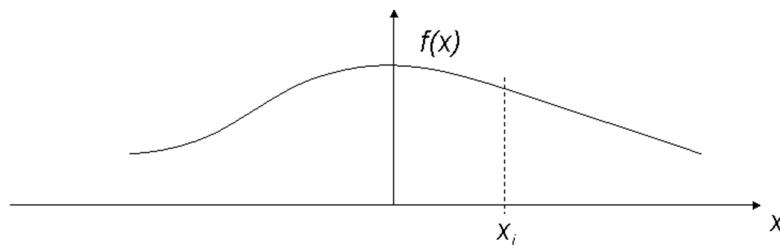
Thí dụ 2-4: Huyết áp của một bệnh nhân; tuổi thọ của một loại bóng đèn điện tử.

2.2.2. Hàm mật độ và phân phối xác suất

Hàm mật độ xác suất (probability density function - pdf)

Mật độ xác suất của một biến ngẫu nhiên x liên tục có thể hiểu là xác suất để biến ngẫu nhiên đó lấy giá trị trong miền $[x_i, x_i + dx]$. Nếu gọi $f(x)$ là hàm mật độ xác xuất của x thì:

$$f(x) \equiv p(x_i \leq x \leq x_i + dx); \quad (\text{PT 2-10})$$



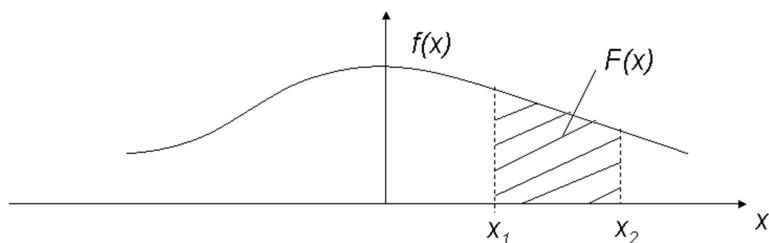
Hình 2-1. Hàm mật độ xác suất

Hàm phân phối xác suất (probability distribution function - PDF)

Hàm phân phối xác suất¹ của một biến ngẫu nhiên liên tục x là xác suất để biến ngẫu nhiên đó lấy giá trị trong miền $[x_1 \leq x \leq x_2]$. Nếu gọi $F(x)$ là hàm phân phối xác suất, ta có định nghĩa hàm phân phối xác suất như sau:

$$F(x) \equiv \int_{x_1}^{x_2} f(x)dx; \quad (\text{PT 2-11})$$

Trong đó $f(x)$ là hàm mật độ xác suất của x .



Hình 2-2.Hàm phân phối xác suất

Từ phương trình trên ta có thể thấy:

$$F(x) = \int_{-\infty}^{\infty} f(x)dx = 1; \quad (\text{PT 2-12})$$

Một số tính chất của hàm mật độ xác suất

Định lý 2-1: Hàm mật độ xác suất của tổng hai biến ngẫu nhiên độc lập sẽ là tích chập của hai hàm mật độ xác suất thành phần.

Tức là:

$$f_{X_{\sum}}(x) = \int_{-\infty}^{\infty} f_{X_i}(y)f_{X_j}(x-y)dy; \quad (\text{PT 2-13})$$

Trong đó: X_i và X_j là hai biến ngẫu nhiên, $X_{\sum} = X_i + X_j$. Hàm $f_{X_{\sum}}(x)$, $f_{X_i}(x)$ và $f_{X_j}(x)$ là các hàm mật độ xác suất tương ứng.

Chứng minh: (pending)

¹ Có hai thuật ngữ tương đương trong sách tiếng Anh biểu thị Hàm phân phối xác suất là: "Probability Distribution Function" (PDF) và "Cumulative Distribution Function" (CDF).

2.2.3. Hàm tần suất và phân phối xác suất của biến ngẫu nhiên rời rạc

Xét X là biến ngẫu nhiên rời rạc có tập các giá trị rời rạc x_1, x_2, \dots, x_K .

Hàm tần suất (frequency function) của biến ngẫu nhiên rời rạc được xác định bởi:

$$\tilde{P}_X(x_i) = P(X = x_i) \quad i = 1, 2, \dots, K; \quad (\text{PT 2-14})$$

Xem xét sự kiện $a < X \leq b$ với $a < x_1$ và $x_k \leq b < x_{k+1}$, chúng ta có:

$$\begin{cases} P(a < X \leq b) &= \tilde{P}_X(x_1) + \tilde{P}_X(x_2) + \dots + \tilde{P}_X(x_k) \\ &= \sum_{i=1}^k \tilde{P}_X(x_i) \\ P(X \leq a) &= 0 \end{cases} \quad (\text{PT 2-15})$$

Khi đó, hàm phân phối xác suất của biến ngẫu nhiên rời rạc được xác định:

$$F_X(x) = \begin{cases} 0 & x < x_1 \\ \sum_{i=1}^k \tilde{P}_X(x_i) & x_k \leq x < x_{k+1}; \\ 1 & x \geq x_K \end{cases} \quad (\text{PT 2-16})$$

$$F_X(x_i) - F_X(x_{i-1}) = \tilde{P}_X(x_i) \quad (\text{PT 2-17})$$

Thí dụ 2-5: Truyền bản tin số qua kênh nhiễu, xác suất lỗi truyền một số là $P(E) = \frac{2}{5}$; xác suất truyền đúng một số là $P(C) = 1 - P(E) = \frac{3}{5}$

- Xác suất truyền không lỗi:

$$P(CCC) = P(C) \cdot P(C) \cdot P(C) = \left(\frac{3}{5}\right)^3 = \frac{27}{125}$$

- Xác suất truyền bản tin ba số, một số bị lỗi: $3 \cdot \frac{2}{5} \cdot \left(\frac{3}{5}\right)^2 = \frac{54}{125}$

- Xác suất truyền bản tin ba số, hai số bị lỗi: $3 \cdot \left(\frac{2}{5}\right)^2 \cdot \frac{3}{5} = \frac{36}{125}$

Xác suất truyền tất cả ba số bị lỗi:

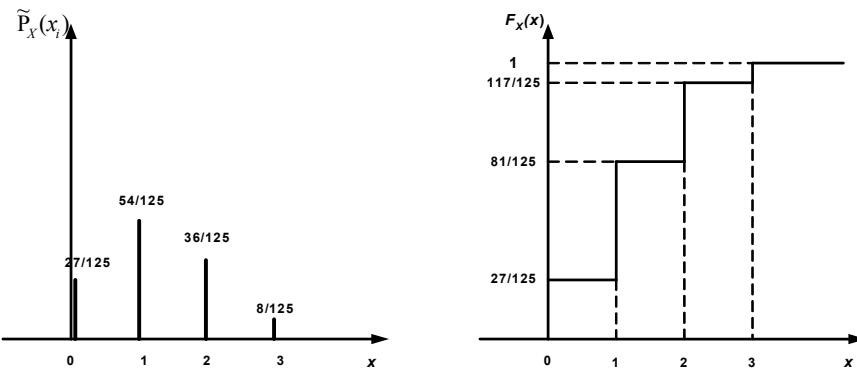
$$P(EEE) = P(E) \cdot P(E) \cdot P(E) = \left(\frac{2}{5}\right)^3 = \frac{8}{125}$$

Nếu ta chọn $X(S)$ là số lỗi ở bản tin thu được thì X là biến ngẫu nhiên rời rạc với $K = 4$ giá trị có thể có là $x_i = 0, 1, 2, 3$. Từ đây ta sẽ có

$\tilde{P}_X(x_i)$ cùng với $F_X(x_i)$ cụ thể là $F_X(0) = \tilde{P}_X(0)$; $F_X(1) = \tilde{P}_X(0) + \tilde{P}_X(1)$,
v.v.

Bảng 2-2: Kết quả Thí dụ 2-5

x_i	$\tilde{P}_X(x_i)$	$F_X(x_i)$
0	$\left(\frac{3}{5}\right)^3 = \frac{27}{125}$	$\frac{27}{125}$
1	$3 \cdot \frac{2}{5} \cdot \left(\frac{3}{5}\right)^2 = \frac{54}{125}$	$\frac{81}{125}$
2	$3 \cdot \left(\frac{2}{5}\right)^2 \cdot \frac{3}{5} = \frac{36}{125}$	$\frac{117}{125}$
3	$\left(\frac{2}{5}\right)^3 = \frac{8}{125}$	$\frac{125}{125}$



Hình 2-3: Kết quả Thí dụ 2-5

2.2.4. Các tham số đặc trưng của biến ngẫu nhiên

Kỳ vọng của biến ngẫu nhiên (expectation)

Định nghĩa 2-7: Kỳ vọng của biến ngẫu nhiên X ký hiệu là m_X hoặc $E[X]$ được xác định:

$$E[X] = m_X = \int_{-\infty}^{+\infty} xf_X(x)dx; X \text{ liên tục} \quad (\text{PT 2-18})$$

$$E[X] = m_X = \sum_{\forall i} x_i \tilde{P}_X(x_i); X \text{ rời rạc} \quad (\text{PT 2-19})$$

Giá trị kỳ vọng của 1 biến ngẫu nhiên là tích phân của biến ngẫu nhiên tương ứng với xác suất của nó. Giá trị kỳ vọng thường không phải là giá trị mà biến ngẫu nhiên có thể lấy được. Nó chỉ có tác dụng diễn giải giá trị trung đợi của

biến ngẫu nhiên như giá trị trung bình trong thời gian dài của biến trong rất nhiều lần chạy thử nghiệm lặp đi lặp lại.

Một số tính chất của kỳ vọng

- $E[c] = c$; c là hằng số
- $E[c \cdot X] = c \cdot E[X]$
- $E[a \cdot X + b \cdot Y] = a \cdot E[X] + b \cdot E[Y]$
- $E[X \cdot Y] = E[X] \cdot E[Y]$ nếu X và Y độc lập với nhau

Thí dụ 2-6: Tìm kỳ vọng của biến ngẫu nhiên X khi biết:

$$F_X(x) = \begin{cases} 0 & x \leq 2 \\ a(x-2)^2 & 2 < x \leq 4 \\ 1 & x > 4 \end{cases}$$

Giải: Do X liên tục, $F_X(x)$ liên tục, nên tại $x=4$, $a(4-2)^2 = 1 \Rightarrow a = \frac{1}{4}$.

Từ đây ta xác định được hàm mật độ xác suất:

$$f_X(x) = \begin{cases} \frac{1}{2} \cdot (x-2) & x \in [2,4] \\ 0 & x \notin [2,4] \end{cases}$$

Kỳ vọng của biến ngẫu nhiên X được xác định:

$$E[X] = \int_{-\infty}^{+\infty} x f_X(x) dx = \int_2^4 \frac{x}{2} \cdot (x-2) dx = \left[\frac{x^3}{6} - \frac{x^2}{2} \right]_2^4 = \frac{10}{3}$$

Thí dụ 2-7: Tìm kỳ vọng của biến ngẫu nhiên X khi biết bảng phân phối

x	0	1	2	3
$\tilde{P}_X(x)$	0,064	0,288	0,432	0,216

Giải:

$$E[X] = \sum_{\forall i} x_i \tilde{P}_X(x_i) = \sum_{i=0}^3 x_i \tilde{P}_X(x_i) = 0,064 + 1,0288 + 2,0432 + 3,0216 = 1,8$$

Phương sai của biến ngẫu nhiên (variance)

Định nghĩa 2-8: Phương sai của biến ngẫu nhiên X ký hiệu là σ_X^2 hoặc $VAR[X]$ được xác định:

$$VAR[X] = \sigma_X^2 = E[(X - m_X)^2] = \int_{-\infty}^{+\infty} (X - m_X)^2 f_X(x) dx; \quad (\text{PT 2-20})$$

Nhận xét: $(X - m_X)$ là độ lệch của biến X so với trung bình của nó, do vậy phương sai chính là trung bình của bình phương độ lệch đó. Phương sai đặc trưng cho độ phân tán của biến ngẫu nhiên xung quanh giá trị kỳ vọng của biến đó. Phương sai càng lớn thì độ bất định của biến tương ứng càng lớn.

$$\sigma_X^2 = E[(X - m_X)^2] = E[X^2] - m_X^2; \quad (\text{PT 2-21})$$

Chứng minh :

$$\begin{aligned} \text{Var}[X] &= E[(X-\mu)^2] = E[X^2 - 2X\mu + \mu^2] = E[X^2] - 2\mu E[X] + \mu^2 \\ &= E[X^2] - 2\mu^2 + \mu^2 = E[X^2] - \mu^2 = E[X^2] - E[X]^2 \end{aligned}$$

Nếu X là biến rời rạc:

$$\sigma_X^2 = \sum_{\forall i} x_i^2 \tilde{P}_X(x_i) - \left(\sum_{\forall i} x_i \tilde{P}_X(x_i) \right)^2; \quad (\text{PT 2-22})$$

Nếu X là biến liên tục :

$$\sigma_X^2 = \int_{-\infty}^{+\infty} x^2 f_X(x) dx - \left(\int_{-\infty}^{+\infty} x f_X(x) dx \right)^2; \quad (\text{PT 2-23})$$

Chú ý :

$$E[X^2] = \int x^2 f(x) dx \text{ với } x \text{ liên tục}$$

Thí dụ 2-8: Tìm phương sai của biến ngẫu nhiên X khi biết bảng phân phối:

x	0	1	2	3
$\tilde{P}_X(x)$	0,064	0,288	0,432	0,216

Giải:

$$\sigma_X^2 = E[X^2] - m_X^2$$

Theo Thí dụ 2-7, $m_X = 1,8$

$$E[X^2] = 0^2 \cdot 0,064 + 1^2 \cdot 0,288 + 2^2 \cdot 0,432 + 3^2 \cdot 0,216 = 3,96$$

$$\sigma_X^2 = E[X^2] - m_X^2 = 3,96 - 3,24 = 0,72$$

Thí dụ 2-9: Cho hàm mật độ xác suất của biến ngẫu nhiên X tuân theo quy luật phân bố hàm số mũ:

$$f_X(x) = \begin{cases} 0 & x \leq 0 \\ \lambda \cdot e^{-\lambda x} & x > 0; \quad \lambda > 0 \end{cases}$$

Tính phương sai của X .

Giải:

$$E[X] = \int_{-\infty}^{+\infty} xf_X(x)dx = \int_0^{\infty} x \cdot \lambda \cdot e^{-\lambda x} dx = \lambda \cdot \frac{1}{\lambda^2} = \frac{1}{\lambda};$$

$$E[X^2] = \int_{-\infty}^{\infty} x^2 f_X(x)dx = \int_0^{\infty} x^2 \cdot \lambda \cdot e^{-\lambda x} dx = \frac{2}{\lambda^2}$$

$$\sigma_X^2 = E[X^2] - m_X^2 = \frac{1}{\lambda^2}$$

Độ lệch chuẩn của biến ngẫu nhiên (standard deviation)

Định nghĩa 2-9: Độ lệch chuẩn của biến ngẫu nhiên X , ký hiệu là σ_X được xác định bởi:

$$\sigma_X = \sqrt{\sigma_X^2} = \sqrt{E[(X - m_X)^2]}; \quad (\text{PT 2-24})$$

Hệ số thay đổi của biến ngẫu nhiên

Định nghĩa 2-10: Hệ số thay đổi của biến ngẫu nhiên X , ký hiệu là c_X được xác định bởi:

$$c_X = \frac{\sigma_X}{E[X]}; \quad (\text{PT 2-25})$$

2.3. Các mô hình phân bố xác suất cơ bản

2.3.1. Phân bố Bernoulli (Bernoulli distribution)

Định nghĩa 2-11: Phân bố Bernoulli là phân bố ngẫu nhiên của biến rời rạc. Biến ngẫu nhiên này nhận hai giá trị là 1 với xác suất p và 0 với xác suất $(1-p)$.

Như vậy, giá trị kỳ vọng và phương sai của phân bố Bernoulli được tính như sau:

$$E(X) = p; \quad (\text{PT 2-26})$$

$$\sigma_X^2 = p(1-p); \quad (\text{PT 2-27})$$

Chứng minh:

Bài tập;

2.3.2. Phân bố nhị thức (binomial distribution)

Định nghĩa 2-12: Một biến ngẫu nhiên X có phân bố nhị thức bậc n , nếu nó có những giá trị $0, 1, 2, \dots, n$ với xác suất:

$$P(X = k) = \binom{n}{k} p^k q^{n-k}; \quad (\text{PT 2-28})$$

Trong đó $0 < p < 1$; $p + q = 1$ và $k = 0, 1, 2, \dots, n$ và $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Hàm mật độ xác suất của phân bố nhị thức được tính như sau:

$$P_X(x) = \sum_{k=0}^n \binom{n}{k} p^k q^{n-k} \delta(x - k); \quad (\text{PT 2-29})$$

với $\delta(x)$ là hàm xung đơn vị (unit impulse function), hay còn gọi là hàm Delta Dirac. Tương ứng, chúng ta có hàm phân phối xác suất:

$$F(x) = \sum_{k=0}^n \binom{n}{k} p^k q^{n-k} u(x - k); \quad (\text{PT 2-30})$$

với $u(x)$ là hàm bước đơn vị (unit step function). Hàm bước đơn vị được định nghĩa như sau :

$$u(x) = \begin{cases} 0; & x < 0 \\ 1; & x \geq 0 \end{cases}; \quad (\text{PT 2-31})$$

Kỳ vọng, phương sai và hệ số thay đổi của biến ngẫu nhiên X tuân theo phân bố nhị thức được xác định:

$$E(X) = m_X = n \cdot p; \quad (\text{PT 2-32})$$

$$\sigma_X^2 = n \cdot p \cdot q = n \cdot p \cdot (1 - p); \quad (\text{PT 2-33})$$

$$c_X = \frac{\sigma_X}{E[X]} = \sqrt{\frac{(1-p)}{n \cdot p}}; \quad (\text{PT 2-34})$$

Bài tập:

khảo sát 15 nhà máy để xem xét số nhà máy sử dụng hệ thống báo cháy. Xác suất để một nhà máy sử dụng hệ thống báo cháy là $p=0.2$. Xác định xác suất để có 5 hoặc ít hơn 5 nhà máy được tìm ra là có sử dụng hệ thống báo cháy.

Giải:

Việc khảo sát là tiến trình nhị thức lặp đi lặp lại 15 phép thử với mỗi phép thử là ột tiến trình Bernoulli trong đó đầu ra phép thử là: sử dụng hệ thống báo cháy hoặc không sử dụng hệ thống báo cháy.

Vậy theo công thức tính xác suất của phân bố nhị thức thì $n=15$ và $p=0.2$.

Ta có xác suất để có đúng 5 nhà máy sử dụng hệ báo cháy là:

$$P(x=5 | B(n=15, p=0.2)) = 0.103$$

Xác suất để có 5 hoặc ít hơn 5 nhà máy có sử dụng hệ báo cháy là

$$P(x \leq 5 | B(n=15, p=0.2)) = P(x=0 | B(n=15, p=0.2)) + P(x=1 | B(n=15, p=0.2)) + P(x=2 | B(n=15, p=0.2)) + P(x=3 | B(n=15, p=0.2)) + P(x=4 | B(n=15, p=0.2)) + P(x=5 | B(n=15, p=0.2))$$

b) Plot the probability distribution of x . Is the distribution highly skewed?

c) What are the mean and the standard deviation of x . What is the coefficient of variation of x ? What would be the value of this coefficient if 150 firms were surveyed rather than 15 (continue to assume that $p=0.2$)? Describe the effect of the larger survey on the relative variability of x .

2.3.2. Phân bố đều (uniform distribution)

Định nghĩa 2-13: Một biến ngẫu nhiên X tuân theo phân bố đều nếu nó có hàm mật độ xác suất là hằng số trong khoảng $[a, b]$:

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & (x < a) \cup (x > b) \end{cases}; \quad (\text{PT 2-35})$$

Nếu $a = 0$, $b = 1$, chúng ta có phân bố đều đơn vị (unit uniform distribution):

$$f(x) = \begin{cases} 1 & 0 \leq x \leq 1 \\ 0 & (x < 0) \cup (x > 1) \end{cases}; \quad (\text{PT 2-36})$$

Tương ứng, chúng ta có hàm phân phối xác suất:

$$F(x) = \frac{x-a}{b-a}; \quad a \leq x \leq b; \quad (\text{PT 2-37})$$

Kỳ vọng, phương sai và hệ số thay đổi của biến ngẫu nhiên X tuân theo phân bố đều được xác định:

$$E(X) = m_X = \frac{a+b}{2}; \quad (\text{PT 2-38})$$

$$\sigma_X^2 = \frac{(b-a)^2}{12}; \quad (\text{PT 2-39})$$

$$c_X = \frac{\sigma_X}{E[X]} = \sqrt{\frac{(b-a)^2}{3(a+b)^2}}; \quad (\text{PT 2-40})$$

2.3.3. Phân bố chuẩn (Gaussian distribution)

Định nghĩa 2-14: Một biến ngẫu nhiên X tuân theo phân bố chuẩn (hay còn được gọi là phân bố Gauss) nếu hàm mật độ xác suất có dạng:

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}; \quad (\text{PT 2-41})$$

Tương ứng, chúng ta có hàm phân phối xác suất:

$$F_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-(\xi-\mu)^2/2\sigma^2} d\xi = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{(x-\mu)/\sigma} e^{-\lambda^2/2} d\lambda; \quad (\text{PT 2-42})$$

Kỳ vọng, phương sai và hệ số thay đổi của biến ngẫu nhiên X tuân theo phân bố chuẩn được xác định:

$$E(X) = m_X = \mu; \quad (\text{PT 2-43})$$

$$\sigma_X^2 = \sigma^2; \quad (\text{PT 2-44})$$

$$c_x = \frac{\sigma_x}{E[X]} = \frac{\sigma}{\mu}; \quad (\text{PT 2-45})$$

2.3.4. Phân bố mũ (exponential distribution)

Định nghĩa 2-15: Phân bố mũ là một phân bố ngẫu nhiên của một biến **x liên tục** với hàm mật độ xác suất:

$$f(x) = \lambda e^{-\lambda x}; \quad (\text{PT 2-46})$$

Ngoài ra, hàm phân bố xác suất, kỳ vọng, phương sai và hệ số thay đổi của biến ngẫu nhiên x tuân theo phân bố mũ được tính như sau:

$$F(x) = 1 - e^{-\lambda x}; \quad (\text{PT 2-47})$$

$$E(x) = \frac{1}{\lambda}; \quad (\text{PT 2-48})$$

$$\sigma(x) = \frac{1}{\lambda^2}; \quad (\text{PT 2-49})$$

$$c_x = \frac{\sigma_x}{E[X]} = 1; \quad (\text{PT 2-50})$$

Trong đó $\lambda > 0$ là tham số tốc độ (rate parameter), $f(x)$ là hàm mật độ xác suất, $F(x)$ là hàm phân phối xác suất, $E(x)$ là kỳ vọng, $\sigma(x)$ là phương sai và c_x là hệ số thay đổi của biến x .

Phân bố hàm số mũ khá quan trọng và thường được dùng phổ biến trong lý thuyết hàng đợi để mô tả các phân bố ngẫu nhiên xảy ra trong các hệ thống thông tin, thí dụ như sự phân bố của khoảng thời gian đến giữa các cuộc gọi cũng như thời gian phục vụ của hệ thống tổng đài điện thoại.

Một tính chất quan trọng của biến ngẫu nhiên X tuân theo phân bố hàm số mũ là **thuộc tính không nhớ** (memoryless):

$$P(X \leq u + t | X > u) = 1 - e^{-\lambda t} = P(X \leq t); \quad (\text{PT 2-51})$$

Nhận xét: Phương trình trên cho thấy hàm phân bố xác suất trong một khoảng $[u; u+t]$ không phụ thuộc vào giá trị khởi đầu u mà chỉ phụ thuộc vào độ rộng của khoảng giá trị t .

2.3.5. Phân bố Poisson (Poisson distribution)

Định nghĩa 2-16: Một biến ngẫu nhiên rời rạc X có phân bố Poisson với tham số α ($\alpha > 0$) nếu nó lấy những giá trị $0, 1, 2, \dots, n$ với xác suất:

$$P(X = k) = e^{-\alpha} \frac{\alpha^k}{k!}; \quad k = 0, 1, \dots \quad (\text{PT 2-52})$$

Phân bố Poisson có hàm mật độ xác suất như sau :

$$P_X(x) = e^{-\alpha} \sum_{k=0}^{\infty} \frac{\alpha^k}{k!} \delta(x - k); \quad (\text{PT 2-53})$$

với $\delta(x)$ là hàm xung đơn vị (unit impulse function). Tương ứng, chúng ta có hàm phân phối xác suất

$$F_X(x) = e^{-\alpha} \sum_{k=0}^{\infty} \frac{\alpha^k}{k!} u(x - k); \quad (\text{PT 2-54})$$

với $u(x)$ là hàm bước đơn vị² (unit step function).

Kỳ vọng, phương sai và hệ số thay đổi của biến ngẫu nhiên X tuân theo phân bố Poisson được xác định:

$$E(X) = m_X = \alpha; \quad (\text{PT 2-55})$$

$$\sigma_X^2 = \alpha; \quad (\text{PT 2-56})$$

$$c_X = \frac{\sigma_X}{E[X]} = \sqrt{\frac{1}{\alpha}}; \quad (\text{PT 2-57})$$

2.3.6. Phân bố Gamma (Gamma distribution)

Định nghĩa 2-17: Phân bố Gamma là phân bố ngẫu nhiên của một biến x liên tục với hàm mật độ xác suất:

$$f(x) = \begin{cases} \lambda^n \frac{x^{n-1}}{(n-1)!} e^{-\lambda x}; & \forall x \geq 0 \\ 0; & \forall x < 0 \end{cases}; \quad (\text{PT 2-58})$$

Ngoài ra, hàm phân bố xác suất, kỳ vọng và phương sai của phân bố Gamma như sau:

$$F(x) = \begin{cases} \int_0^x y^{n-1} e^{-y} dy / (n-1)!; & \forall x \geq 0 \\ 0; & \forall x < 0 \end{cases}; \quad (\text{PT 2-59})$$

$$E(x) = \frac{n}{\lambda}; \quad (\text{PT 2-60})$$

² Hàm bước đơn vị đã được định nghĩa theo PT 2-30

$$\sigma_x^2 = \frac{n}{\lambda^2}; \quad (\text{PT 2-61})$$

Trong đó: λ và n là hai tham số của phân bố Gamma. $\lambda > 0$ được gọi là tham số tỷ lệ (scale parameter), n nguyên dương được gọi là tham số hình dạng (shape parameter).

2.3.7. Mối liên hệ giữa phân bố mũ và phân bố Gamma

Định lý 2-2: Có một tập hợp biến ngẫu nhiên X_i độc lập và tuân theo phân bố mũ với tham số $\lambda > 0$, $i \in \{1, 2, \dots, n\}$. Biến ngẫu nhiên X được định nghĩa như sau:

$$X = \sum_{i=1}^n X_i; \quad (\text{PT 2-62})$$

Hàm mật độ xác suất của biến X tuân theo phân bố Gamma, nghĩa là:

$$f_X(X) = \begin{cases} \lambda^n \frac{X^{n-1}}{(n-1)!} e^{-\lambda X}; & \forall X \geq 0 \\ 0; & \forall X < 0 \end{cases}; \quad (\text{PT 2-63})$$

Chứng minh:

Các biến X_i độc lập tuân theo phân bố mũ với tham số $\lambda > 0$, vì thế:

$$f_{X_i}(x) = \begin{cases} \lambda e^{-\lambda x}; & x \geq 0 \\ 0; & x < 0 \end{cases};$$

Cần phải tìm mật độ xác suất của biến ngẫu nhiên $X = \sum_{i=1}^n X_i$. Sử dụng phương pháp quy nạp:

- Trường hợp $n=1$:

$X = X_1$, do đó:

$$f_X(x) = \begin{cases} \lambda \frac{x^0}{0!} e^{-\lambda x}; & \forall x \geq 0 \\ 0; & \forall x < 0 \end{cases}; \quad (\text{PT 2-64})$$

- Giả thiết (PT 2-62) đúng cho trường hợp n nguyên dương bất kỳ, phải chứng minh (PT 2-62) đúng cho trường hợp $n+1$:

$$X = \sum_{i=1}^n X_i + X_{n+1}; \quad (\text{PT 2-65})$$

Trong đó theo giả thiết thì:

$$f_{\sum_{i=1}^n X_i}(X) = \begin{cases} \frac{\lambda^n X^{n-1}}{(n-1)!} e^{-\lambda X}; & \forall X \geq 0 \\ 0; & \forall X < 0 \end{cases} \quad (\text{PT 2-66})$$

Do hai biến $\sum_{i=1}^n X_i$ và X_{n+1} là hai biến độc lập, theo Định lý 2-1 hàm mật độ xác suất của biến X tổng hai biến trên sẽ là tích chập của hai hàm mật độ xác suất thành phần, tức là:

$$f_X(X) = \int_{-\infty}^{\infty} f_{\sum_{i=1}^n X_i}(X) \times f_{X_{n+1}}(X - Y) dY; \quad (\text{PT 2-67})$$

$$\Leftrightarrow f_X(X) = \begin{cases} \int_0^X \lambda^n \frac{Y^{n-1}}{(n-1)!} e^{-\lambda Y} \lambda e^{-\lambda(X-Y)} dY; & \forall X \geq 0 \\ 0; & \forall X < 0 \end{cases}; \quad (\text{PT 2-68})$$

$$\Leftrightarrow f_X(X) = \begin{cases} \lambda^{n+1} \frac{e^{-\lambda X}}{n!} \int_0^X n Y^{n-1} dY; & \forall X \geq 0 \\ 0; & \forall X < 0 \end{cases}; \quad (\text{PT 2-69})$$

Như vậy cuối cùng ta có:

$$f_X(X) = \begin{cases} \lambda^n \frac{X^{n-1}}{(n-1)!} e^{-\lambda X}; & \forall X \geq 0 \\ 0; & \forall X < 0 \end{cases}; \quad (\text{PT 2-70})$$

Định lý được chứng minh ■

Định lý 2-2 sẽ được sử dụng trong một số chứng minh ở Chương 3.

2.4. Tiến trình ngẫu nhiên (stochastic process)

2.4.1. Khái niệm và định nghĩa

Vấn đề cơ bản liên quan đến phân tích thống kê các hệ thống thông tin số là đặc tính của các tín hiệu ngẫu nhiên như các tín hiệu thoại, tín hiệu truyền hình, dữ liệu máy tính số và tín hiệu nhiễu... Những tín hiệu ngẫu nhiên này có hai đặc tính:

- Các tín hiệu này đều là những hàm biến đổi theo thời gian được xác định trong một số khoảng thời gian quan trắc.
- Các tín hiệu này là ngẫu nhiên và không thể mô tả được chính xác dạng sóng của tín hiệu quan sát được.

Xem xét một thí nghiệm ngẫu nhiên được định ra bởi các kết quả s từ một tập hợp các giá trị S của phép thử. Với mỗi một điểm mẫu s, tương ứng chúng ta có một hàm biến thiên theo thời gian:

$$X(t, s), \quad -T \leq t \leq T; \quad (\text{PT 2-71})$$

Ở đây $2T$ là toàn bộ khoảng thời gian quan trắc. Đối với mỗi một điểm mẫu cố định s_j , đồ thị của hàm $X(t, s_j)$ theo thời gian được gọi là hàm mẫu của một tiến trình ngẫu nhiên. Để đơn giản về mặt ký hiệu, chúng ta biểu thị hàm mẫu này dưới dạng:

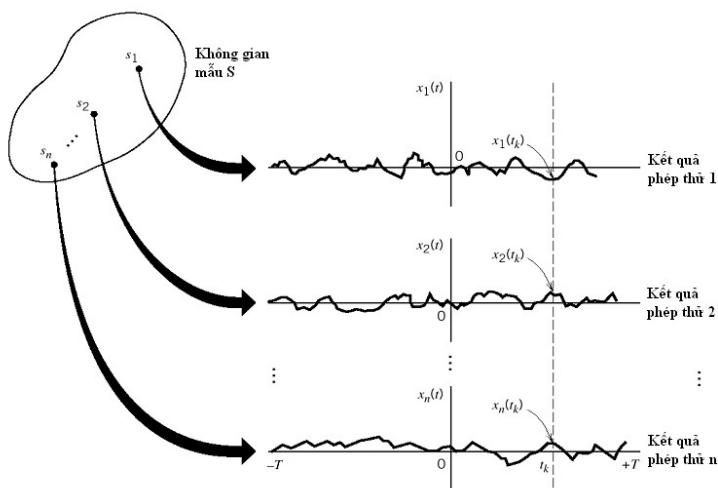
$$x_j(t) = X(t, s_j); \quad (\text{PT 2-72})$$

Hình 2.5 minh họa một tập hợp các hàm mẫu $\{x_j(t) | j = 1, 2, \dots, n\}$. Từ hình vẽ này, chúng ta thấy rằng, đối với thời điểm t_k cố định ở trong khoảng thời gian quan trắc, tập hợp các số:

$$\{x_1(t_k), x_2(t_k), \dots, x_n(t_k)\} = \{X(t_k, s_1), X(t_k, s_2), \dots, X(t_k, s_n)\}; \quad (\text{PT 2-73})$$

hình thành nên một biến ngẫu nhiên. Do vậy, chúng ta có một tập hợp các biến ngẫu nhiên $X(t, s)$ hình thành nên một tiến trình ngẫu nhiên.

Để đơn giản về mặt ký hiệu, người ta thường bỏ s và dùng $X(t)$ biểu thị cho một tiến trình ngẫu nhiên.



Hình 2-4. Tập hợp các hàm mẫu

Định nghĩa 2-18: Tiến trình ngẫu nhiên $X(t)$ được định nghĩa là một tập hợp của các hàm thời gian cùng với một luật xác suất nhất định

2.4.2. Phân loại

Sự khác nhau cơ bản giữa **biến ngẫu nhiên** và **tiến trình ngẫu nhiên** ở chỗ: với biến ngẫu nhiên, kết quả phép thử của thí nghiệm ngẫu nhiên được ánh xạ thành một giá trị số; còn với tiến trình ngẫu nhiên, kết quả của phép thử được ánh xạ thành một hàm thời gian.

Phụ thuộc vào tính liên tục hoặc rời rạc của không gian mẫu S và t , một tiến trình ngẫu nhiên có thể được phân chia thành bốn loại khác nhau:

- **Chuỗi ngẫu nhiên rời rạc** (Discrete Random Sequence) nếu cả S và t là những giá trị rời rạc: Thí dụ: Nếu X_n đại diện cho kết quả thứ n của việc tung một con xúc sắc thì $\{X_n, n \geq 1\}$ là một chuỗi ngẫu nhiên rời rạc vì $t = \{1, 2, 3, \dots\}$ và $S = \{1, 2, 3, 4, 5, 6\}$.
- **Chuỗi ngẫu nhiên liên tục** (Continuous Random Sequence) nếu t là rời rạc còn S là liên tục. Thí dụ: Nếu X_n đại diện cho nhiệt độ tại giờ thứ n trong ngày thì $\{X_n, 1 \leq n \leq 24\}$ là một chuỗi ngẫu nhiên liên tục vì nhiệt độ có thể là bất kỳ giá trị nào trong một khoảng giá trị xác định và do vậy liên tục.
- **Tiến trình ngẫu nhiên rời rạc** (Discrete Random Process) nếu t là liên tục và S là rời rạc. Thí dụ: Nếu $X(t)$ đại diện cho số cuộc gọi điện thoại nhận được trong khoảng thời gian $(0, t)$ thì $\{X(t)\}$ là tiến trình ngẫu nhiên rời rạc vì $S = \{0, 1, 2, 3, \dots\}$.
- **Tiến trình ngẫu nhiên liên tục** (Continuous Random Process) nếu cả t và S là liên tục. Thí dụ: Nếu $X(t)$ đại diện cho nhiệt độ cao nhất tại một địa điểm nào đó trong khoảng thời gian $(0, t)$, thì $\{X(t)\}$ là tiến trình ngẫu nhiên liên tục.

Nói một cách tổng quát, từ **rời rạc** hoặc **liên tục** liên quan đến S , còn **chuỗi** hoặc **tiến trình** liên quan đến t .

Ngoài ra, các tiến trình ngẫu nhiên còn được phân biệt thành: **tiến trình ngẫu nhiên xác định** và **tiến trình ngẫu nhiên không xác định**:

- Nếu giá trị của bất kỳ hàm mẫu nào không được dự đoán chính xác từ các giá trị quan trắc được trong quá khứ thì tiến trình ngẫu nhiên đó được gọi là không xác định.
- Một tiến trình ngẫu nhiên được gọi là xác định nếu giá trị tương lai của bất kỳ một hàm mẫu nào đều có thể dự đoán được từ các giá trị trong quá khứ. Thí dụ: $X(t) = A \cdot \cos(\omega_0 t + \varphi)$

2.5. Các đặc tính thống kê của tiến trình ngẫu nhiên

2.5.1. Các hàm quan hệ xác suất

Xem xét một tiến trình ngẫu nhiên $X(t)$. Đối với một thời điểm xác định t_1 , $X(t_1) = X_1$ là một biến ngẫu nhiên và **hàm phân bố xác suất** $F_X(x_1; t_1)$ của nó được xác định:

$$F_X(x_1; t_1) = P\{X(t_1) \leq x_1\}, \quad X_1 \in \mathfrak{N} \quad (\text{PT 2-74})$$

$F_X(x_1; t_1)$ được gọi là **hàm bậc một** của $X(t)$. **Hàm mật độ xác suất bậc một** tương ứng được xác định:

$$f_X(x_1; t_1) = \frac{\partial F_X(x_1; t_1)}{\partial x_1}; \quad (\text{PT 2-75})$$

Tương tự, với t_1, t_2 , $X(t_1) = X_1$ và $X(t_2) = X_2$ đại diện cho hai biến ngẫu nhiên. Phân bố chung của chúng được gọi là **phân bố bậc hai** và được xác định bởi:

$$F_X(x_1, x_2; t_1, t_2) = P\{X(t_1) \leq x_1, X(t_2) \leq x_2\}; \quad (\text{PT 2-76})$$

Trong đó $X_1, X_2 \in \mathfrak{R}$.

Hàm mật độ bậc hai tương ứng được xác định:

$$f_X(x_1, x_2; t_1, t_2) = \frac{\partial^2 F_X(x_1, x_2; t_1, t_2)}{\partial x_1 \partial x_2}; \quad (\text{PT 2-77})$$

Theo cách tương tự, đối với n biến ngẫu nhiên $X(t_i) = X_i$, ($i = 1, \dots, n$), **hàm phân bố bậc n** là:

$$F_X(x_1, x_2, \dots, x_n; t_1, t_2, \dots, t_n) = P\{X(t_1) \leq x_1, X(t_2) \leq x_2, \dots, X(t_n) \leq x_n\} \quad (\text{PT 2-78})$$

Hàm mật độ bậc n tương ứng là:

$$f_X(x_1, x_2, \dots, x_n; t_1, t_2, \dots, t_n) = \frac{\partial^n F_X(x_1, x_2, \dots, x_n; t_1, t_2, \dots, t_n)}{\partial x_1 \partial x_2 \dots \partial x_n};$$

$$(\text{PT 2-79})$$

2.5.2. Các trung bình thống kê

Kỳ vọng (mean) của một tiến trình ngẫu nhiên $X(t)$ được xác định:

$$E[X(t)] = m_X(t) = \int_{-\infty}^{+\infty} xf_X(x; t) dx; \quad (\text{PT 2-80})$$

Như vậy, ở đây $X(t)$ được xem như là biến ngẫu nhiên đối với một giá trị cố định t .

Hàm tự tương quan (autocorrelation) của $X(t)$ được xác định:

$$R_{XX}(t_1, t_2) = E[X(t_1)X(t_2)] = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x_1 x_2 f_X(x_1, x_2; t_1, t_2) dx_1 dx_2 ;$$

(PT 2-81)

Hàm tự hiệp biến (autocovariance) của $X(t)$ được xác định:

$$C_{XX}(t_1, t_2) = E\{[X(t_1) - m_X(t_1)][X(t_2) - m_X(t_2)]\} = R_{XX}(t_1, t_2) - m_X(t_1)m_X(t_2)$$

(PT 2-82)

Hàm mô men chung bậc n của $X(t)$ được xác định:

$$E[X(t_1) \cdots X(t_n)] = \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} x_1 \cdots x_n f_X(x_1, \dots, x_n; t_1, \dots, t_n) dx_1 \cdots dx_n ;$$

(PT 2-83)

2.5.3. Tính dừng

Tính dừng cảm nhận chặt

Định nghĩa 2-19: Một tiến trình ngẫu nhiên $X(t)$ được gọi là có tính dừng cảm nhận chặt (strict-sense stationary – SSS) nếu các đặc tính thống kê của nó không thay đổi khi có sự dịch chuyển nào đó về mặt thời gian so với ban đầu. Nói cách khác, tiến trình $X(t)$ là SSS nếu:

$$f_X(x_1, \dots, x_n; t_1, \dots, t_n) = f_X(x_1, \dots, x_n; t_1 + c, \dots, t_n + c); \quad \forall c ;$$

(PT 2-84)

Từ phương trình (PT 2-84), chúng ta thấy rằng $f_X(x_1; t_1) = f_X(x_1; t_1 + c); \quad \forall c$. Do vậy, hàm mật độ bậc một của tiến trình dừng $X(t)$ là độc lập với t . Do đó:

$$f_X(x_1; t) = f_X(x_1) ; \quad (PT 2-85)$$

Tương tự, $f_X(x_1, x_2; t_1, t_2) = f_X(x_1, x_2; t_1 + c, t_2 + c)$; $\forall c$. Đặt $c = -t_1$, chúng ta có:

$$f_X(x_1, x_2; t_1, t_2) = f_X(x_1, x_2; 0, t_2 - t_1); \quad (\text{PT 2-86})$$

Nhận xét: Các phương trình trên chỉ cho chúng ta thấy rằng, nếu $X(t)$ là SSS, hàm mật độ chung của các biến ngẫu nhiên $X(t)$ và $X(t + \tau)$ là độc lập với t và chỉ phụ thuộc vào khoảng thời gian τ chênh lệch giữa chúng.

Tính dừng cảm nhận lỏng

Định nghĩa 2-20: Một tiến trình ngẫu nhiên $X(t)$ có tính dừng cảm nhận lỏng (wide-sense stationary – WSS) nếu kỳ vọng của nó là hằng số:

$$E[X(t)] = m_X; \quad (\text{PT 2-87})$$

và hàm tự tương quan của nó chỉ phụ thuộc vào chênh lệch thời gian τ :

$$E[X(t)X(t + \tau)] = R_{XX}(\tau); \quad (\text{PT 2-88})$$

Từ phương trình (2-86) và (2-87), hàm tự hiệp biến của một tiến trình WSS cũng chỉ phụ thuộc vào chênh lệch thời gian τ :

$$C_{XX}(\tau) = R_{XX}(\tau) - m_X^2; \quad (\text{PT 2-89})$$

Đặt $\tau = 0$, từ phương trình (2-88), chúng ta có:

$$E[X^2(t)] = R_{XX}(0); \quad (\text{PT 2-90})$$

Như vậy, công suất trung bình của tiến trình WSS độc lập với t và bằng $R_{XX}(0)$.

Lưu ý rằng, một tiến trình SSS là WSS, nhưng một tiến trình WSS chưa hẳn đã là SSS.

Hai tiến trình $X(t)$ và $Y(t)$ được gọi là có **tính dừng cảm nhận lỏng chung** (jointly WSS) nếu bản thân riêng từng tiến trình là WSS và **hàm tương quan chéo** (cross-correlation) giữa chúng chỉ phụ thuộc vào chênh lệch thời gian τ .

$$R_{XY}(t, t + \tau) = E[X(t)Y(t + \tau)] = R_{XY}(\tau); \quad (\text{PT 2-91})$$

Từ phương trình (2-90), **hàm hiệp biến chéo** (cross-covariance) của các tiến trình WSS chung giữa $X(t)$ và $Y(t)$ cũng phụ thuộc vào chênh lệch thời gian τ :

$$C_{XY}(\tau) = R_{XY}(\tau) - m_X m_Y; \quad (\text{PT 2-92})$$

Trung bình thời gian và ergodic

Kỳ vọng trung bình thời gian (time-averaged) của một hàm mẫu $x(t)$ của một tiến trình ngẫu nhiên $X(t)$ được định nghĩa:

$$\bar{x} = \langle x(t) \rangle = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t) dt; \quad (\text{PT 2-93})$$

Tương tự, **hàm tự tương quan trung bình thời gian** của một hàm mẫu $x(t)$ được xác định:

$$\bar{R}_{xx}(\tau) = \langle x(t)x(t+\tau) \rangle = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t)x(t+\tau) dt; \quad (\text{PT 2-94})$$

Lưu ý rằng: \bar{x} và $\bar{R}_{xx}(\tau)$ là các biến ngẫu nhiên; giá trị của chúng phụ thuộc hàm mẫu của $X(t)$ được sử dụng.

Nếu $X(t)$ là tiến trình dừng, bằng việc lấy giá trị trung bình (kỳ vọng) trên cả hai phía của phương trình (2-92) và (2-93) chúng ta có:

$$E[\bar{x}] = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{T/2} E[x(t)] dt = m_x; \quad (\text{PT 2-95})$$

$$E[\bar{R}_{xx}(\tau)] = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{T/2} E[x(t)x(t+\tau)] dt = R_{xx}(\tau); \quad (\text{PT 2-96})$$

Từ các phương trình (2-94) và (2-95) chúng ta thấy, kỳ vọng của trung bình thời gian bằng với kỳ vọng thống kê và kỳ vọng của tự tương quan trung bình thời gian bằng với tự tương quan thống kê.

Một tiến trình ngẫu nhiên $X(t)$ có **tính ergodic** nếu các trung bình thời gian là như nhau cho tất cả các hàm mẫu và bằng các trung bình thống kê tương ứng.

Do vậy, trong một tiến trình ergodic, tất cả các thống kê của nó có thể nhận được qua việc quan trắc một hàm mẫu đơn $x(t) = X(t, s)$; (s là cố định) của tiến trình.

Một tiến trình dừng $X(t)$ được gọi là ergodic theo kỳ vọng nếu:

$$\bar{x} = \langle x(t) \rangle = E[X(t)] = m_x; \quad (\text{PT 2-97})$$

Tương tự, một tiến trình dừng $X(t)$ được gọi là ergodic theo tự tương quan nếu:

$$\bar{R}_{XX}(\tau) = \langle x(t)x(t+\tau) \rangle = E[X(t)X(t+\tau)] = R_{XX}(\tau); \quad (\text{PT 2-98})$$

Trong thực tế, việc kiểm tra tính ergodic của một tín trình ngẫu nhiên là rất khó khăn. Một giả thiết hợp lý cho việc phân tích phần lớn các tín hiệu thông tin các dạng sóng ngẫu nhiên của chúng có tính ergodic theo kỳ vọng và theo tự tương quan.

2.6. Các tiến trình ngẫu nhiên thường gặp

2.6.1. Tiến trình đếm

Tiến trình đếm (counting process)

Định nghĩa 2-21: Một tiến trình ngẫu nhiên $\{N(t), t \geq 0\}$ được gọi là tiến trình đếm (counting process) nếu $N(t)$ đại diện cho tổng số các “sự kiện” xuất hiện cho tới thời điểm t . Nói cách khác, một tiến trình đếm phải thỏa mãn các điều kiện sau:

- i. $N(t) \geq 0$.
- ii. $N(t)$ là số nguyên.
- iii. Nếu $s < t$, thì $N(s) \leq N(t)$.
- iv. Đối với $s < t$ thì $N(t) - N(s)$ chính là số sự kiện xuất hiện trong khoảng thời gian (s, t) .

Hình 2-5. Tiến trình đếm

Thí dụ 2-10:

Tiến trình tăng trưởng độc lập (independent increments)

Định nghĩa 2-22: Một tiến trình đếm $\{N(t), t \geq 0\}$ được gọi tăng trưởng độc lập (independent increments) nếu như $[N(s) - N(t)]$ ($s > t$) và $[N(u) - N(v)]$ ($u > v$) là hai biến ngẫu nhiên độc lập với điều kiện (t, s) và (v, u) là hai khoảng thời gian độc lập, không trùng nhau.

Hình 2-6. Tiến trình tăng trưởng độc lập

Thí dụ 2-11:

Tiến trình tăng trưởng dừng (stationary increments)

Định nghĩa 2-23: Một tiến trình đếm được gọi là tăng trưởng dừng (stationary increments) nếu số các sự kiện trong khoảng thời gian $(t_1 + s, t_2 + s)$; tức là $\{N(t_2 + s) - N(t_1 + s)\}$ có cùng phân bố với số sự kiện xuất hiện trong khoảng thời gian (t_1, t_2) ; tức là $\{N(t_2) - N(t_1)\}$ đối với mọi $t_1 < t_2$ và $s > 0$.

Nói cách khác, tiến trình đếm là tăng trưởng dừng nếu như việc phân bố số các sự kiện xuất hiện trong bất kỳ khoảng thời gian nào chỉ phụ thuộc vào độ dài của khoảng thời gian quan trắc mà không phụ thuộc vào thời điểm bắt đầu tiến hành quan trắc.

Hình 2-7. Tiến trình tăng trưởng dừng

Thí dụ 2-12:

2.6.2. Tiến trình Poisson

2.7. Kết luận

Chương 2 tập trung giới thiệu những khái niệm cơ bản về xác suất, biến ngẫu nhiên và các hàm xác suất, các phân bố xác suất cơ bản, tổng quan về tiến trình ngẫu nhiên và một số tiến trình ngẫu nhiên phổ biến như Poisson, Gauss, Markov. Đây có thể được xem như là những kiến thức cơ bản, nhằm hỗ trợ bạn đọc trong việc học và tìm hiểu những chương tiếp theo.

- Tiến trình tăng trưởng độc lập
- Tiến trình tăng trưởng dừng
- Tiến trình Bernoulli (Bernoulli process)
- Tiến trình nhị thức (Binomial process)
- Tiến trình Poisson:
 - Một số định lý của tiến trình Poisson
 - Tính chất của tiến trình Poisson
 - Tính hợp nhất (merging property)
 - Tính tách biệt

Chương 3 Hệ thống hàng đợi

3.1. Giới thiệu

Trong Chương 2, chúng ta đã khảo sát các tiến trình ngẫu nhiên thông dụng. Đây chính là các công cụ toán học thường được sử dụng trong hệ thống hàng đợi được đề cập trong chương này.

Hàng đợi là thành phần không thể thiếu khi mô hình hóa hệ thống sử dụng phương pháp toán học và là công cụ hữu hiệu để đánh giá hiệu năng hoạt động của các hệ thống phục vụ nói chung cũng như của hệ thống máy tính và thông tin nói riêng.

Định nghĩa 3-1: Hệ thống phục vụ là hệ thống được sử dụng để phục vụ các yêu cầu nhất định hoặc các khách hàng thông qua các điểm cung cấp dịch vụ. Yêu cầu hoặc khách hàng rời hệ thống sau khi dịch vụ đã được cung cấp.

Thí dụ 3-1:

Các hệ thống điện thoại: đáp ứng số lượng lớn khách hàng quay số để kết nối đến một trong những đường ra hữu hạn của tổng đài.

Trong mạng máy tính: khi gói tin được chuyển từ nguồn tới đích và đi qua một số lượng các nút trung gian. Hệ thống hàng đợi xuất hiện tại mỗi nút mạng ở quá trình lưu tạm thông tin tại bộ đệm.

Hệ thống máy tính: khi các công việc tính toán và truyền làm việc của hệ thống yêu cầu dịch vụ từ bộ xử lý trung tâm và từ các nguồn khác.

Tất cả các hệ thống trên thí dụ trên đều được mô tả bằng khái niệm hàng đợi.

3.2. Mô hình hàng đợi – Ký hiệu Kendall

3.2.1. Mô hình hàng đợi đơn

Một hàng đợi đơn giản được định nghĩa bởi các đặc điểm cơ bản sau đây:

- **Tiến trình tới** (arrival process): được đặc trưng bởi tốc độ yêu cầu (hoặc sự kiện) λ tới hệ thống (yêu cầu trên đơn vị thời gian) và tính chất ngẫu nhiên của tiến trình tới.

Thí dụ: các gói tin được gửi tới đầu vào của một router là một tiến trình tới, được đặc trưng bởi số gói/s và phân bố ngẫu nhiên của các gói, tức là phân bố ngẫu nhiên của khoảng thời gian giữa hai gói liên tiếp, thí dụ như phân bố mũ.

- **Tiến trình phục vụ** (service process): tiến trình phục vụ gắn liền với các trạm phục vụ (server) trong một hệ thống phục vụ.

Mỗi trạm phục vụ bao giờ cũng được đặc trưng bởi tốc độ phục vụ trung bình μ (số yêu cầu trên một đơn vị thời gian) và tính chất ngẫu nhiên của tiến trình phục vụ.

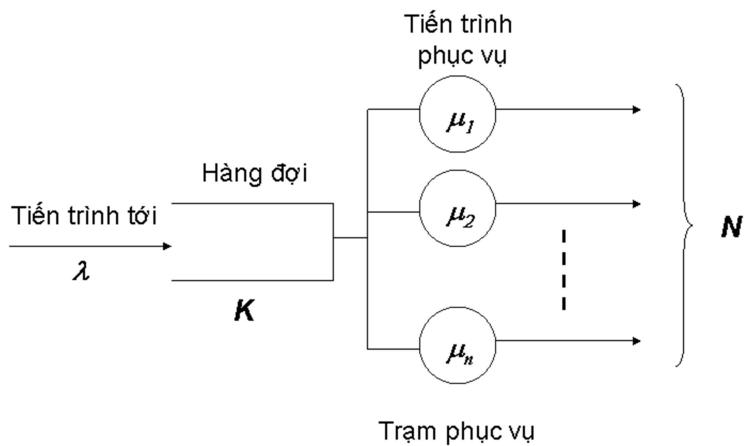
Thí dụ: Các gói tin được gửi ra đầu ra của một router được coi như một tiến trình phục vụ và được đặc trưng bởi số gói được phục vụ trong 1s. Thông thường các gói tin trong Internet có độ dài thay đổi, vì vậy thời gian phục vụ một gói tin trung bình (thời gian để gửi gói đó từ bit đầu tiên đến bit cuối cùng lên đường truyền) sẽ có tính chất ngẫu nhiên tuân theo một phân bố nào đó.

- **Số trạm phục vụ** (server): Một hệ thống phục vụ có thể có nhiều trạm phục vụ. Trong trường hợp này hệ thống phục vụ có thể phục vụ một lúc nhiều yêu cầu.
- **Quy tắc phục vụ** (service strategy): thể hiện nguyên tắc và trình tự phục vụ các yêu cầu khi chúng đi vào hệ thống phục vụ.

Thí dụ: trong nguyên tắc FIFO yêu cầu nào vào hệ thống trước thì được phục vụ trước. Trong nguyên tắc LIFO, yêu cầu nào vào hệ thống sau cùng lại được phục vụ đầu tiên.

- **Hàng đợi** (queue): Khi một yêu cầu đi vào một hệ thống phục vụ, nếu tất cả các trạm phục vụ đều bận thì yêu cầu đó sẽ phải đợi trong hàng đợi. Hàng đợi được đặc trưng bởi độ lớn K .

Hình 3-1 minh họa một hệ thống hàng đợi với N trạm phục vụ, độ lớn hàng đợi K , tiến trình tới với tham số λ và N tiến trình phục vụ với tham số $\{\mu_1, \dots, \mu_n\}$.



Hình 3-1. Hệ thống hàng đợi với N trạm phục vụ, hàng đợi có độ lớn K

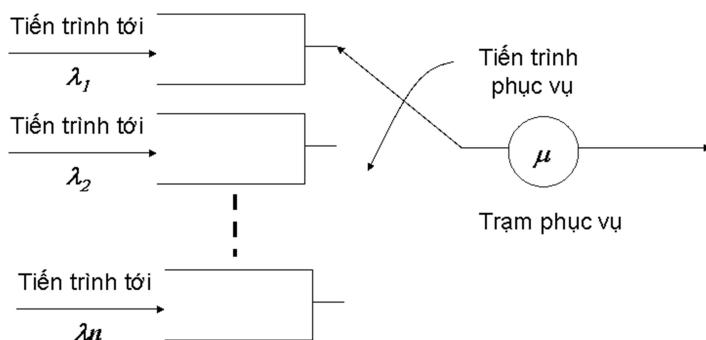
3.2.2. Ký hiệu Kendall

Ký hiệu Kendall được sử dụng để mô tả hệ thống hàng đợi đơn, do David George Kendall phát triển. Hệ thống hàng đợi được mô tả bằng một chuỗi ký tự như sau:

A/S/C/[K]/[P]/[D]

Các ký tự trong dấu ngoặc vuông không bắt buộc phải có. Trong đó:

- **A** và **S**: Thể hiện phân bố ngẫu nhiên của tiến trình tới (A) và tiến trình phục vụ (S). Một số phân bố ngẫu nhiên thông dụng là:
 - **M**: phân bố Markov hay phân bố mũ
 - **D**: phân bố xác định, phi ngẫu nhiên (derterministic). Trong phân bố D khoảng thời gian giữa 2 sự kiện đến hệ thống hoặc thời gian xử lý một yêu cầu tại trạm phục vụ là hằng số.
 - **G** hoặc **GI**: Phân bố ngẫu nhiên bất kỳ.
 - **C**: Số trạm phục vụ, $C \in E ; C \geq 1$
- **K**: Thể hiện số yêu cầu mà hệ thống có thể chứa được, kể cả yêu cầu đang được phục vụ bởi trạm phục vụ, $K \in N ; K \geq 0$. Nếu K được bỏ qua thì mặc định $K = \infty$
- **P**: Số lượng các yêu cầu đi vào hệ thống (Calling population). Nếu P được bỏ qua thì mặc định $P = \infty$.
- **D**: Quy tắc phục vụ (service discipline/strategy).
- FIFO (First-In-First-Out): Yêu cầu nào vào hệ thống trước được phục vụ trước.
- LIFO (Last-In-First-Out): Yêu cầu nào vào hệ thống sau cùng được phục vụ đầu tiên.
- Round-robin (RR): Hệ thống RR có nhiều hàng đợi cho nhiều tiến trình tới khác nhau. Các yêu cầu của các tiến trình tới được phục vụ theo thứ tự lần lượt (Hình 3-2).



Hình 3-2. Round robin

- SIRO (Service In Random Order): Các yêu cầu được phục vụ theo trình tự ngẫu nhiên, không phụ thuộc vào trình tự tới hệ thống.

- PQ (Priority Queue): Hàng đợi có ưu tiên. Giống như RR, hệ thống này có nhiều hàng đợi cho các tiến trình tới khác nhau. Tuy nhiên thứ tự phục vụ các yêu cầu trong các hàng đợi tuân theo trình tự ưu tiên. Trạm phục vụ sẽ phục vụ các yêu cầu trong hàng đợi có mức ưu tiên cao nhất. Khi hàng đợi rỗng, trạm phục vụ sẽ chuyển xuồng phục vụ hàng đợi có yêu cầu ở mức ưu tiên thấp hơn. Khi một hàng đợi bất kỳ ở mức ưu tiên cao hơn có yêu cầu, trạm phục vụ sẽ chuyển sang phục vụ yêu cầu ở hàng đợi có mức ưu tiên cao.

Thí dụ 3-2:

Hệ thống hàng đợi M/M/2/K:

- Tiến trình tới theo phân bố mũ.
- Tiến trình phục vụ theo phân bố mũ.
- Hệ thống có 2 trạm phục vụ.
- Hàng đợi có độ lớn là K.
- Số lượng yêu cầu đi vào hệ thống là ∞ .
- Quy tắc phục vụ: FIFO (ngầm định).

3.2.3. Các tham số quan trọng để đánh giá đặc tính của hệ thống hàng đợi

Khi đánh giá đặc tính của hệ thống hàng đợi, người ta thường hay chú ý đến các tham số sau:

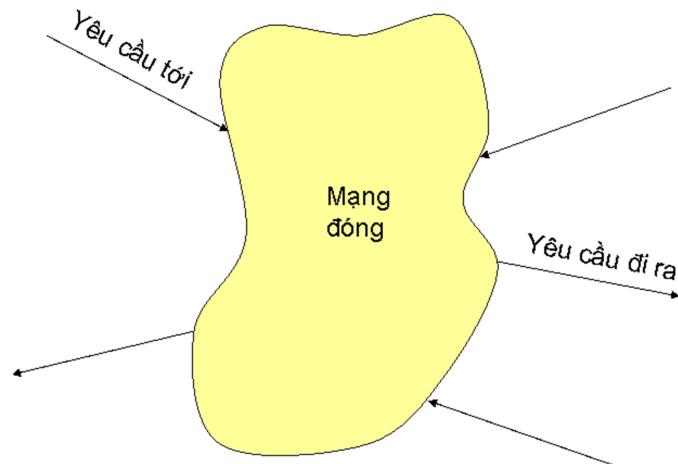
- **Số yêu cầu trung bình nằm trong hệ thống N .**
- **Số yêu cầu trung bình nằm trong hàng đợi N_q .**
- **Số yêu cầu trung bình được xử lý ở trạm phục vụ N_s .**
Mỗi quan hệ và ý nghĩa của 3 tham số trên sẽ được đề cập kỹ hơn ở các phần tiếp theo.
- **Thời gian trung bình một yêu cầu lưu lại trong hệ thống W .**
- **Thời gian trung bình một yêu cầu phải chờ trong hàng đợi trước khi được phục vụ T_q .**
- **Thời gian phục vụ trung bình một yêu cầu tại trạm phục vụ T_s .**
- **Xác suất để thời gian một yêu cầu phải đợi trong hàng đợi t_q nhỏ hơn một khoảng thời gian t cho trước**
 $p\{t_q < t\}$.
- **Xác suất để hệ thống có i yêu cầu p_i .**
- **Xác suất để một yêu cầu bị từ chối** e, tức xác suất yêu cầu không được phép đi vào hệ thống do hàng đợi bị đầy.

3.3. Hệ thống hàng đợi ở trạng thái ổn định – Định lý Little

3.3.1. Hệ thống hàng đợi ổn định

Xét một hệ thống có ranh giới đóng (enclosed network), gọi tắt là mạng đóng (Hình 3-3). Mạng này có thể có một hoặc nhiều cổng cho yêu cầu đi tới hệ thống (cổng vào) và một số cổng cho yêu cầu đi ra (cổng ra).

Định nghĩa 3-2: Một mạng không tự tạo ra các yêu cầu, không hủy các yêu cầu, không thay đổi tính chất các yêu cầu trong mạng được gọi là **mạng đóng** (enclosed network).



Hình 3-3. Mạng đóng

Do mạng không tạo ra mà cũng không hủy các yêu cầu, yêu cầu chỉ có thể đi vào, được lưu giữ trong mạng và đi ra khỏi mạng. Nếu tổng yêu cầu trung bình đi vào mạng lớn hơn tổng yêu cầu đi ra khỏi mạng thì tổng số yêu cầu nằm trong mạng sẽ không ngừng tăng lên. Mặt khác, nếu tốc độ tới nhỏ hơn tốc độ đi ra thì tổng yêu cầu trong mạng sẽ giảm về không. Tại thời điểm đó, tốc độ tới và tốc độ ra sẽ cân bằng nhau.

Với lý luận như trên, chúng ta nhận thấy một hệ thống ở trạng thái ổn định luôn có tổng tốc độ tới trung bình và tổng tốc độ ra trung bình bằng nhau.

Một tính chất quan trọng của hệ thống ở trạng thái ổn định là tính chất thống kê của mạng tại một thời điểm cho trước sẽ tương đương với tính chất thống kê của mạng tại một thời điểm bất kỳ.

Các hệ thống hàng đợi được xem xét trong quyển sách này là một trường hợp riêng của mạng đóng, hoạt động ổn định. Thông thường khi xác định đặc tính của một hệ thống hàng đợi, người ta quan tâm đến các tham số sau:

- $N(t)$: số yêu cầu nằm trong hệ thống tại thời điểm t .
- $N_q(t)$: số yêu cầu nằm trong hàng đợi tại thời điểm t .
- $N_s(t)$: số yêu cầu đang được phục vụ.
- T : Thời gian tổng cộng một yêu cầu nằm trong hệ thống hàng đợi.
- T_q : Thời gian một yêu cầu phải nằm chờ trong hàng đợi trước khi đến lượt nó được phục vụ.
- T_s : thời gian một yêu cầu được phục vụ bởi server.

Ta có thể dễ dàng suy luận:

$$N(t) = N_s(t) + N_q(t); \quad (\text{PT 3-1})$$

$$T = T_q + T_s; \quad (\text{PT 3-2})$$

Mặt khác, nếu gọi:

- λ là tốc độ trung bình của yêu cầu đi tới hệ thống (số yêu cầu/đơn vị thời gian).
- μ là tốc độ phục vụ trung bình của trạm phục vụ (số yêu cầu/đơn vị thời gian).
- c là số trạm phục vụ của hệ thống hàng đợi

Ta định nghĩa **tải của hệ thống** bằng biểu thức:

$$\rho = \frac{\lambda}{c\mu}; \quad (\text{PT 3-3})$$

ρ còn được gọi là **mật độ lưu lượng của hệ thống** (traffic intensity) và có ý nghĩa là phần thời gian mà người quan sát bên ngoài bắt gặp tối thiểu một trạm phục vụ ở trạng thái bận (đang phục vụ yêu cầu).

Ngoài ra, có thể hiểu $c\mu$ là tốc độ phục vụ trung bình của toàn hệ thống hàng đợi. Từ định nghĩa chúng ta thấy rằng:

$$0 \leq \rho \leq 1; \quad (\text{PT 3-4})$$

3.3.2. Định lý Little

Xét một mạng đóng bất kỳ với các tham số sau:

- $N(t)$ - Số yêu cầu nằm trong hệ thống tại thời điểm t .
- α_t - Số yêu cầu đi đến hệ thống trong khoảng thời gian từ $(0,t)$.
- β_t - Số yêu cầu rời khỏi hệ thống trong khoảng thời gian từ $(0,t)$.

Như vậy nếu gọi N_t là số lượng yêu cầu trung bình nằm trong hệ thống trong $(0,t)$, ta có:

$$N_t = \frac{1}{t} \int_0^t N(t) dt ; \quad (\text{PT 3-5})$$

Mặt khác gọi:

- T_i là thời gian mà yêu cầu i lưu lại trong hệ thống.
- λ_t là số yêu cầu đến hệ thống trong khoảng thời gian $(0,t)$:

$$\lambda_t = \frac{\alpha_t}{t} ; \quad (\text{PT 3-6})$$

- T_t là thời gian lưu lại trung bình của yêu cầu trong hệ thống trong khoảng thời gian $(0,t)$:

$$T_t = \frac{1}{\alpha_t} \sum_{i=1}^{\alpha_t} T_i ; \quad (\text{PT 3-7})$$

Giả sử các giới hạn sau đây tồn tại :

$$N = \lim_{t \rightarrow \infty} N_t ; \quad \lambda = \lim_{t \rightarrow \infty} \lambda_t ; \quad T = \lim_{t \rightarrow \infty} T_t ;$$

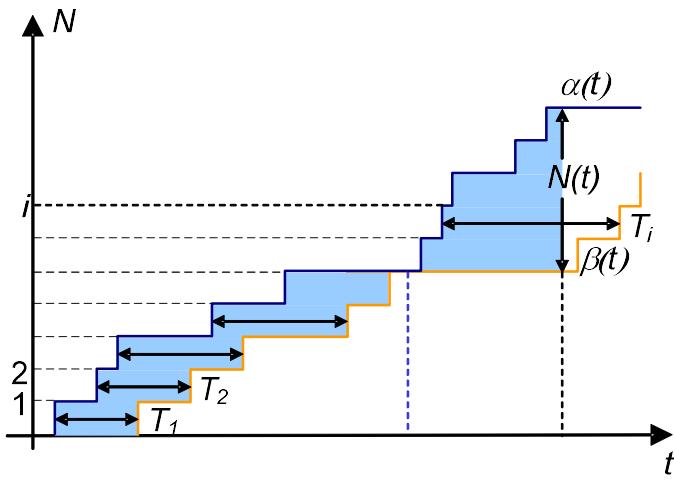
Định lý Little được phát biểu như sau:

Định lý 3-1: Số yêu cầu trung bình nằm trong hệ thống bằng tích của tốc độ tới trung bình với thời gian lưu lại hệ thống trung bình của yêu cầu:

$$N = \lambda T ; \quad (\text{PT 3-8})$$

Chứng minh định lý Little:

Chứng minh công thức Little bằng phương pháp hình học theo minh họa dưới đây.



Hình 3-4. Quan hệ giữa số yêu cầu tới và số yêu cầu được phục vụ

Xét trong khoảng $(0, t)$:

Diện tích phần gạch chéo:

$$S_t = \int_0^t N(t) dt = \int_0^t [\alpha(t) - \beta(t)] dt; \quad (\text{PT 3-9})$$

Mặt khác diện tích này cũng bằng:

$$S_t = 1 \times \sum_{i=1}^{\alpha_t} T_i; \quad (\text{PT 3-10})$$

Như vậy, từ (PT 3-9, 3-10) ta rút ra:

$$\int_0^t N(t) dt = \sum_{i=1}^{\alpha_t} T_i \quad (\text{PT 3-11})$$

$$\text{Hay: } \frac{1}{t} \int_0^t N(t) dt = \frac{\alpha_t}{t} \frac{\sum_{i=1}^{\alpha_t} T_i}{\alpha_t} \quad (\text{PT 3-12})$$

tức là :

$$N_t = \lambda_t T_t; \quad (\text{PT 3-13})$$

Nếu giới hạn sau đây tồn tại :

$$N = \lim_{t \rightarrow \infty} N_t; \quad \lambda = \lim_{t \rightarrow \infty} \lambda_t; \quad T = \lim_{t \rightarrow \infty} T_t; \quad (\text{PT 3-14})$$

Từ (PT 3-13) và (3-14)

$$N = \lambda T; \quad (\text{PT 3-15})$$

Công thức được chứng minh ■

Nhận xét: Định lý Little đúng cho một mạng đóng bất kỳ, không phụ thuộc vào các tham số hàng đợi, tiến trình đến và tiến trình phục vụ.

3.3.3. Một số đặc tính khác của hệ thống đóng, hoạt động ở trạng thái ổn định

Xét một hệ thống hàng đợi như Hình 3-5. Hệ thống hàng đợi được đặc trưng bởi các tham số sau:

- Tiến trình tới với tốc độ trung bình λ .
- c trạm phục vụ với tốc độ phục vụ trung bình của từng trạm là μ .
- Hàng đợi có độ lớn vô tận (yêu cầu luôn được chấp nhận).

Nếu xem xét hệ thống hàng đợi như một mạng đóng bao gồm hàng đợi và các trạm phục vụ như Hình 3-5 a, theo định lý Little ta có:

$$N = \lambda T; \quad (\text{PT 3-16})$$

Mặt khác, nếu xem xét riêng khối hàng đợi của hệ thống hàng đợi trên (Hình 3-5 b), căn cứ vào định nghĩa ta thấy đây vẫn là một mạng đóng, vì vậy có thể áp dụng định lý Little như sau:

$$N_q = \lambda T_q; \quad (\text{PT 3-17})$$

Mặt khác theo (PT 3-2): $T = T_q + T_s$ với $T_s = \frac{1}{c\mu}$ và theo (PT 3-16) và (PT 3-17):

$$\frac{N}{\lambda} = \frac{1}{c\mu} + \frac{N_q}{\lambda}; \quad (\text{PT 3-18})$$

Hay:

$$N = \frac{\lambda}{c\mu} + N_q; \quad (\text{PT 3-19})$$

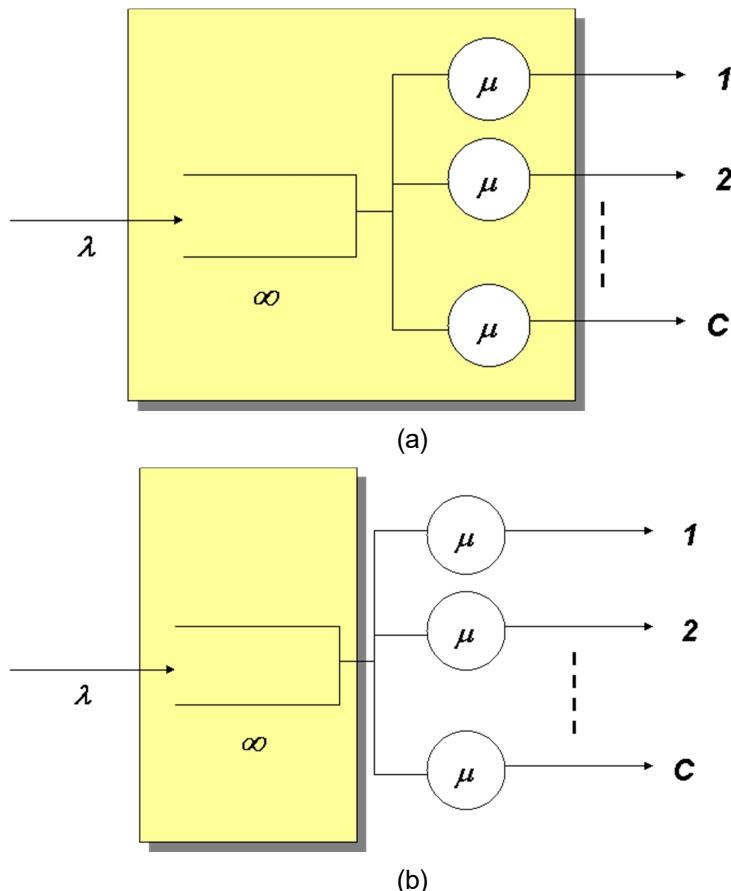
Ngoài ra theo (PT 3-3) $\rho = \frac{\lambda}{c\mu}$ nên (PT 3-19) trở thành:

$$N = \rho + N_q \quad (\text{PT 3-20})$$

Như vậy ta có thể nói rằng ρ chính là số yêu cầu trung bình mà hệ thống phục vụ phải xử lý tại các trạm phục vụ.

Nhận xét: Định lý Little và các đặc tính của hệ thống đóng mà chúng ta xét đến thời điểm này không phụ thuộc vào các đặc

điểm riêng biệt của từng hệ thống hàng đợi như tiến trình đến, tiến trình phục vụ, số trạm phục vụ .v.v. Tuy nhiên nếu muốn khảo sát các đặc tính khác của hệ thống hàng đợi chúng ta cần phải biết các đặc tính thống kê của luồng lưu lượng đầu vào cũng như tiến trình xử lý yêu cầu.



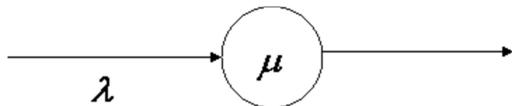
Hình 3-5.Hệ thống hàng đợi được quan sát với hai ranh giới khác nhau:
(a) Hàng đợi và các trạm phục vụ; (b) Hàng đợi

3.4. Hàng đợi M/M/1/1

Kể từ phần này, chúng ta lần lượt khảo sát đặc tính hoạt động của các hệ thống hàng đợi từ đơn giản đến phức tạp, với các đặc tính thống kê cho trước của tiến trình đến và tiến trình phục vụ.

Chú ý rằng điều kiện để khảo sát đặc tính một hệ thống hàng đợi là chúng phải hoạt động ở chế độ ổn định, nghĩa là các tính chất

thống kê của hệ thống không phụ thuộc vào thời gian. Một hệ thống như vậy còn được gọi là một **hệ thống “dừng”**.



Hình 3-6. Hàng đợi M/M/1/1

Hệ thống hàng đợi M/M/1/1 được đặc trưng bởi:

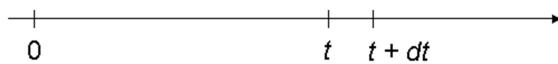
- Tiến trình tới tuân theo phân bố mũ với tham số là λ .
- Tiến trình phục vụ tuân theo phân bố mũ với tham số là μ .
- Một trạm phục vụ
- Không có hàng đợi, như vậy hệ thống có thể chứa tối đa một yêu cầu.

Gọi $N(t)$ là số yêu cầu trong hệ thống tại thời điểm t . Do chỉ có hai khả năng là hệ thống không có yêu cầu, hoặc trạm phục vụ đang xử lý một yêu cầu nên: $N(t) \in \{0,1\}$.

Gọi:

- $p_n^i(t_1, t_2)$ là xác suất để trong khoảng thời gian (t_1, t_2) có n yêu cầu đi vào hệ thống.
- $p_k^o(t_1, t_2)$ là xác suất để trong khoảng thời gian (t_1, t_2) có k yêu cầu đi ra khỏi hệ thống.

Nhiệm vụ đầu tiên đặt ra là phải tính xác suất $p\{N(t) = 0\} \equiv p_0(t)$ và $p\{N(t) = 1\} \equiv p_1(t)$. Muốn vậy ta xét khoảng thời gian $(0, t + dt)$ như trên Hình 3-7.



Hình 3-7. Khoảng thời gian xét

Xét trường hợp $N(t) = 0$

Trước hết phải tính $p\{N(t + dt) = 0\}$. Ta nhận thấy rằng:

Nếu (Tại thời điểm $t + dt$ hệ thống không có yêu cầu nào) thì:

[(Tại thời điểm t hệ thống không có yêu cầu nào) VÀ (Trong khoảng thời gian $(t, t + dt)$ không có yêu cầu nào đến hệ thống)]

HOẶC

[(Tại thời điểm t có 1 yêu cầu trong hệ thống) VÀ (Trong khoảng thời gian $(t, t + dt)$ có một yêu cầu rời khỏi hệ thống)]

HOẶC

[(Tại thời điểm t hệ thống không có yêu cầu nào) VÀ (Trong khoảng thời gian $(t, t+dt)$ có 1 yêu cầu đến hệ thống) VÀ (Trong khoảng thời gian $(t, t+dt)$ có 1 yêu cầu rời khỏi hệ thống)]

Chúng ta có thể biểu diễn trường hợp trên bằng biểu thức toán học như sau:

$$p_0(t+dt) = p_0(t)p_0^i(t, t+dt) + \\ + p_1(t)p_1^o(t, t+dt) + p_0(t)p_1^i(t, t+dt)p_1^o(t, t+dt) \quad (\text{PT 3-21})$$

Theo định nghĩa của phân bố Markov, ta có:

$$p_0^i(t, t+dt) = 1 - \lambda dt; \quad (\text{PT 3-22})$$

$$p_1^o(t, t+dt) = \mu dt; \quad (\text{PT 3-23})$$

$$p_1^i(t, t+dt) = \lambda dt; \quad (\text{PT 3-24})$$

Thay vào (PT 3-21) ta có:

$$p_0(t+dt) = p_0(t)(1 - \lambda dt) + p_1(t)\mu dt + p_0(t)\lambda dt\mu dt \quad (\text{PT 3-25})$$

Số hạng thứ 3 của vế phải trong (PT 3-25) bằng 0 do $(dt)^2$ tiến tới 0 nhanh hơn dt . Do đó ta có:

$$p_0(t+dt) = p_0(t)(1 - \lambda dt) + p_1(t)\mu dt$$

Hoặc:

$$\frac{p_0(t+dt) - p_0(t)}{dt} = -p_0(t)\lambda + p_1(t)\mu; \quad (\text{PT 3-26})$$

Cuối cùng có:

$$\frac{dp_0(t)}{dt} = -p_0(t)\lambda + p_1(t)\mu; \quad (\text{PT 3-27})$$

Nhận xét rằng hệ thống đang xét là một hệ thống dừng, không phụ thuộc vào thời gian nên:

$$\frac{dp_0(t)}{dt} = 0; \quad (\text{PT 3-28})$$

Do đó (PT 3-27) trở thành:

$$p_0\lambda = p_1\mu; \quad (\text{PT 3-29})$$

Mặt khác:

$$p_0 + p_1 = 1; \quad (\text{PT 3-30})$$

Giải hệ phương trình (PT 3-29) và (PT 3-30) ta có:

$$\begin{cases} p_0 = \frac{\mu}{\mu + \lambda} \\ p_1 = \frac{\lambda}{\mu + \lambda} \end{cases}; \quad (\text{PT 3-31})$$

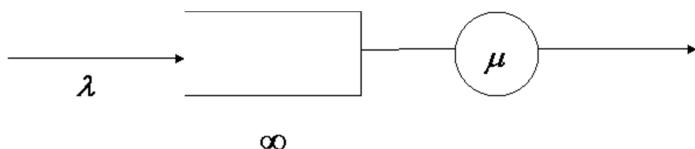
Từ (PT 3-31) ta nhận thấy rằng xác suất p_B để một yêu cầu bị từ chối chính là xác suất p_1 để yêu cầu đi vào hệ thống nhận thấy trạm phục vụ đang bận.

$$p_B = \frac{\lambda}{\mu + \lambda}; \quad (\text{PT 3-32})$$

Số yêu cầu trung bình trong một hệ thống:

$$N = \sum_{n=0}^1 np_n = p_1 = \frac{\lambda}{\mu + \lambda}; \quad (\text{PT 3-33})$$

3.5. Hàng đợi M/M/1/ ∞



Hình 3-8. Hàng đợi M/M/1/ ∞

Hệ thống hàng đợi M/M/1/ ∞ được đặc trưng bởi:

- Tiến trình tới tuân theo phân bố mũ với tham số là λ .
- Tiến trình phục vụ tuân theo phân bố mũ với tham số là μ .
- Một trạm phục vụ
- Hàng đợi có độ lớn là vô hạn, tức là không xảy ra trường hợp dịch vụ bị từ chối.

3.5.1. Xác suất có n yêu cầu trong hệ thống

Cũng như trường hợp hàng đợi M/M/1/1, đầu tiên chúng ta phải tính xác suất $p_n(t)$ để tại thời điểm t có n yêu cầu trong hệ thống hàng đợi. Cũng lý luận như trường hợp trên, chúng ta xét trong một khoảng thời gian $[0, t+dt]$ (Hình 3-7).

Mặt khác theo định nghĩa về phân bố Poisson, chúng ta cũng biết rằng một yêu cầu đến hệ thống trong khoảng thời gian $[t, t+dt]$ với xác suất

là $\lambda \cdot dt$, một yêu cầu rời khỏi hệ thống trong khoảng thời gian $[t, t+dt]$ với xác suất là $\mu \cdot dt$. Cũng giả thiết rằng xác suất xảy ra các sự kiện trong các khoảng thời gian không trùng nhau là độc lập.

- Với $n > 0$ xác suất $p_n(t + dt)$ được tính như sau:

$$\begin{aligned}
 p_n(t + dt) = & P\{n-1 \text{ yêu cầu trong hệ thống tại } t\} \times \\
 & P\{1 \text{ yêu cầu đến hệ thống trong } [t, t+dt]\} \times \\
 & P\{0 \text{ yêu cầu đi ra khỏi hệ thống trong } [t, t+dt]\} + \\
 & + P\{n \text{ yêu cầu trong hệ thống tại } t\} \times \\
 & P\{0 \text{ yêu cầu đến hệ thống trong } [t, t+dt]\} \times \\
 & P\{0 \text{ yêu cầu đi ra khỏi hệ thống trong } [t, t+dt]\} + \\
 & + P\{n \text{ yêu cầu trong hệ thống tại } t\} \times \\
 & P\{1 \text{ yêu cầu đến hệ thống trong } [t, t+dt]\} \times \\
 & P\{1 \text{ yêu cầu đi ra khỏi hệ thống trong } [t, t+dt]\} + \\
 & + P\{n+1 \text{ yêu cầu trong hệ thống tại } t\} \times \\
 & P\{0 \text{ yêu cầu đến hệ thống trong } [t, t+dt]\} \times \\
 & P\{1 \text{ yêu cầu đi ra khỏi hệ thống trong } [t, t+dt]\} ;
 \end{aligned}$$

Như vậy chúng ta có thể biểu diễn phương trình trên dưới dạng:

$$\begin{aligned}
 p_n(t + dt) = & p_{n-1}(t)\lambda dt(1 - \mu dt) + \\
 & + p_n(t)(1 - \lambda dt)(1 - \mu dt) + \\
 & + p_n(t)\lambda dt\mu dt + \\
 & + p_{n+1}(t)(1 - \lambda dt)\mu dt ; \quad (\text{PT 3-34})
 \end{aligned}$$

Các số hạng có chứa thành phần $(dt)^2$ có thể bỏ qua nên (PT 3-34) trở thành:

$$p_n(t + dt) = p_{n-1}(t)\lambda dt + p_n(t)(1 - \lambda dt - \mu dt) + p_{n+1}(t)\mu dt ; \quad (\text{PT 3-35})$$

Từ (PT 3-35):

$$\frac{p_n(t + dt) - p_n(t)}{dt} = \lambda p_{n-1}(t) - (\lambda + \mu)p_n(t) + \mu p_{n+1}(t) ;$$

Hay:

$$\frac{dp_n(t)}{dt} = \lambda p_{n-1}(t) - (\lambda + \mu)p_n(t) + \mu p_{n+1}(t) ; \quad (\text{PT 3-36})$$

- Với $n=0$, xác suất $p_0(t + dt)$ được tính tương tự như trong hệ thống M/M/1/1:

$$\frac{dp_0(t)}{dt} = -p_0(t)\lambda + p_1(t)\mu ; \quad (\text{PT 3-37})$$

Như vậy ta có hệ phương trình (PT 3-36) và (PT 3-37). Do hệ thống đang xét là hệ thống dừng, không phụ thuộc thời gian nên:

$$\frac{dp_n(t)}{dt} = 0; \forall n \in \{0, 1, \dots, \infty\}; \quad (\text{PT 3-38})$$

Từ (PT 3-36, 37, 38) ta có hệ phương trình:

$$\begin{cases} \mu p_1 = \lambda p_0 \\ (\lambda + \mu) p_n = \lambda p_{n-1} + \mu p_{n+1} \end{cases}; \quad (\text{PT 3-39})$$

Từ (PT 3-39) rút ra:

$$p_n = \left(\frac{\lambda}{\mu}\right)^n p_0 = \rho^n p_0; \quad (\text{PT 3-40})$$

Ở đây ρ là tải của hệ thống và được tính theo (PT 3-3) ($\rho \leq 1$). Với λ và μ cho trước, để tính p_n cần phải biết p_0 . Dựa vào điều kiện sau

$$\sum_{n=0}^{\infty} p_n = 1 \text{ ta có:}$$

$$\sum_{n=0}^{\infty} \rho^n p_0 = 1; \quad (\text{PT 3-41})$$

Hay

$$p_0 = \frac{1}{\sum_{n=0}^{\infty} \rho^n} = \frac{1}{1/(1-\rho)} = 1 - \rho; \quad (\text{PT 3-42})$$

Do đó:

$$p_n = \rho^n (1 - \rho); \quad (\text{PT 3-43})$$

3.5.2. Tính toán các tham số hiệu năng

- Đầu tiên chúng ta phải tính số yêu cầu trung bình trong hệ thống:

$$N = \sum_{n=0}^{\infty} n p_n = (1 - \rho) \sum_{n=0}^{\infty} n \rho^n = \frac{\rho}{1 - \rho};$$

Như vậy:

$$N = \frac{\rho}{1 - \rho}; \quad (\text{PT 3-44})$$

- Tiếp theo để tính số yêu cầu nằm trong hàng đợi trung bình, chúng ta để ý rằng nếu có n yêu cầu nằm trong hệ thống thì sẽ

có $(n-1)$ yêu cầu nằm trong hàng đợi và 1 yêu cầu đang được phục vụ bởi trạm phục vụ ($n \geq 1$), do đó:

$$\begin{aligned} N_q &= \sum_{n=1}^{\infty} (n-1)p_n = \sum_{n=1}^{\infty} np_n - \sum_{n=1}^{\infty} p_n = \sum_{n=0}^{\infty} np_n - (\sum_{n=0}^{\infty} p_n - p_0) = \\ &= N - (1 - p_0); \end{aligned} \quad (\text{PT 3-45})$$

Từ (PT-3-45) có:

$$N_q = \frac{\rho^2}{1-\rho}; \quad (\text{PT 3-46})$$

- Tính xác suất để một yêu cầu phải đợi trong hàng đợi một khoảng thời gian $t_q \leq t$ với t cho trước - $p(t_q \leq t)$. Chúng ta lý luận như sau:

Nếu một yêu cầu mới đến hệ thống khi không có yêu cầu nào đang nằm trong hệ thống, yêu cầu đó sẽ được phục vụ ngay, có nghĩa là $t_q = 0$:

$$p(t_q = 0) = p_0 = 1 - \rho; \quad (\text{PT 3-47})$$

Nếu một yêu cầu mới đến hệ thống và bắt gặp n yêu cầu đã ở trong hệ thống thì yêu cầu đó sẽ phải đợi n khoảng thời gian phục vụ với phân bố mũ trước khi nó được phục vụ.

$$t_q = S_1 + S_2 + \dots + S_n; \quad (\text{PT 3-48})$$

Với S_i là khoảng thời gian phục vụ yêu cầu thứ i , có phân bố mũ với tham số μ . Như vậy theo **Định lý 2-2**, hàm mật độ xác suất của t_q sẽ tuân theo phân bố Gamma, do đó xác suất $p(t_q \leq t | N = n)$ được tính như sau:

$$p(t_q \leq t | N = n) = \int_0^t \mu \frac{(\mu x)^{n-1}}{(n-1)!} e^{-\mu x} dx; \quad (\text{PT 3-49})$$

Như vậy ta có:

$$\begin{aligned} p(0 < t_q \leq t) &= \sum_{n=1}^{\infty} p(t_q \leq t | N = n) p_n = \\ &= \sum_{n=1}^{\infty} \int_0^t \mu \frac{(\mu x)^{n-1}}{(n-1)!} e^{-\mu x} dx \cdot (1 - \rho) \rho^n = \int_0^t \lambda e^{-\mu x} (1 - \rho) \sum_{n=1}^{\infty} \frac{(\lambda x)^{n-1}}{(n-1)!} dx = \\ &= \int_0^t \lambda e^{-\mu x} (1 - \rho) e^{\lambda x} dx = \frac{\lambda}{\mu} \int_0^t (\mu - \lambda) e^{-(\mu-\lambda)x} dx = \\ &= \rho [1 - e^{-(\mu-\lambda)t}]; \end{aligned}$$

Do đó:

$$p(0 < t_q \leq t) = \rho[1 - e^{-(\mu-\lambda)t}]; \quad (\text{PT 3-50})$$

Theo (PT 3-47) ta có:

$$p(t_q \leq t) = 1 - \rho + \rho[1 - e^{-(\mu-\lambda)t}];$$

Hay:

$$p(t_q \leq t) = 1 - \rho e^{-(\mu-\lambda)t}; \quad (\text{PT 3-51})$$

- Tính thời gian lưu lại trung bình của một yêu cầu trong hệ thống:

Theo định lý Little: $N = \lambda T$ hay $T = \frac{N}{\lambda}$; Theo (PT 3-44) ta có:

$$T = \frac{1}{\lambda} \times \frac{\rho}{1-\rho} = \frac{1}{\mu} \times \frac{1}{1-\rho}; \quad (\text{PT 3-52})$$

- Tính thời gian trung bình một yêu cầu phải đợi trong hàng đợi:

Cũng theo định lý Little ta có: $T_q = \frac{N_q}{\lambda}$; Theo (PT 3-46):

$$T_q = \frac{1}{\lambda} \times \frac{\rho^2}{1-\rho} = \frac{1}{\mu} \times \frac{\rho}{1-\rho}; \quad (\text{PT 3-53})$$

Nhận xét:

- Trong hệ thống hàng đợi M/M/1/ ∞ khi $\rho \rightarrow \infty$ thì tất cả các tham số hiệu năng đều tiến tới vô cùng. Có nghĩa là nếu $\rho \rightarrow \infty$ thì hệ thống hoạt động không ổn định.
- T_q có thể tính trực tiếp từ (PT 3-51) như sau:

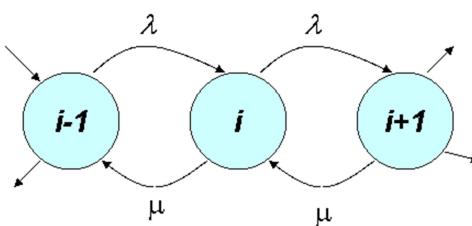
$$T_q = \int_0^{\infty} t \frac{d}{dt} p(t_q \leq t) dt; \quad (\text{PT 3-54})$$

Chú ý rằng $p(t_q \leq t)$ chính là hàm phân phối xác suất của t_q do đó $\frac{d}{dt} p(t_q \leq t)$ là hàm mật độ xác suất, vì vậy (PT 3-54) là kỳ vọng của t_q .

3.6. Tiết trình sinh – tử (Birth – Death Process)

Ta gọi một hệ thống hàng đợi đang ở trạng thái i nếu trong hệ thống đang có i yêu cầu. Nhìn vào các trạng thái của hệ thống hàng đợi M/M/1/1 và M/M/1/ ∞ , cụ thể là các phương trình xác suất (PT 3-28) và (PT 3-39) ta có nhận xét sau:

Nhận xét: Nếu hệ thống hàng đợi có tiết trình tới và tiết trình phục vụ đều tuân theo phân bố mũ thì hệ thống sẽ có tính chất cân bằng xác suất, nghĩa là xác suất hệ thống chuyển từ trạng thái i sang trạng thái $(i+1)$ hoặc $(i-1)$ sẽ bằng xác suất tổng xác suất hệ thống chuyển từ trạng thái $(i+1)$ hoặc $(i-1)$ về trạng thái i .



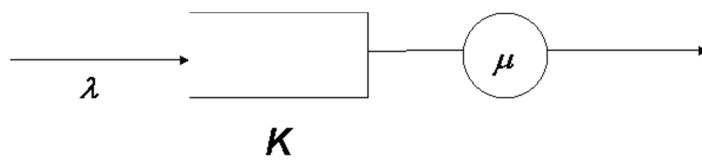
Hình 3-9. Cân bằng xác suất tại trạng thái i . Tiết trình sinh – tử

Nhận xét trên có thể được biểu diễn như trên Hình 3-9 hoặc

$$(\lambda + \mu)p_i = \lambda p_{i-1} + \mu p_{i+1}; \quad (\text{PT 3-55})$$

Trong đó p_j là xác suất để hệ thống có j yêu cầu. Quá trình sinh – tử và các tính chất của nó được trình bày chi tiết trong [13, 14, 19].

3.7. Hàng đợi M/M/1/K



Hình 3-10. Hệ thống M/M/1/K

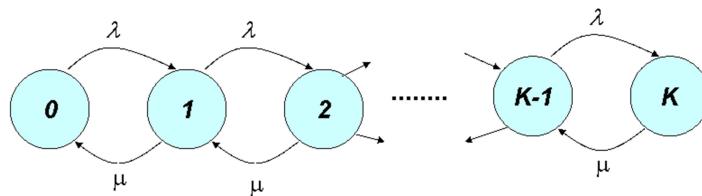
Hệ thống hàng đợi M/M/1/K được đặc trưng bởi:

- Tiết trình tới tuân theo phân bố mũ với tham số là λ .
- Tiết trình phục vụ tuân theo phân bố mũ với tham số là μ .
- Một trạm phục vụ.

- Hàng đợi có độ lớn K phần tử. Như vậy nếu một yêu cầu đi vào hàng đợi mà trong hàng đợi đã có K yêu cầu khác thì yêu cầu mới đến sẽ bị từ chối.

3.7.1. Xác suất có n yêu cầu trong hệ thống

Đầu tiên chúng ta biểu diễn hệ thống hàng đợi M/M/1/K bằng tiến trình sinh – tử (Hình 3-11).



Hình 3-11. Hệ thống hàng đợi M/M/1/K

Hệ phương trình cân bằng của hệ thống này được biểu diễn như sau:

$$\begin{cases} \lambda p_0 = \mu p_1 \\ \lambda p_{n-1} + \mu p_{n+1} = (\lambda + \mu) p_n; n \in \{1, \dots, K-1\} \\ \lambda p_{K-1} = \mu p_K; n = K \end{cases} \quad (\text{PT 3-56})$$

Từ hệ phương trình trên, ta được:

$$p_n = \rho^n p_0; n \in \{0, K\} \quad (\text{PT 3-57})$$

Mặt khác ta cũng có:

$$\sum_{n=0}^K p_n = 1;$$

Từ đó tính ra được:

$$p_0 = \frac{1}{\sum_{n=0}^K \rho^n} = \frac{1-\rho}{1-\rho^{K+1}}; \quad (\text{PT 3-58})$$

$$p_n = \frac{1-\rho}{1-\rho^{K+1}} \times \rho^n; \quad (\text{PT 3-59})$$

3.7.2. Tính toán các tham số hiệu năng

- Tính xác suất từ chối dịch vụ (xác suất mất yêu cầu)

Chúng ta nhận xét rằng một yêu cầu bị từ chối khi nó đến hàng đợi và thấy có K yêu cầu đang ở trong hàng đợi. Gọi xác suất để một yêu cầu bị từ chối dịch vụ là P_{loss} , ta nhận thấy rằng:

$$P_{loss} = p_K = \frac{1-\rho}{1-\rho^{K+1}} \times \rho^K; \quad n \in \{0, \dots, K\} \quad (\text{PT 3-60})$$

- Tính số yêu cầu trung bình trong hệ thống:

Ta có:

$$N = \sum_{n=1}^K np_n = \frac{1-\rho}{1-\rho^{K+1}} \times \sum_{n=1}^K n\rho^n; \quad (\text{PT 3-61})$$

Để tính được (PT 3-61) chúng ta chú ý đến phương trình sau:

$$f(\rho) = \sum_{n=0}^K \rho^n = \frac{1-\rho^{K+1}}{1-\rho}; \quad (\text{PT 3-62})$$

Lấy vi phân (PT 3-62):

$$f'(\rho) = \sum_{n=1}^K n\rho^{n-1} = \frac{1-(K+1)\rho^K + K\rho^{K+1}}{(1-\rho)^2}; \quad (\text{PT 3-63})$$

Thay (PT 3-63) vào (PT 3-61) ta có:

$$N = \frac{\rho}{1-\rho} - (K+1) \frac{\rho^{K+1}}{1-\rho^{K+1}}; \quad (\text{PT 3-64})$$

Chú ý rằng trong trường hợp $\rho \rightarrow 1$, khác với hàng đợi M/M/1/ ∞ , từ hệ phương trình (PT 3-56) ta có thể rút ra:

$$p_n = p_0 \text{ với } n=0, 1, \dots, K.$$

Do đó:

$$p_n = \frac{1}{K+1}; \quad n \in \{0, 1, \dots, K\} \quad (\text{PT 3-65})$$

Vậy trong trường hợp $\rho \rightarrow 1$ thay vào (PT 3.61):

$$N = \sum_{n=1}^K np_n = \frac{K}{2}; \quad (\text{PT 3-66})$$

- Tính số yêu cầu trung bình trong hàng đợi:

$$N_q = \sum_{n=1}^K (n-1)p_n = \sum_{n=1}^K np_n - \sum_{n=1}^K p_n = N - (1-p_0); \quad (\text{PT 3-67})$$

Cuối cùng ta có:

$$N_q = \frac{1}{1-\rho} + \frac{\rho - (K+1)\rho^{K+1}}{1-\rho^{K+1}}; \quad (\text{PT 3-68})$$

- Cũng như trong các hệ thống hàng đợi trước, ta tính phân bố xác suất của thời gian đợi t_q trong hàng đợi. Ở đây cần chú ý rằng, khi một yêu cầu tới hàng đợi và bắt gặp tổng cộng K yêu cầu đã nằm trong hàng đợi thì yêu cầu đó sẽ bị từ chối.

Như vậy đầu tiên phải tính xác suất q_n là xác suất có n yêu cầu đang ở trong hệ thống, với điều kiện $n \leq K-1$:

$$q_n \equiv P(N = n \mid n \leq K-1) = \frac{P_n}{P(n \leq K-1)} = \frac{p_n}{1-p_K}; \quad (\text{PT 3-69})$$

Do đó:

$$q_n = \frac{(1-\rho)\rho^n}{1-\rho^K}; \forall n = 0, 1, \dots, K-1; \quad (\text{PT 3-70})$$

Gọi xác suất để một yêu cầu phải đợi một khoảng thời gian $t_q \leq t$ và đang có n yêu cầu trong hệ thống ($n = 0, 1, \dots, K-1$) là: $p(t_q \leq t, N = n)$. Theo công thức xác suất có điều kiện:

$$\begin{aligned} p(t_q \leq t, N = n) &= p(t_q \leq t \mid N = n)P(N = n \mid n \leq K-1) \\ &= p(t_q \leq t \mid N = n)q_n; \end{aligned} \quad (\text{PT 3-71})$$

Như vậy, cũng lý luận như trong trường hợp hàng đợi M/M/1/∞ ta có:

$$p(t_q \leq t) = p(t_q = 0) + \sum_{n=1}^{K-1} p(t_q \leq t \mid N = n)q_n; \quad (\text{PT 3-72})$$

$$\Leftrightarrow p(T_q \leq t) = q_0 + \sum_{n=1}^{K-1} q_n \int_0^t \frac{\mu(\mu x)^{n-1}}{(n-1)!} e^{-\mu x} dx;$$

$$\Leftrightarrow p(t_q \leq t) = \sum_{n=0}^{K-1} q_n \int_0^t \frac{x^{n-1}}{(n-1)!} e^{-x} dx;$$

Cuối cùng ta tính được:

$$p(t_q \leq t) = \frac{1-\rho}{1-\rho^K} \sum_{n=0}^{K-1} \rho^n \int_0^t \frac{x^{n-1}}{(n-1)!} e^{-x} dx; \quad (\text{PT 3-73})$$

- Từ (PT 3-73) ta có thể tính thời gian trung bình mà các yêu cầu phải nằm chờ trong hàng đợi là:

$$T_q = \int_0^\infty t \frac{d}{dt} p(t_q \leq t) dt = \sum_{n=0}^{K-1} q_n \int_0^\infty t \mu \frac{(\mu t)^{n-1}}{(n-1)!} e^{-\mu t} dt; \quad (\text{PT 3-74})$$

Chú ý rằng:

$$\frac{1}{(n-1)!} \int_0^\infty x^n e^{-x} dx = \frac{n!}{(n-1)!} = n; \quad (\text{PT 3-75})$$

Do đó có thể được tính (kết hợp với (PT 3-69)):

$$T_q = \frac{1}{\mu} \sum_{n=0}^{K-1} n q_n = \frac{1}{\mu} \frac{1}{1-p_K} \sum_{n=0}^{K-1} n p_n = \frac{1}{\mu} \frac{1}{1-p_K} (N - K p_K); \quad (\text{PT 3-76})$$

Mặt khác theo (PT 3-67):

$$N_q = N - (1 - p_0) = \rho(N - K p_K);$$

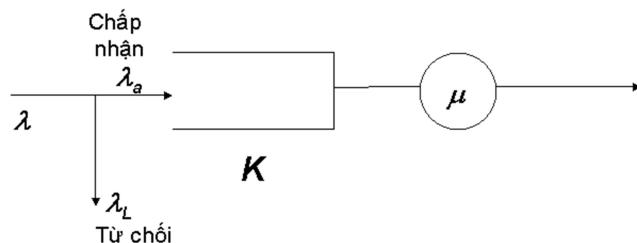
Cuối cùng ta có:

$$T_q = \frac{1}{\lambda} \frac{1}{1-p_K} N_q; \quad (\text{PT 3-77})$$

Nhận xét: Nếu gọi $\lambda_a \equiv \lambda(1 - p_K)$ là **tốc độ yêu cầu đi vào hàng đợi** thì (PT 3-77) tuân theo định lý Little:

$$T_q = \frac{N_q}{\lambda_a}; \quad (\text{PT 3-78})$$

Nếu quan sát hàng đợi M/M/1/K trên quan điểm một hệ thống đóng, chúng ta để ý rằng tại đầu vào hệ thống dòng yêu cầu đầu vào với tốc độ trung bình λ sẽ được chia thành hai nhánh: nhánh yêu cầu đi vào hàng đợi với tốc độ trung bình $\lambda_a = \lambda(1 - p_K)$ và nhánh các yêu cầu bị từ chối với tốc độ trung bình $\lambda_L = \lambda p_K$.

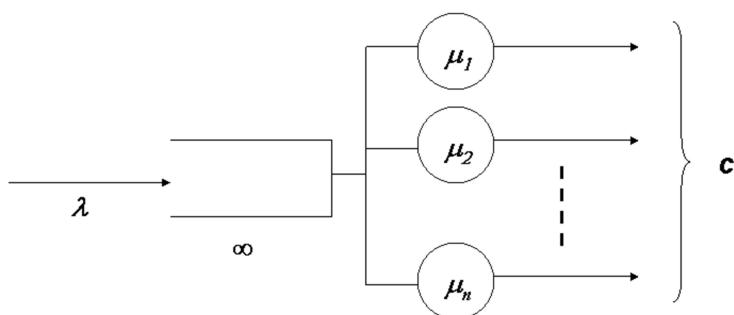


Hình 3-12. Hệ thống M/M/1/K trên quan điểm hệ thống đóng

- Cũng tính toán tương tự như trên, ta có thời gian trung bình T một yêu cầu lưu lại hệ thống:

$$T = \frac{1}{\lambda} \times \frac{1}{1 - p_K} \times N = \frac{N}{\lambda_a}; \quad (\text{PT 3-79})$$

3.8. Hàng đợi M/M/c/∞

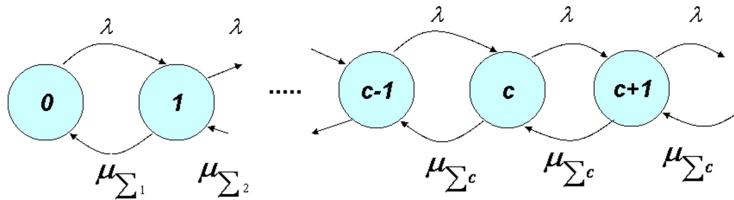


Hình 3-13 Hàng đợi M/M/c/∞

Hệ thống hàng đợi M/M/c/∞ có các đặc điểm sau:

- Tiến trình tới tuân theo phân bố mũ với tham số là λ .
 - Có c trạm phục vụ.
 - Tiến trình phục vụ của một trạm i bất kỳ tuân theo phân bố mũ với tham số là μ_i .
 - Hàng đợi có chiều dài vô tận.
 - Tốc độ đến của các sự kiện là
 - Tốc độ phục vụ của các server là với $i = 1, 2, \dots, c$. Để đơn giản, giả thiết
 - Mật độ lưu lượng :
 - Giả thiết thứ tự phục vụ của các server là từ 1, 2, 3, ..., c
 - Trong sơ đồ sinh-tử ?

3.8.1. Xác suất có n yêu cầu trong hệ thống



Hệ phương trình cân bằng chuyển đổi trạng thái:

$$\left\{ \begin{array}{l} \lambda p_0 = \mu p_1 \\ \lambda p_{n-1} + (n+1)\mu p_n = (\lambda + n\mu) p_n; n \in \{1, \dots, c-1\} \\ \lambda p_{n-1} + c\mu p_n = (\lambda + c\mu) p_n; n > c-1 \\ \sum_{n=0}^{\infty} p_n = 1 \end{array} \right\}$$

Đặt:

$$a = \frac{\lambda}{\mu} = c\rho$$

- Xác suất để có n sự kiện trong hệ thống

$$p_n = \frac{1}{n!} a^n p_0$$

Chương 4 Mạng hàng đợi

4.1. Mạng nối tiếp

Chương 5 Định tuyến trong mạng thông tin

5.1. Yêu cầu về định tuyến trong mạng thông tin

5.1.1. Vai trò của định tuyến trong mạng thông tin

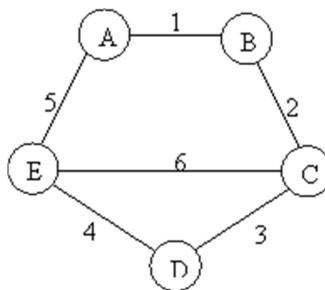
5.1.2. Các khái niệm trong lý thuyết graph

Phần này giới thiệu các thuật ngữ và các khái niệm cơ bản nhằm mô tả các mạng, graph, và các thuộc tính của nó. Lý thuyết graph là một môn học xuất hiện từ lâu, nhưng lý thuyết này có một số thuật ngữ được chấp nhận khác nhau dùng cho các khái niệm cơ bản. Vì thế có thể sử dụng một số thuật ngữ khác nhau để lập mô hình graph cho mạng. Các thuật ngữ được trình bày dưới đây này là các thuật ngữ đã được công nhận và được sử dụng thường xuyên chương này.

Một *graph* G , được định nghĩa bởi tập hợp các đỉnh V và tập hợp các cạnh E . Các đỉnh thường được gọi là các **nút** và chúng biểu diễn vị trí (ví dụ một điểm chứa lưu lượng hoặc một khu vực chứa thiết bị truyền thông). Các cạnh được gọi là các **liên kết** và chúng biểu diễn phương tiện truyền thông. Graph có thể được biểu diễn như sau:

$$G=(V, E)$$

Hình 4.1 là một ví dụ của một graph.



Hình 4.1. Một graph đơn giản

Mặc dù theo lý thuyết, V có thể là tập hợp rỗng hoặc không xác định, nhưng thông thường V là tập hợp xác định khác rỗng, nghĩa là có thể biểu diễn

$$V=\{v_i \mid i=1,2,\dots,N\}$$

Trong đó N là số lượng nút. Tương tự E được biểu diễn:

$$E=\{e_i \mid i=1,2,\dots,M\}$$

Một liên kết, e_j , tương ứng một kết nối giữa một cặp nút. Có thể biểu diễn một liên kết e_j giữa nút i và k bởi

Comment [TTH1]: Dinh tuyen tinh, flooding; random, hot potato (broadcast)

Dinh tuyen dong : MST, kruskal, MST tao 1 cay ngan nhat, va muc tieu lam gi

SPT la tao ra nhanh co khoang cach ngan nhat toi goc. -: dijkstra, bellman So sanh giua MST va spt: luc nao dung Ap dung vao dau;mst trong cac co che ve broadcasting, thi phai tao ra 1 cay mst, de khong tao ra loop neu ko se khong broadcast duoc.

MST: khong can nut goc

Spt: can nut goc, tao duong ngan nhat tu cac nut toi nut goc, vi the tao ra nhieu cay spt khac nhau

So sanh dijstra va bellman
Bellman di theo lop, di theo chang, di theo chieu ngan la di theo hop

Dijkstra la di theo chieu doc, moi an tim mot cay ngan nhat di tu goc, den khi tim duoc cay ngan hon thi quay lai chuyen

Can ca toan bo hieu biet tu topology trong mang, nen cac router khong can biet vvv. Bellman chi can biet distance giua cac mang

Convergence time cua cac routing protocol → complexity

Bellman co the tien toi vo cung → khong hoa tu, phan ki. Neu mot duong link bi dut

Dijkstra : can biet topo ve mang, va biet distance giua cac nut.

Noi ve ly thuyet thuat toan, sau do dua cac vi du,

Lay mot vi du ve giao thuc de xem thuat toan no se cu the the nao

So sanh MST va spt

Dij phu thuoc vao so nut N, bellman chay co the phan ky, khong phu thuoc vao so nut N

Lien quan toi giao thuc dinh tuyen hieenj nay, tai sao dung Ospf, tai sao khong dung Rip

Dua qua tinh complexity vao, nhung dua o muc don gian

$$e_j = (v_i, v_k)$$

hoặc bởi

$$e_j = (i, k)$$

Một liên kết gọi là **đi tới một nút** nếu nút đó là một trong hai điểm cuối của liên kết. Nút i và k gọi là **kề nhau** nếu tồn tại một liên kết (i, k) giữa chúng. Những nút như vậy được xem là các nút **láng giềng**. Độ của nút là số lượng liên kết đi tới nút hay là số lượng nút láng giềng. Hai khái niệm trên là tương đương nhau trong các graph thông thường. Tuy nhiên với các graph có nhiều hơn một liên kết giữa cùng một cặp nút, thì hai khái niệm trên là không tương đương. Trong trường hợp đó, độ của một nút được định nghĩa là số lượng liên kết đi tới nút đó.

Một liên kết có thể có hai hướng. Khi đó thứ tự của các nút là không có ý nghĩa. Ngược lại thứ tự các nút có ý nghĩa. Trong trường hợp thứ tự các nút có ý nghĩa, một liên kết có thể được xem như là một **cung** và được định nghĩa

$$a_j = [v_i, v_k]$$

hoặc đơn giản hơn

$$a_j = [i, k]$$

k được gọi là **cận kè hướng ra** đối với i nếu một cung $[i, k]$ tồn tại và **bậc hướng ra** của i là số lượng các cung như vậy. Khái niệm **cận kè hướng vào** và **bậc cận kè hướng vào** cũng được định nghĩa tương tự.

Một graph gọi là một **mạng** nếu các liên kết và các nút có mặt trong liên kết có các thuộc tính (chẳng hạn như độ dài, dung lượng, loại...). Các mạng được sử dụng để mô hình các vấn đề cần quan tâm trong truyền thông, các thuộc tính riêng biệt của nút và liên kết thì liên quan đến các vấn đề cụ thể trong truyền thông.

Sự khác nhau giữa các liên kết và các cung là rất quan trọng cả về việc lập mô hình cho mạng lẫn quá trình hoạt động bên trong của các thuật toán, vì vậy sự khác nhau cần phải luôn được phân biệt rõ ràng. Về mặt hình học các liên kết là các đường thẳng kết nối các cặp nút còn các cung là các đường thẳng có mũi tên ở một đầu, biểu diễn chiều của cung.

Một graph có các liên kết gọi là **graph vô hướng**, tuy nhiên một graph có các cung gọi là **graph hữu hướng**. Một graph hữu hướng có thể có cả các liên kết vô hướng. Thông thường, các graph được giả sử là vô hướng, hoặc sự phân biệt đó là không có ý nghĩa.

Có thể có khả năng xảy ra hiện tượng xuất hiện nhiều hơn một liên kết giữa cùng một cặp nút (điều này tương ứng với việc có nhiều kênh thông tin giữa hai chuyển mạch). Những liên kết như vậy được gọi là các **liên kết song song**. Một graph có liên kết song song gọi là một **multigraph**.

Cũng có khả năng xuất hiện các liên kết giữa một nút nào đó và chính nút đó. Những liên kết đó được gọi là các **self loop**. Chúng ít khi xuất hiện và thường xuất hiện do việc xem hai nút như là một nút trong quá trình lập mô hình graph cho một mạng hoặc phát sinh trong quá trình thực hiện một thuật toán có việc hợp nhất các nút. Hình 4.2 minh họa một graph có các liên kết song song và các self loop. Một graph không có các liên kết song song hoặc các self loop gọi là một **graph đơn giản**. Việc biểu diễn và vận dụng các graph đơn giản là tương đối dễ dàng, vì vậy giả thiết rằng các graph được xem xét là các graph đơn giản. Nếu có sự khác biệt với giả thiết này, chúng sẽ được chỉ ra.

5.2. Các mô hình định tuyến quảng bá (broadcast routing)

5.2.1. Lan tràn gói (flooding)

Một dạng mạnh hơn của định tuyến riêng biệt đó là lan tràn gói. Trong phương thức này, mỗi gói đi đến router sẽ được gửi đi trên tất cả các đường ra trừ đường mà nó đi đến. Phương thức lan tràn gói này hiển nhiên là tạo ra rất nhiều gói sao chép (duplicate). Trên thực tế, số gói này là không xác định trừ khi thực hiện một số biện pháp để hạn chế quá trình này.

Một trong những biện pháp đó là sử dụng bộ đếm bước nhảy trong phần tiêu đề của mỗi gói. Giá trị này sẽ bị giảm đi một tại mỗi bước nhảy. Gói sẽ bị loại bỏ khi bộ đếm đạt giá trị không. Về mặt lý tưởng, bộ đếm bước nhảy sẽ có giá trị ban đầu tương ứng với độ dài từ nguồn đến đích. Nếu như người gửi không biết độ dài của đường đi, nó có thể đặt giá trị ban đầu của bộ đếm cho trường hợp xấu nhất. Khi đó giá trị ban đầu đó sẽ được đặt bằng đường kính của mạng con.

Một kỹ thuật khác để ngăn sự lan tràn gói là thêm số thứ tự vào tiêu đề các gói. Mỗi router sẽ cần có một danh sách theo nút nguồn để chỉ ra những số thứ tự từ nguồn đó đã được xem xét. Để tránh danh sách phát triển không giới hạn, mỗi danh sách sẽ tăng lên bởi số đếm k để chỉ ra rằng tất cả các số thứ tự đến k đã được xem. Khi một gói đi tới, rất dễ dàng có thể kiểm tra được gói là bản sao hay không. Nếu đúng gói là bản sao thì gói này sẽ bị loại bỏ.

Lan tràn gói có ưu điểm là lan tràn gói luôn luôn chọn đường ngắn nhất. Có được ưu điểm này là do về phương diện lý thuyết nó chọn tất cả các đường có thể do đó nó sẽ chọn được đường ngắn nhất. Tuy nhiên nhược điểm của nó là số lượng gói gửi trong mạng quá nhiều.

Sử dụng lan tràn gói trong hầu hết các ứng dụng là không thực tế. Tuy vậy lan tràn gói có thể sử dụng trong những ứng dụng sau.

- Trong ứng dụng quân sự, mạng sử dụng phương thức lan tràn gói để giữ cho mạng luôn luôn hoạt động tốt khi đối mặt với quân địch.
- Trong những ứng dụng cơ sở dữ liệu phân bố, đôi khi cần thiết phải cập nhật tất cả cơ sở dữ liệu. Trong trường hợp đó sử dụng lan tràn gói là cần thiết. Ví dụ sử dụng lan tràn gói để gửi

cập nhật bản định tuyến bởi vì cập nhật không dựa trên độ chính xác của bảng định tuyến.

- Phương pháp lan tràn gói có thể được dùng như là đơn vị để so sánh phương thức định tuyến khác. Lan tràn gói luôn luôn chọn đường ngắn nhất. Điều đó dẫn đến không có giải thuật nào có thể tìm được độ trễ ngắn hơn.

Một biến đổi của phương pháp lan tràn gói là lan tràn gói có chọn lọc. Trong giải thuật này, router chỉ gửi gói đi ra trên các đường mà đi theo hướng đích. Điều đó có nghĩa là không gửi gói đến những đường mà rõ ràng nằm trên hướng sai.

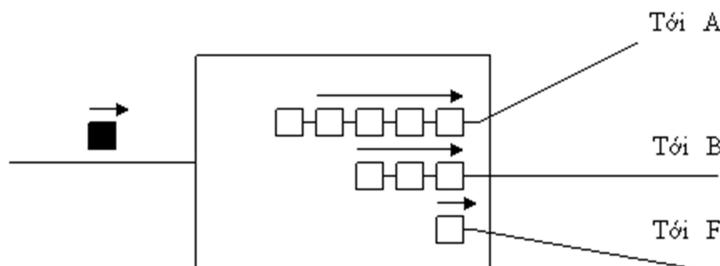
5.2.2. Định tuyến bước ngẫu nhiên (random walk)

Trong phương pháp định tuyến này, router sẽ chuyển gói đi đến trên một đường đầu ra được chọn một cách ngẫu nhiên. Mục tiêu của phương pháp này là các gói lang thang trong mạng cuối cùng cũng đến đích. Với phương pháp này giúp cho quá trình cân bằng tải giữa các đường. Cũng giống như phương pháp định tuyến lan tràn gói, phương pháp này luôn đảm bảo là gói cuối cùng sẽ đến đích. So với phương pháp trước thì sự nhân rộng gói trong mạng sẽ ít hơn. Nhược điểm của phương pháp này là đường từ nguồn đến đích có thể dài hơn đường ngắn nhất. Do đó trễ đường truyền sẽ dài hơn sẽ trễ ngắn nhất thực sự tồn tại trong mạng.

5.2.3. Định tuyến khoai tây nóng (hot potato)

Định tuyến riêng biệt là loại định tuyến mà router quyết định tuyến đi chỉ dựa vào thông tin bản thân nó lượm lặt được.

Đây là một thuật toán tương thích riêng biệt (isolated adaptive algorithm). Khi một gói đến một nút, router sẽ cố gắng chuyển gói đó đi càng nhanh càng tốt bằng cách cho nó vào hàng chờ đầu ra ngắn nhất. Nói cách khác, khi có gói đi đến router sẽ tính toán số gói được nằm chờ để truyền trên mỗi đường đầu ra. Sau đó nó sẽ gán gói mới vào cuối hàng chờ ngắn nhất mà không quan tâm đến đường đó sẽ đi đâu. Hình 5-1 biểu diễn các hàng chờ đầu ra bên trong một router tại một thời điểm nào đó. Có ba hàng chờ đầu ra tương ứng với 03 đường ra. Các gói đang xếp hàng trên mỗi đường để chờ được truyền đi. Trong ví dụ ở đây, hàng chờ đến F là hàng chờ ngắn nhất với chỉ có một gói nằm trên hàng chờ này. Giải thuật khoai tây nóng do đó sẽ đặt gói mới đến vào hàng chờ này.



Hình 5-1. Hàng chờ bên trong router

Có thể biến đổi ý tưởng này một chút bằng cách kết hợp định tuyến tĩnh với giải thuật khoai tây nóng. Khi gói đi đến, router sẽ tính đến cả những trọng số tĩnh của đường dây và độ dài hàng chờ. Một khả năng là sử dụng lựa chọn tĩnh tốt nhất trừ khi độ dài hàng chờ lớn hơn một ngưỡng nào đó. Một khả năng khác là sử dụng độ dài hàng chờ ngắn nhất trừ trọng số tĩnh của nó là quá thấp. Còn một cách khác là sắp xếp các đường theo trọng số tĩnh của nó và sau đó lại sắp xếp theo độ dài hàng chờ của nó. Sau đó sẽ chọn đường có tổng vị trí sắp xếp là nhỏ nhất. Dù giải thuật nào được chọn đi chăng nữa cũng có đặc tính là khi ít tải thì đường có trọng số cao nhất sẽ được chọn, nhưng sẽ làm cho hàng chờ cho đường này tăng lên. Sau đó một số lưu lượng sẽ được chuyển sang đường ít tải hơn.

5.2.4. Định tuyến nguồn (source routing) và mô hình cây (spanning tree)

Chúng ta sẽ xét một số thuật toán cơ bản dùng cho việc tìm kiếm các cây được sử dụng để thiết kế và phân tích mạng. Một cây là một graph không có các vòng; bất kỳ một cặp nút nào cũng chỉ có duy nhất một đường đi. ở đây chủ yếu xem xét các graph vô hướng, những graph đó có các liên kết được sử dụng cả hai chiều trong quá trình tạo ra các đường đi.

Vì một số lý do, các cây rất hữu dụng và được sử dụng như là graph cơ bản cho các thuật toán và các kỹ thuật phân tích và thiết kế mạng. Thứ nhất, các cây là mạng tối thiểu; cung cấp một sự kết nối mà không có một liên kết nào là không cần thiết. Thứ hai, do việc chỉ cung cấp duy nhất một đường đi giữa một cặp nút bất kỳ, các cây giải quyết các vấn đề về định tuyến (nghĩa là quyết định việc chuyển lưu lượng giữa hai nút). Điều đó làm đơn giản mạng và dạng của nó. Tuy nhiên, vì các cây liên thông tối thiểu nên cũng đơn giản và có độ tin cậy tối thiểu. Đó là nguyên nhân tại sao các mạng thực tế thường có tính liên thông cao hơn. Chính vì vậy, việc thiết kế một mạng thường bắt đầu bằng một cây.

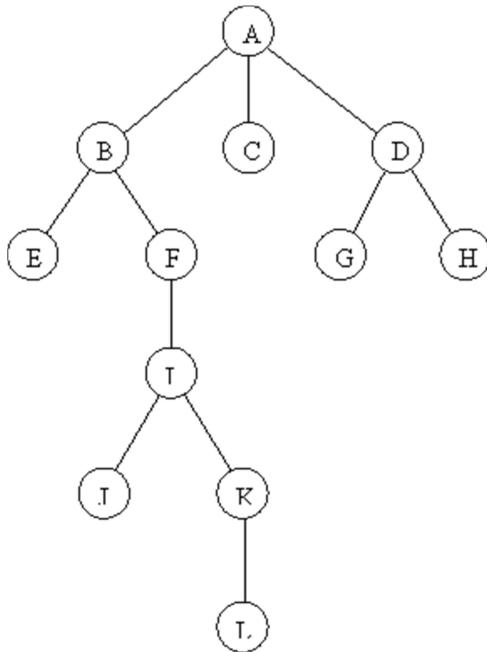
5.2.5. Duyệt cây

Cho trước một cây nào đó, chúng ta có thể đi tới mọi nút của nó. Quá trình đó gọi là một quá trình duyệt cây. Trong quá trình thực hiện, các cạnh trong cây được duyệt hai lần, mỗi lần theo một hướng khác nhau. Có nhiều cách duyệt khác nhau. Đầu tiên, chỉ ra một nút của cây làm nút gốc. Việc duyệt được thực hiện xoay quanh nút đó. Có một số điều kiện để lựa chọn nút gốc này (chẳng hạn nút gốc là một khu vực máy tính trung tâm). Ngoài ra, nút gốc có thể được chọn một cách ngẫu nhiên.

Giả sử nút A trong hình 4.1 là nút gốc của cây. Từ A chúng ta có thể lần lượt đi tới các nút kề cận của nó như là B, C hoặc D. Sau đó, lại đi theo các nút kề cận của chúng (B, C và D) là E, F, G và H. Tiếp tục đi tới lần lượt các nút kề cận khác bên cạnh các nút này. Khi đó, việc

duyệt này sẽ kết thúc khi tới các nút I , J , K và L . Quá trình này được gọi là tìm kiếm theo chiều rộng. Trong quá trình tìm kiếm theo chiều rộng một đặc điểm cần chú ý là những nút gần nút gốc nhất sẽ được tới trước. Việc tìm kiếm sẽ thực hiện theo mọi hướng cùng lúc. Điều đó đôi khi có ích và được thực hiện dễ dàng.

Một thuật toán nhằm đi tới mọi nút của cây thì được gọi là thuật toán duyệt cây. Thuật toán sau đây, *Bfintree*, thực hiện một quá trình tìm kiếm theo chiều rộng. (Chúng ta quy ước rằng, các tên hàm có ký tự đầu tiên là ký tự hoa để phân biệt chúng với các tên biến). *Bfintree* sẽ sử dụng một danh sách kề cận n_adj_list , danh sách này liệt kê tất cả các nút kề cận của mỗi nút thuộc cây. Để đơn giản hơn, giả sử rằng cây này là một cây hữu hướng hướng ra nhìn từ gốc và do đó n_adj_list sẽ chỉ bao gồm các nút kề cận với một nút nào đó mà các nút kề cận đó xa gốc hơn so với nút đang xét.



Hình 5-2. Duyệt cây

```

void <-BfsTree ( n, root, n_adj_list ):
dcl n_adj_list [n, list ]
scan_queue [queue ]

InitializeQueue (scan_queue )
Enqueue( root, scan_queue )
while (NotEmpty(scan_queue))
    node <- Dequeue (scan_queue)
    Visit(node )
    for each (neighbor , n_adj_list [node ])
  
```

Enqueue (*neighbor*, *scan_queue*)

Visit là một thủ tục trong đó thực hiện một số quá trình nào đó đối với mỗi nút (chẳng hạn như in lên màn hình các thông tin của mỗi nút .v.v).

Thuật toán này được thực hiện cùng một hàng đợi. Hàng đợi là một FIFO; trong đó các phần tử được thêm vào từ phía sau hàng đợi và chuyển ra từ phía trước. Các thủ tục *InitializeQueue*, *Enqueue*, *Dequeue*, *NotEmpty* làm việc trên các hàng đợi. *InitializeQueue* thiết lập một hàng đợi rỗng. *Enqueue*, *Dequeue* là các thủ tục để thêm một phần tử vào cuối hàng đợi và chuyển một phần tử ra từ đầu hàng đợi. Hàm *NotEmpty* trả về TRUE hoặc FALSE tùy thuộc vào hàng đợi có rỗng hay không.

n_adj_list là một chuỗi mà mỗi phần tử của chuỗi là một danh sách. *n_adj_list[n]* là một danh sách các nút kề cận nút *n*. Như đã nói ở chương trước, *for_each(element, list)*, là một cấu trúc điều khiển thực hiện vòng lặp đối với tất cả các phần tử của *list* và thực hiện các mã ở bên trong vòng lặp, trong vòng lặp đó các phần tử của *list* lần lượt được sử dụng. Thủ tục trên hoạt động với giả thiết là *n_adj_list* đã được thiết lập trước khi thủ tục *BfsTree* được gọi.

Tương tự, ta có thể định nghĩa một quá trình tìm kiếm theo chiều sâu. Quá trình này cũng bắt đầu từ nút gốc. Quá trình duyệt tiếp tục thực hiện nút láng giềng chưa được duyệt của nút vừa mới được duyệt. Ta cũng giả sử rằng cây bao gồm các liên kết có hướng đi ra xa nút gốc.

Ví dụ 4.1:

Trở lại với graph trong hình 4.1, ta có thể tới nút *B* từ nút *A*. Sau đó, ta tới nút *E*, kề cận với nút *B*-nút được duyệt gần thời điểm hiện tại nhất. Nút *E* này không có nút kề cận chưa duyệt nào, do vậy ta phải quay trở lại nút *B* để đi sang nút *F*. Ta tiếp tục đi tới các nút *I*, *J*, *K* (cùng với việc quay lại nút *I*), và nút *L*. Sau đó ta quay trở về nút *A*, tiếp tục tới các nút còn lại là *C*, *D*, *G* và *H*. Do vậy, toàn bộ quá trình duyệt là:

A, B, E, F, I, J, K, L, C, D, G, H

Nhớ rằng thứ tự của quá trình duyệt là không duy nhất. Trong quá trình duyệt trên ta chọn các nút kề cận để xâm nhập theo thứ tự từ trái qua phải. Nếu chọn theo thứ tự khác, quá trình duyệt là:

A, B, F, I, J, K, L, E, D, H, G, C

Trật tự thực tế của quá trình duyệt phụ thuộc vào từng thuật toán cụ thể. Điều này cũng đúng với một quá trình tìm kiếm theo chiều rộng. Kiểm tra thuật toán *BfsTree*, trật tự này là một hàm của trật tự các nút kề cận trong *n_adj_list*.

Thuật toán *DfsTree* sau sẽ thực hiện một quá trình tìm kiếm theo chiều sâu.

```
void <- DfsTree(n, root, n_adj_list):
    dcl n_adj_list [n, list]
```

```

Visit(root)
for each(neighbor, n_adj_list[node])
    DfsTree(n, neighbor, n_adj-list)

```

Quá trình tìm kiếm này sẽ được thực hiện với sự trợ giúp của một ngăn xếp theo kiểu LIFO, nghĩa là phần tử được thêm vào và chuyển ra từ đỉnh ngăn xếp. Trong trường hợp này, chúng ta thường gọi đệ quy *DfsTree*, thực tế chúng ta đã sử dụng ngăn xếp hệ thống, nghĩa là sử dụng loại ngăn xếp mà hệ thống sử dụng để lưu giữ các lời gọi hàm và đổi số.

Cả hai loại duyệt trình bày ở trên đều là quá trình duyệt thuận (nghĩa là các quá trình này duyệt một nút rồi sau đó duyệt tới nút tiếp theo của nút đó). Quá trình duyệt ngược đôi khi cũng rất cần thiết, trong quá trình duyệt ngược một nút được duyệt sau khi đã duyệt nút tiếp của nút đó. Dĩ nhiên, cũng có thể thành lập một danh sách thuận và sau đó đảo ngược danh sách đó. Cũng có thể thay thế trật tự tìm kiếm một cách trực tiếp như thủ tục sau:

```

void <- PostorderDfsTree(n, root, n_adj_list):
    dcl n_adj_list [n, list]

    for each(neighbor, n_adj_list[node])
        PostorderDfsTree(n, neighbor,
    n_adj_list)
    Visit (root)

```

Các thành phần liên thông trong các graph vô hướng

Ta có thể áp dụng khái niệm duyệt các nút vào một graph vô hướng, đơn giản chỉ bằng cách theo dõi các nút đã được duyệt và sau đó không duyệt các nút đó nữa.

Có thể duyệt một graph vô hướng như sau:

```

void <- Dfs(n, root, n_adj_list):
    dcl n_adj_list [n, list]
    visited [n]

    void <- DfsLoop (node)
        if (not(visited [node]))
            visited [node]<-TRUE
            visit [node]
    for each(neighbor, n_adj_list[node])
        DfsLoop (neighbor)
    visited <-FALSE
    DfsLoop (root)

```

Chú ý rằng câu lệnh

Visited <- FALSE

khởi tạo toàn bộ các phần tử mảng được duyệt bằng FALSE. Cũng cần chú ý rằng thủ tục *DfsLoop* được định nghĩa bên trong thủ tục *Dfs* nên *DfsLoop* có thể truy cập tới *visited* và *n_adj_list* (Lưu ý rằng cách dễ nhất để đọc các giả mã cho các hàm có dạng hàm *Dfs* ở trên là trước tiên hãy đọc thân của hàm chính rồi quay trở lại đọc thân của các hàm nhúng như hàm *DfsLoop*).

Chú ý rằng trong quá trình duyệt chúng ta đã ngầm kiểm tra tất cả các cạnh trong graph, một lần cho mỗi đầu cuối của mỗi cạnh. Cụ thể, với mỗi cạnh (i, j) của graph thì j là một phần tử của *n_adj_list[i]* và i là một thành phần trong *n_adj_list[j]*. Thực tế, có thể đưa chính các cạnh đó vào các danh sách kề cận của nó và sau đó tìm nút ở điểm cuối khác của cạnh đó bằng hàm:

```
node <- OtherEnd(node1, edge)
```

Hàm này sẽ trả về một điểm cuối của edge khác với node1. Điều đó làm phức tạp quá trình thực hiện đôi chút. Có thể dễ dàng thấy rằng độ phức tạp của các thuật toán duyệt cây này bằng $O(E)$, với E là số lượng cạnh trong graph.

Bây giờ chúng ta có thể tìm được các thành phần liên thông của một graph vô hướng bằng cách duyệt mỗi thành phần. Chúng ta sẽ đánh dấu mỗi nút bằng một chỉ số thành phần khi chúng ta tiến hành. Các biến *n_component* sẽ theo dõi bất kỳ thành phần nào mà chúng ta đi tới

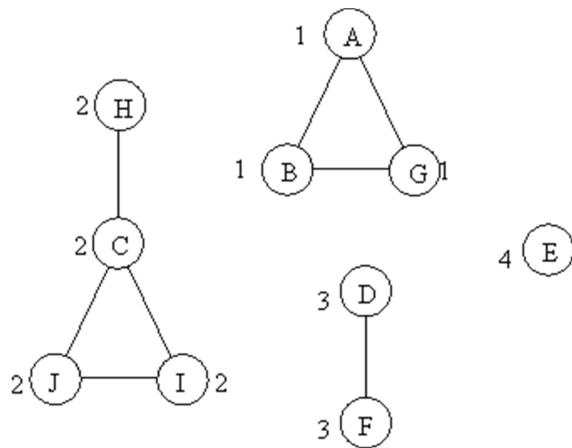
```
void <- LabelComponent (n, n_adj_list):
    dcl n_component_number [n],
    n_adj_list[n,list]

    void <- Visit [node]
        n_component_number [node] <- ncomponents

    n_component_number <- 0
    ncomponent <- 0
    for each (node, node_set)
        if (n_component_number [node] = 0)
            ncomponent += 1
            Dfs (node, n_adj_list)
```

Chúng ta định nghĩa một hàm *Visit* để thiết lập một chỉ số thành phần các nút được duyệt. Hàm này nằm bên trong thủ tục *LabelComponent* và chỉ có thể được gọi từ trong thủ tục đó. Mặt khác, *Dfs* còn được định nghĩa ở bên ngoài, vì thế nó có thể được gọi từ bất kỳ đâu.

Trong khi thực hiện quá trình duyệt theo chiều rộng và chiều sâu một graph vô hướng, những cạnh nối một nút với một nút láng giềng chưa duyệt trước khi duyệt nút đó tạo ra một cây, nếu graph là không liên thông thì tạo ra một rừng.



Hình 5-3. Các thành phần

Hình 5-3 biểu diễn một graph có 4 thành phần. Giả sử vòng trên tập các nút đi theo tuần tự alphabet, các thành phần được đánh số theo trật tự các nút có chữ cái "thấp nhất" và chỉ số thành phần được biểu diễn ở bên cạnh nút.

Với mỗi thành phần, thuật toán trên sẽ gọi *Dfs* để kiểm tra thành phần đó. Trong đó, thuật toán cũng kiểm tra các cạnh, mỗi cạnh một lần. Vì thế, độ phức tạp của nó có bậc bằng bậc của tổng số các nút cộng với số các cạnh trong tất cả các thành phần (nghĩa là độ phức tạp của thuật toán bằng $O(N+E)$).

Cây bắc cầu tối thiểu (*Minimum Spanning Tree*)

Có thể sử dụng *Dfs* để tìm một cây bắc cầu nếu có một cây bắc cầu tồn tại. Cây tìm được thường là cây vô hướng. Việc tìm cây "tốt nhất" thường rất quan trọng. Chính vì vậy, chúng ta có thể gắn một "độ dài" cho mỗi cạnh trong graph và đặt ra yêu cầu tìm một cây có độ dài tối thiểu. Thực tế, "độ dài" có thể là khoảng cách, giá, hoặc là một đại lượng đánh giá độ trễ hoặc độ tin cậy. Một cây có tổng giá là tối thiểu được gọi là cây bắc cầu tối thiểu.

Nói chung, nếu graph là một graph không liên thông, chúng ta có thể tìm được một rừng bắc cầu tối thiểu. Một rừng bắc cầu tối thiểu là một tập hợp các cạnh nối đến graph một cách tối đa có tổng độ dài là tối thiểu. Bài toán này có thể được xem như là việc lựa chọn một graph con của graph gốc chứa tất cả các nút của graph gốc và các cạnh được lựa chọn. Đầu tiên, tạo một graph có n nút, n thành phần và không có cạnh nào cả. Mỗi lần, chúng ta chọn một cạnh để thêm vào graph này hai thành phần liên thông trước đó chưa được kết nối được liên kết lại với nhau tạo ra một thành phần liên thông mới (chứ không chọn các cạnh thêm vào một thành phần liên thông trước đó và tạo ra một vòng). Vì vậy, tại bất kỳ giai đoạn nào của thuật toán, quan hệ:

$$n=c+e$$

luôn được duy trì, ở đây n là số lượng nút trong graph, e là số cạnh được lựa chọn tính cho tới thời điểm xét và c là số lượng thành phần trong graph tính cho tới thời điểm xét. Ở cuối thuật toán, e bằng n trừ đi số thành phần trong graph gốc; nếu graph gốc là liên thông, chúng ta sẽ tìm được một cây có $(n-1)$ cạnh. Như đã giải thích ở trên, Dfs sẽ tìm ra một rừng bắc cầu. Tuy nhiên, chúng ta thường không tìm được cây bắc cầu có tổng độ dài tối thiểu.

Thuật toán "háu ăn"

Một cách tiếp cận khả dĩ để tìm một cây có tổng độ dài tối thiểu là, ở mỗi giai đoạn của thuật toán, lựa chọn cạnh ngắn nhất có thể. Thuật toán đó gọi là thuật toán "háu ăn". Thuật toán này có tính chất "thiến cận" nghĩa là không lường trước được các kết quả cuối cùng do các quyết định mà chúng đưa ra ở mỗi bước gây ra. Thay vào đó, chúng chỉ đưa ra cách chọn tốt nhất cho mỗi quá trình lựa chọn. Nói chung, thuật toán "háu ăn" không tìm được lời giải tối ưu cho một bài toán. Thực tế thuật toán thậm chí còn không tìm được một lời giải khả thi ngay cả khi lời giải đó tồn tại. Tuy nhiên chúng hiệu quả và dễ thực hiện. Chính vì vậy chúng được sử dụng rộng rãi. Các thuật toán này cũng thường tạo cơ sở cho các thuật toán có tính hiệu quả và phức tạp hơn.

Vì thế, câu hỏi đầu tiên đặt ra khi xem xét việc ứng dụng một thuật toán để giải quyết một bài toán là liệu bài toán ấy có hay không cầu trúc nào đó đảm bảo cho thuật toán hoạt động tốt. Hy vọng rằng thuật toán ít ra cũng đảm bảo được một lời giải khả thi nếu lời giải đó tồn tại. Khi đó, nó sẽ đảm bảo tính tối ưu và đảm bảo yêu cầu nào đó về thời gian thực hiện. Bài toán tìm các cây bắc cầu tối thiểu thực sự có một cấu trúc mạnh cho phép thuật toán "háu ăn" đảm bảo cả tính tối ưu cũng như đảm bảo độ phức tạp tính toán ở mức độ vừa phải.

Dạng chung của thuật toán "háu ăn" là:

Bắt đầu bằng một lời giải rỗng s .

Trong khi vẫn còn có các phần tử cần xét,

Tìm e , phần tử "tốt nhất" vẫn chưa xét

Nếu việc thêm e vào s là khả thi thì e được thêm vào s , nếu việc thêm đó không khả thi thì loại bỏ e .

Các yêu cầu các khả năng sau:

- So sánh giá trị của các phần tử để xác định phần tử nào là "tốt nhất"
- Kiểm tra tính khả thi của một tập các phần tử

Khái niệm "tốt nhất" liên quan đến mục đích của bài toán. Nếu mục đích là tối thiểu, "tốt nhất" nghĩa là bé nhất. Ngược lại, "tốt nhất" nghĩa là lớn nhất.

Thường thường, mỗi giá trị gắn liền với một phần tử, và giá trị gắn liền với một tập đơn giản chỉ là tổng các giá trị đi cùng của các phần tử trong tập đó. Đó là trường hợp cho bài toán cây bắc cầu tối thiểu được xét trong phần này. Tuy nhiên, đó không phải là trường hợp chung. Chẳng hạn, thay cho việc tối thiểu tổng độ dài của tất cả các cạnh trong một cây, mục đích của bài toán là tối thiểu hoá độ dài các cạnh dài nhất trong cây. Trong trường hợp đó, giá trị của một cạnh là độ dài của cạnh đó và giá trị của một tập sẽ là độ dài của cạnh dài nhất nằm trong tập.

Muốn tìm được cạnh "tốt nhất" để bổ sung, hãy đánh giá các cạnh theo độ ảnh hưởng về giá trị của nó tới giá trị của tập. Giả sử $V(S)$ là giá trị của tập S và $v(e, S)$ là giá trị của một phần tử e thì $v(e, S)$ có quan hệ với tập S bởi công thức

$$v(e, S) = V(S \cup e) - V(S)$$

Trong trường hợp tối thiểu độ dài của cạnh dài nhất trong một cây. $v(e, S)$ bằng 0 đối với bất kỳ cạnh nào không dài hơn cạnh dài nhất đã được chọn. Ngược lại, nó sẽ bằng hiệu độ dài giữa cạnh với cạnh dài nhất đã được chọn, khi hiệu đó lớn hơn 0.

Trong trường hợp chung, giá trị của tập có thể thay đổi một cách ngẫu nhiên khi các phần tử được bổ sung vào nó. Chúng ta có thể gán giá trị 1 cho các tập có số lượng phần tử là chẵn và 2 cho các tập có số lượng phần tử là lẻ. Điều đó làm cho các giá trị của các phần tử chỉ là một trong hai giá trị +1 và -1. Trong trường hợp này, thuật toán "háu ăn" không được sử dụng. Bây giờ giả sử rằng "trọng lượng" của một tập biến đổi theo một cách hợp lý hơn thì khi đó, sẽ có một cơ sở hợp lý hơn cho việc chỉ ra phần tử "tốt nhất". Một điều quan trọng cần chú ý đó là, khi tập lớn lên, giá trị của phần tử mà trước đó không được xem xét có thể thay đổi do các phần tử thêm vào tập đó. Khi điều này xảy ra, thuật toán "háu ăn" có thể mắc lỗi trong các lựa chọn của nó và sẽ ảnh hưởng tới chất lượng của lời giải mà chúng ta nhận được.

Tương tự, trong hầu hết các trường hợp, tính khả thi có thể bị ảnh hưởng một cách ngẫu nhiên do sự bổ sung phần tử. Chính vì vậy, trong các bài toán mà những tập có số lượng phần tử chẵn có thể được xem là khả thi và những tập có số phần tử là lẻ có thể được xem là không khả thi thì thuật toán "háu ăn" hoặc bất kỳ thuật toán nào có bổ sung các phần tử, mỗi lần một phần tử, sẽ không hoạt động. Vì vậy chúng ta sẽ giả thiết các tính chất sau, những tính chất này luôn được duy trì trong mọi trường hợp xem xét:

Tính chất 1:

Bất kỳ một tập con nào của một tập khả thi thì cũng khả thi, đặc biệt tập rỗng cũng là một tập khả thi.

Ngoài ra giả thiết rằng độ phức tạp của thuật toán để tính toán giá trị của một tập và kiểm tra sự khả thi của chúng là vừa phải, đặc biệt, khi độ phức tạp này là một đa thức của số nút và cạnh trong graph.

```

list<-Greedy (properties)
dcl properties [list, list]
candidate_set[list]
solution[list]

void<-GreedyLoop ( *candidate_set,
*solution)
dcl test_set[list],solution[list],
candidate_set[list]
element <-
SelectBestElement(candidate_set)
test_set <-Append(element,solution)
if(Test(test_set))
    solution<-test_set
candidate_set<-
Delete(element,candidate_set)
if(not(Empty(candidate_set)))
    Greedy_loop(*candidate_set,
*solution)

candidate_set<-ElementsOf(properties)
solution<-φ
if(! (Empty(element_set)))
    GreedyLoop(*candidate_set, *solution)
return(solution)

```

Bây giờ ta đã có thể xem xét sâu hơn các câu lệnh của thuật toán "háu ăn". Các câu lệnh của thuật toán hơi khó hiểu vì chúng dựa trên định nghĩa của hai hàm, *Test* và *SelectBestElement* (là hàm kiểm tra tính khả thi và đánh giá các tập). Chúng ta cũng giả sử rằng có một cấu trúc *properties*, là một danh sách của các danh sách chứa tất cả các thông tin cần thiết để kiểm tra và đánh giá tất cả các tập. Một danh sách của các danh sách đơn giản chỉ là một danh sách liên kết, mà mỗi thành viên của nó là một danh sách. Thậm chí cấu trúc đó có thể được lồng vào nhau sâu hơn, nghĩa là có các danh sách nằm bên trong các danh sách nằm bên trong các danh sách. Cấu trúc như vậy tương đối phổ biến và có thể được sử dụng để biểu diễn hầu hết các kiểu thông tin. Có thể lưu giữ độ dài, loại liên kết, dung lượng, hoặc địa chỉ. Bản thân các mục thông tin này có thể là một cấu trúc phức tạp; nghĩa là cấu trúc đó có thể lưu giữ giá và các dung lượng của một vài loại kenh khác nhau cho mỗi liên kết.

Trên thực tế, điều đó rất có ích cho việc duy trì các cấu trúc dữ liệu trợ giúp để cho phép thuật toán thực hiện hiệu quả hơn. Bài toán về cây bắc cầu tối thiểu là một ví dụ. Tuy nhiên, để rõ ràng, giả sử rằng tất cả quá trình tính toán được thực hiện trên một cấu trúc *properties* sẵn có (đã được khởi tạo). ϕ được sử dụng để biểu diễn tập rỗng. *Append* và *Delete* là các hàm bổ sung và chuyển đi một phần tử khỏi một danh sách. *ElementsOf* chỉ đơn giản để chỉ ra các phần tử của một danh sách; vì vậy, ban đầu tất cả các phần tử trong *properties* là các ứng

cử. Có rất nhiều cách thực hiện các quá trình này. *properties* có thể là một dãy và các hàm *Append*, *Delete* và *ElementsOf* có thể hoạt động với các danh sách chỉ số (danh sách mà các phần tử là các chỉ số mạng). Trong thực tế cách thực hiện được chọn là cách làm sao cho việc thực hiện các hàm *Test* và *SelectBestElement* là tốt nhất.

Đoạn giả mã trên giả thiết rằng thuật toán "háu ăn" sẽ dùng lại khi không còn phần tử nào để xem xét. Trong thực tế, có nhiều nguyên nhân để thuật toán dừng lại. Một trong những nguyên nhân là khi kết quả xấu đi khi các phần tử được tiếp tục thêm vào. Điều này xảy ra khi tất cả các phần tử còn lại đều mang giá trị âm trong khi chúng ta đang cố tìm cho một giá trị tối đa. Một nguyên nhân khác là khi biết rằng không còn phần tử nào ở trong tập ứng cử có khả năng kết hợp với các phần tử vừa được chọn tạo ra một lời giải khả thi. Điều này xảy ra khi một cây bắc cầu toàn bộ các nút đã được tìm thấy.

Giả sử rằng thuật toán dừng lại khi điều đó là hợp lý, còn nếu không, các phần tử không liên quan sẽ bị loại ra khỏi lời giải.

Giả thiết rằng, các lời giải cho một bài toán thoả mãn tính chất 1 và giá trị của tập đơn giản chỉ là tổng các giá trị của các phần tử trong tập. Ngoài ra, giả thiết thêm rằng tính chất sau được thoả mãn:

Tính chất 2:

Nếu hai tập S_p và S_{p+1} lần lượt có p và $p+1$ phần tử là các lời giải và tồn tại một phần tử e thuộc tập S_{p+1} nhưng không thuộc tập S_p thì $S_p \cup \{e\}$ là một lời giải.

Chúng ta thấy rằng, các cạnh của các rùng thoả mãn tính chất 2, nghĩa là nếu có hai rùng, một có p cạnh và rùng kia có $p+1$ thì luôn tìm được một cạnh thuộc tập lớn hơn mà việc thêm cạnh đó vào tập nhỏ hơn không tạo ra một chu trình.

Một tập các lời giải thoả mãn các tính chất trên gọi là một matroid. Định lý sau đây là rất quan trọng (chúng ta chỉ thừa nhận chứ không chứng minh).

Định lý 4.1

Thuật toán "háu ăn" đảm bảo đảm một lời giải tối ưu cho một bài toán khi và chỉ khi các lời giải đó tạo ra một matroid.

Có thể thấy rằng, tính chất 1 và tính chất 2 là điều kiện cần và đủ để đảm bảo tính tối ưu của thuật toán "háu ăn". Nếu có một lời giải cho một bài toán nào đó mà nó thoả mãn hai tính chất 1 và 2 thì cách đơn giản nhất là dùng thuật toán "háu ăn" để giải quyết nó. Điều đó đúng với một cây bắc cầu.

Sau đây là một định lý không kém phần quan trọng.

Định lý 4.2

Nếu các lời giải khả thi cho một bài toán nào đó tạo ra một matroid thì tất cả các tập khả thi tối đa có số lượng phần tử như nhau.

Trong đó, một tập khả thi tối đa là một tập mà khi thêm các phần tử vào thì tính khả thi của nó không được bảo toàn; Nó không nhất thiết phải có số lượng phần tử tối đa cũng như không nhất thiết phải có trọng lượng lớn nhất.

Định lý đảo của định lý trên cũng có thể đúng nghĩa là nếu tính chất 1 được thoả mãn và mọi tập khả thi tối đa có cùng số lượng phần tử, thì tính chất 2 được thoả mãn.

Định lý 4.2 cho phép chúng ta chuyển đổi một bài toán tối thiểu P thành một bài toán tối đa P' bằng cách thay đổi các giá trị của các phần tử. Giả thiết rằng tất cả $v(x_j)$ trong P có giá trị âm. Lời giải tối ưu cho bài toán P có số lượng phần tử tối đa là m thì chúng ta có thể tạo ra một bài toán tối đa P' từ P bằng cách thiết lập các giá trị của các phần tử trong P' thành $-v(x_j)$. Tất cả các phần tử đều có giá trị dương và P' có một lời giải tối ưu chứa m phần tử. Thực ra, thứ tự của các lời giải tối đa phải được đảo lại: lời giải có giá trị tối đa trong P' cũng là lời giải có giá trị tối thiểu trong P .

Giả sử lúc nay ta cần tìm một lời giải có giá trị tối thiểu, tuân theo điều kiện là có số lượng tối đa các phần tử. Sẽ tính cả các phần tử có giá trị dương. Có thể giải quyết bài toán P như là một bài toán tối đa P' bằng cách thiết lập các giá trị của các phần tử thành $B-v(x_j)$ với B có giá trị lớn hơn giá trị lớn nhất của x_j . Khi đó các giá trị trong P' đều dương và P' là một lời giải tối ưu có m phần tử. Thứ tự của tất cả các tập khả thi tối đa đã bị đảo ngược: một tập có giá trị là V trong P thì có giá trị là $mB-V$ trong lời giải P' . Một giá trị tối đa trong P' thì có giá trị tối thiểu trong P . Quy tắc này cũng đúng với các cây bắc cầu thoả mãn tính chất 1 và tính chất 2 và có thể tìm một cây bắc cầu tối thiểu bằng cách sử dụng một thuật toán “háu ăn”.

Thuật toán Kruskal

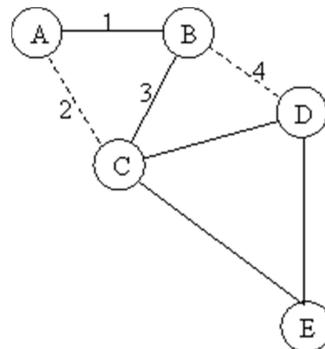
Thuật toán Kruskal là một thuật toán “háu ăn” được sử dụng để tìm một cây bắc cầu tối thiểu. Tính đúng đắn của thuật toán dựa trên các định lý sau:

Định lý 4.3

Các rừng thì thoả mãn tính chất 1 và 2.

Như chúng ta đã biết, một rừng là một tập hợp các cạnh mà tập hợp đó không chứa các chu trình. Rõ ràng là bất kỳ một tập con các cạnh nào của một rừng (thậm chí cả tập rỗng) cũng là một rừng, vì vậy tính chất 1 được thoả mãn.

Để thấy rằng tính chất 2 cũng thoả mãn, xét một graph được biểu diễn trong hình 4.4.



Hình 4.3.

Giả sử có một rừng F_1 có p cạnh. Rừng $\{2, 4\}$ là một ví dụ với $p=2$, và nó được biểu diễn bằng nét đứt trong hình 4.4. Khi đó xét một rừng khác F_2 có $p+1$ cạnh. Có hai trường hợp được xét.

Trường hợp 1: F_2 đi tới một nút n , nhưng F_1 không đi tới nút đó. Một ví dụ của trường hợp này là rừng $\{1, 4, 6\}$, rừng này đi tới E còn F_1 thì không. Trong trường hợp này, có thể tạo ra rừng $\{2, 4, 6\}$ bằng cách thêm cạnh 6 vào rừng $\{2, 4\}$.

Trường hợp 2: F_2 chỉ đi tới các nút mà F_1 đi tới. Một ví dụ của trường hợp này là rừng $\{1, 4, 5\}$. Xét S , một tập các nút mà F_1 đi tới. Cho rằng có k nút trong tập S . Vì F_1 là một rừng nên mỗi cạnh trong F_1 giảm số lượng thành phần trong S đi một, do đó tổng số lượng thành phần là $k-p$. Tương tự, F_2 tạo ra $k-(p+1)$ thành phần từ S (số lượng thành phần vừa nói bé hơn với số lượng thành phần của F_1). Vì vậy, một cạnh tồn tại trong F_2 mà các điểm cuối của nó nằm ở các thành phần khác nhau trong F_1 thì có thể thêm cạnh đó vào F_1 mà không tạo ra một chu trình. Cạnh 3 là một cạnh có tính chất đó trong ví dụ này (cạnh 1 và 5 cũng là những cạnh như vậy).

Vì thế, chúng ta thấy rằng nếu tính chất 1 và 2 được thoả mãn thì một thuật toán “háu ăn” có thể tìm được một lời giải tối ưu cho cả bài toán cây bắc cầu tối thiểu lẫn bài toán cây bắc cầu tối đa. Chú ý rằng một cây bắc cầu là một rừng có số cạnh tối đa $N-1$ cạnh với N là số nút trong mạng. Sau đây chúng ta sẽ xét bài toán tối thiểu.

Thuật toán Kruskal thực hiện việc sắp xếp các cạnh với cạnh đầu tiên là cạnh ngắn nhất và tiếp theo chọn tất cả các cạnh mà những cạnh này không cùng với các cạnh được lựa chọn trước đó tạo ra các chu trình. Chính vì thế, việc thực hiện thuật toán đơn giản là:

```

list <- kruskal_l( n, m, lengths )
dcl length[m], permutation[m],
solution[list]

permutation <- VectorSort( n , lengths )
solution <- Φ
for each ( edge , permutation )
    if ( Test(edge , solution ) )
        solution <- Append ( edge , solution )
return( solution )

```

VectorSort có đầu vào là một vector có độ dài là n và kết quả trả về là thứ tự sắp xếp các số nguyên từ 1 tới n . Sự sắp xếp đó giữ cho giá trị tương ứng trong vector theo thứ tự tăng dần.

Ví dụ 4.2:

Giả sử rằng $n=5$ và giá trị của một vector là

31, 19, 42, 66, 27

VectorSort sẽ trả về thứ tự sắp xếp như sau:

2, 5, 1, 3, 4

Test nhận một danh sách các cạnh và trả về giá trị TRUE nếu các cạnh đó không chứa một chu trình. Vì Test được gọi cho mỗi nút, sự hiệu quả của toàn bộ thuật toán tuỳ thuộc vào tính hiệu quả của việc thực hiện Test. Nếu mỗi khi các cạnh được thêm vào cây, chúng ta theo dõi được các nút của cạnh thuộc các thành phần nào thì Test trở nên đơn giản; đó đơn giản chỉ là việc kiểm tra xem các nút cuối của các cạnh đang được xét có ở cùng một thành phần không. Nếu cùng, cạnh sẽ tạo ra một chu trình. Ngược lại, cạnh đó không tạo nên chu trình.

Tiếp đó là xem xét việc duy trì cấu trúc thành phần. Có một số cách tiếp cận. Một trong các cách đó là ở mỗi nút duy trì một con trỏ đến một nút khác trong cùng một thành phần và có một nút ở mỗi thành phần gọi là nút gốc của thành phần thì trỏ vào chính nó. Vì thế lúc đầu, bản thân mỗi nút là một thành phần và nó trỏ vào chính nó. Khi một cạnh được thêm vào giữa hai nút i và j , trỏ i tới j . Sau đó, khi một cạnh được thêm vào giữa một nút i trong một thành phần có nút gốc là k và một nút j trong một thành phần có nút gốc là l thì trỏ k tới l . Vì vậy, chúng ta có thể kiểm tra một cạnh bằng cách dựa vào các con trỏ từ các nút cuối của nó và xem rằng chúng có dẫn đến cùng một nơi hay không. Chuỗi các con trỏ càng ngắn, việc kiểm tra càng dễ dàng. Nhằm giữ cho các chuỗi các con trỏ đó ngắn, Tarjan gợi ý nên làm gọn các chuỗi khi chúng được duyệt trong quá trình kiểm tra. Cụ thể, ông gợi ý một hàm *FindComponent* được tạo ra như sau:

```

index <- FindComponent (node , *next)
dcl next []

```

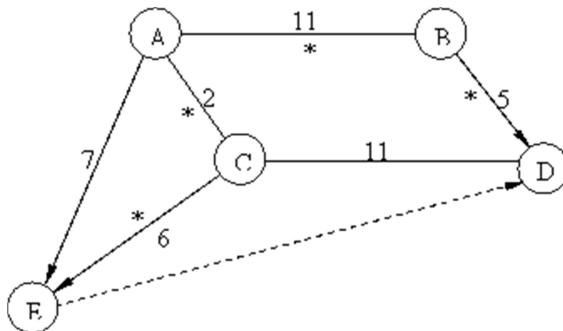
```

p=next [node]
q=next [p]
while ( p!=q )
    next [node]= q
    node = q
p=next [node]
q=next [p]
return (p)

```

FindComponent trả về nút gốc của thành phần chứa *node*. Hàm này cũng điều chỉnh *next*, nút hướng về nút gốc chứa nút đó. Đặc biệt, hàm này điều chỉnh *next* hướng tới điểm ở tầng cao hơn. Tarjan chỉ ra rằng, bằng cách đó, thà làm gọn đường đi tới nút gốc một cách hoàn toàn còn hơn là không làm gọn một chút nào cả và toàn bộ kết quả trong việc tìm kiếm và cập nhật *next* chỉ lớn hơn so với $O(n+m)$ một chút với n là số lượng nút và m là số lượng cạnh được kiểm tra.

Ví dụ 4.3:



Hình 5-4. Phép tính Minimum Spanning Tree (MST)

Xét một mạng được biểu diễn trong hình 4.4. các dấu * trong hình được giải thích dưới đây. Đầu tiên, sắp xếp các cạnh và sau đó lần lượt xem xét từng cạnh, bắt đầu từ cạnh nhỏ nhất. Vì thế, chúng ta xem (A, C) là cạnh đầu tiên. Gọi *FindComponent* cho nút A ta thấy cả *p* lẫn *q* đều là A nên *FindComponent* trả về A như là nút gốc của thành phần chứa nút A. Tương tự, *FindComponent* trả về C như là nút gốc của thành phần chứa nút C. Vì thế, chúng ta mang A và C vào cây và thiết lập *next[A]* bằng C. Sau đó, xét (B, D). Hàm cũng thực hiện tương tự và B, D được thêm vào cây, *next[B]* bằng D. Chúng ta xét (C, E), chấp nhận nó và thiết lập *next[C]* bằng E.

Bây giờ, xét (A, E). Trong *FindComponent*, *p* là C còn *q* là E. Vì thế chúng ta chạy vào vòng lặp *while*, thiết lập *next[A]* bằng E và rút ngắn đường đi từ A tới E với E là nút gốc của thành phần chứa chúng. *Node, p* và *q* được thiết lập thành E và *FindComponent* trả về E như là nút gốc của thành phần chứa nút A. *FindComponent* cũng trả về E

như là nút gốc của thành phần chứa E . Vì thế, cả hai điểm cuối của (A, E) là cùng một thành phần nên (A, E) bị loại bỏ.

Tiếp đến, xét (A, B) . Trong quá trình gọi *FindComponent* đối với nút A , chúng ta thấy rằng $p=q=E$ và *next* không thay đổi. Tương tự, quá trình gọi *FindComponent* đối với nút B ta được $p=q=D$. Vì thế, chúng ta thiết lập *next*[E] bằng D . Chú ý rằng, chúng ta không thiết lập *next*[A] bằng B , mà lại thiết lập *next* đối với nút gốc của thành phần của A bằng với nút gốc của thành phần của B .

Cuối cùng, (C, D) được kiểm tra và bị loại bỏ.

Trong hình 4.4 những cạnh trong cây bắc cầu được phân biệt bởi một dấu * ở ngay bên cạnh các cạnh đó. Nội dung các *next* được chỉ ra bằng các cung (các cạnh hữu hướng) có mũi tên. Chẳng hạn, *next*[B] bằng D được chỉ ra bằng một mũi tên từ B tới D . Chú ý rằng, các cung được định nghĩa bởi *next* tạo ra một cây, nhưng nói chung cây đó không phải là một cây bắc cầu tối thiểu. Thực vậy, với trường hợp có một cung (E, D) , ngay cả khi các cung đó không cần thiết phải là một phần graph. Vì vậy, bản thân *next* chỉ định nghĩa cấu trúc thành phần khi tiến hành thực hiện thuật toán. Chúng ta tạo một danh sách hiện các cạnh được chọn dành cho việc bao gồm trong cây. Giá của cây được định nghĩa bởi *next* tương đối bằng phẳng, nghĩa là các đường đi tới các nút gốc của các thành phần là ngắn khiến *FindComponent* hoạt động hiệu quả.

Hiển nhiên, sự phức tạp của thuật toán Kruskal được quyết định bởi việc sắp xếp các cạnh, sự sắp xếp đó có độ phức tạp là $O(m \log m)$. Nếu có thể tìm được cây bắc cầu trước khi phải kiểm tra tất cả các cạnh thì chúng ta có thể cải tiến quá trình đó bằng cách thực hiện sắp xếp phân đoạn. Cụ thể, chúng ta có thể lưu giữ các cạnh trong một khối (heap) và sau đó lấy ra, kiểm tra mỗi cạnh cho đến khi một cây được tạo ra. Chúng ta dễ dàng biết được quá trình đó dừng vào lúc nào; chỉ đơn giản là theo dõi số lượng cạnh đã được xét và dừng lại khi đã có $n-1$ cạnh được chấp nhận.

Chúng ta giả sử rằng, các quá trình quản lý khói (heap) như thiết lập, bổ xung và lấy dữ liệu ra là đơn giản. Điều quan trọng cần chú ý ở đây là độ phức tạp của việc thiết lập một khói (heap) có m phần tử là $O(m)$, độ phức tạp của việc tìm phần tử bé nhất là $O(1)$ và độ phức tạp của việc khôi phục một khói (heap) sau khi bổ xung, xoá, hoặc thay đổi một giá trị là $O(\log m)$. Chính vì vậy, nếu chúng ta xét k cạnh để tìm cây bắc cầu, độ phức tạp trong việc duy trì một khói (heap) bằng $O(m+k\log m)$, độ phức tạp này bé hơn $O(m\log m)$ nếu k có bậc bé hơn bậc của m . k tối thiểu bằng $O(n)$ nên nếu graph là khá mỏng thì việc sử dụng khói (heap) sẽ không có lợi. Nếu graph là dày đặc thì việc lưu trữ đó có thể được xem xét. Đây là phiên bản cuối cùng của thuật toán Kruskal, thuật toán này tận dụng các hiệu ứng nói trên.

```

list <- Kruskal_l( n, m, lengths )
dcl length[m], ends[m,2], next[n],
solution[list],           l_heap[heap]

for each ( node , n )
    next[node]<-node

l_heap <-HeapSet(m, lengths)
#_accept <-0
solution <-  $\emptyset$ 

while ( (#_accept < n-1) and
! (HeapEmpty(l_heap))
    edge <- HeapPop(*l_heap)
    c1=FindComponent(ends[edge,1], *next)
    c2=FindComponent(ends[edge,2], *next)
    if (c1 !=c2 )
next[c2] <- c1
solution <- Append ( edge , solution )
    #_accept=#_accept+1
return( solution )

```

HeapSet tạo ra một khối (heap) dựa vào các giá trị cho trước và trả về chính khói (heap) đó. *HeapPop* trả về chỉ số của giá trị ở đỉnh của khói (heap) chứ không phải bản thân giá trị đó. Điều này có lợi hơn việc trả về một giá trị vì từ chỉ số luôn biết được giá trị có chỉ số đó chứ từ giá trị không thể biết được chỉ số của giá trị đó. Cũng cần chú ý rằng *HeapPop* làm khói (heap) thay đổi. *HeapEmpty* trả về giá trị TRUE nếu khói (heap) rỗng. Mảng *ends* chứa các điểm cuối của các cạnh.

Thuật toán Prim

Thuật toán này có những ưu điểm riêng biệt, đặc biệt là khi mạng dày đặc, trong việc xem xét một bài toán tìm kiếm các cây bắc cầu tối thiểu (MST). Hơn nữa các thuật toán phức tạp hơn được xây dựng dựa vào các thuật toán MST này; và một số các thuật toán này hoạt động tốt hơn với các cấu trúc dữ liệu được sử dụng cho thuật toán sau đây, thuật toán này được phát biểu bởi Prim. Tóm lại, các thuật toán này phù hợp với các quá trình thực hiện song song bởi vì các quá trình đó được thực hiện bằng các toán tử vector. Thuật toán Prim có thể được miêu tả như sau:

Bắt đầu với một nút thuộc cây còn tất cả các nút khác không thuộc cây (ở ngoài cây).

Trong khi còn có các nút không thuộc cây

Tìm nút không thuộc cây gần nhất so với cây

Đưa nút đó vào cây và ghi lại cạnh nối nút đó với cây

Thuật toán Prim dựa trên những định lý sau đây:

Định lý 4.4.

Một cây là một MST nếu và chỉ nếu cây đó chứa cạnh ngắn nhất trong mọi cutset chia các nút thành hai thành phần.

Để thực hiện thuật toán Prim, cần phải theo dõi khoảng cách từ mỗi nút không thuộc cây tới cây và cập nhật khoảng cách đó mỗi khi có một nút được thêm vào cây. Việc đó được thực hiện dễ dàng; đơn giản chỉ là duy trì một dãy `d_tree` có các thông tin về khoảng cách đã nói ở trên. Quá trình đó tuân theo:

```
array[n] <- Prim( n , root , dist )
dcl dist[n,n] , pred[n], d_tree[n],
in_tree[n]

index <- FindMin()
d_min <- INFINITY
for each( i , n )
    if(! (in_tree[j]) and (d_tree[i]<
d_min))
        i_min <- i
        d_min <- d_tree[i]
return ( i_min )

void <- Scan(i)
    for each ( j , n )
        if(! (in_tree[j]) and
(d_tree[j]>dist[i,j]))
            d_tree[j]<- dist[i,j]
            pred[j]<-i

    d_tree <- INFINITY
    pred <- -1
    in_tree <- FALSE
    d_tree(root)<-0
    #_in_tree <-0
    while ( #_in_tree < n)
        i <- FindMin()
        in_tree[i]<- TRUE
Scan(i)
#_in_tree =#_in_tree + 1
return (pred)
```

FindMin trả về một nút không thuộc cây và gần cây nhất. *Scan* cập nhật khoảng cách tới cây đối với các nút không thuộc cây.

Có thể thấy rằng độ phức tạp của thuật toán này là $O(n^2)$; cả hai hàm *FindMin* và *Scan* có độ phức tạp là $O(n)$ và mỗi hàm được thực hiện n lần. So sánh với thuật toán Kruskal ta thấy rằng độ phức tạp của thuật toán Prim tăng nhanh hơn so với độ phức tạp của thuật toán Kruskal nếu m , số lượng các cạnh, bằng $O(n^2)$, còn nếu m có cùng bậc với n thì độ phức tạp của thuật toán Kruskal tăng nhanh hơn.

Có thể tăng tốc thuật toán Prim trong trường hợp graph là một graph mỏng bằng cách chỉ quan tâm đến các nút láng giềng của nút i vừa được thêm vào cây. Nếu sẵn có các thông tin kè liền, vòng lặp *for* trong *Scan* có thể trở thành.

```
for each (j , n_adj_list[i] )
```

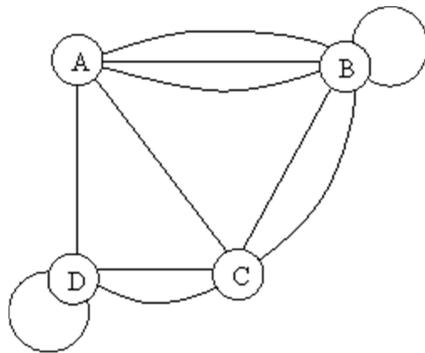
Độ phức tạp của *Scan* trở thành $O(d)$ với d là bậc của nút i . Chính vì thế độ phức tạp tổng cộng của *Scan* giảm từ $O(n^2)$ xuống $O(m)$.

Thiết lập một tập kè liền cho toàn bộ một graph là một phép toán có độ phức tạp bằng $O(m)$:

```
index[nn,list] <- SetAdj (n ,m, ends)
dcl ends[m,2], n_adj_list[n,list]

for node = 1 to n
    n_adj_list[node] <- Φ
for edge = 1 to m
    Append(edge, n_adj_list[end[edge,1]])
    Append(edge, n_adj_list[end[edge,2]])
```

Có thể tăng tốc *FindMin* nếu ta thiết lập một khối (heap) chứa các giá trị trong d_tree . Vì thế, chúng ta có thể lấy ra giá trị thấp nhất và độ phức tạp tổng cộng của quá trình lấy ra là $O(n \log n)$. Vấn đề ở chỗ là chúng ta phải điều chỉnh khối (heap) khi một giá trị trong d_tree thay đổi. Quá trình điều chỉnh đó có độ phức tạp là $O(m \log n)$ trong trường hợp xấu nhất vì có khả năng mỗi cạnh sẽ có một lần cập nhật và mỗi lần cập nhật đòi hỏi một phép toán có độ phức tạp là $O(\log n)$. Do đó, độ phức tạp của toàn bộ thuật toán Prim là $O(m \log n)$. Qua thí nghiệm có thể thấy rằng hai thuật toán Prim và Kruskal có tốc độ như nhau, nhưng nói chung, thuật toán Prim thích hợp hơn với các mạng dày còn thuật toán Kruskal thích hợp hơn đối với các mạng mỏng. Tuy vậy, những thuật toán này chỉ là một phần của các thủ tục lớn và phức tạp hơn, đó là những thủ tục hoạt động hiệu quả với một trong những thuật toán này.



Hình 4.2. Graph có liên kết song song và self loop

Bảng 4.1

Nút	init.	A	C	E	B	D
A	0	0(-)	0(-)	0(-)	0(-)	0(-)
B	100	10(A)	10(A)	10(A)	10(A)	10(A)
C	100	2(A)	2(A)	2(A)	2(A)	2(A)
D	100	100(-)	11(A)	11(A)	5(B)	5(B)
E	100	7(A)	6(C)	6(C)	6(C)	6(C)

Ví dụ 4.4:

Trở lại hình 4.4, giả sử rằng các cạnh không được biểu diễn có độ dài bằng 100. Thuật toán Kruskal sẽ chọn (A, C) , (B, D) , (C, E) , và loại (A, E) bởi vì nó tạo ra một chu trình với các cạnh đã được chọn là (A, C) và (C, E) , chọn (A, B) sau đó dừng lại vì một cây bắc cầu hoàn toàn đã được tìm thấy.

Thuật toán Prim bắt đầu từ nút A, nút A sẽ được thêm vào cây. Tiếp theo là các nút C, E, B và D. Bảng 4.1 tổng kết các quá trình thực hiện của thuật toán Prim, biểu diễn d_tree và $pred$ khi thuật toán thực hiện. Cuối thuật toán, $pred[B]$ là A, tương ứng với (A, B) là một phần của cây. Tương tự, $pred$ chỉ ra (A, C) , (B, D) và (C, E) là các phần của cây. Vì vậy, thuật toán Prim sẽ lựa chọn được cây giống với cây mà thuật toán Kruskal nhưng thứ tự các liên kết được lựa chọn là khác nhau.

Một *đường đi* trong một mạng là một chuỗi liên tiếp các liên kết bắt đầu từ một nút s nào đó và kết thúc tại một nút t nào đó. Những đường đi như vậy được gọi là một *đường đi s, t*. Chú ý rằng thứ tự các liên kết trong đường đi là có ý nghĩa. Một đường đi có thể là hữu hướng hoặc vô hướng tùy thuộc vào việc các thành phần của nó là các liên kết hay là các cung. Người ta gọi một đường đi là đường đi đơn giản nếu không có nút nào xuất hiện quá hai lần trong đường đi đó. Chú ý rằng một đường đi đơn giản trong một graph đơn giản có thể được biểu

diễn bằng chuỗi liên tiếp các nút mà đường đi đó chứa vì chuỗi các nút đó biểu diễn duy nhất một chuỗi các liên kết.

Nếu s trùng với t thì đường đi đó gọi là một **chu trình**, và nếu một nút trung gian xuất hiện không quá một lần thì chu trình đó được gọi là **chu trình đơn giản**. Một chu trình đơn giản trong một graph đơn giản có thể được biểu diễn bởi một chuỗi các nút liên tiếp.

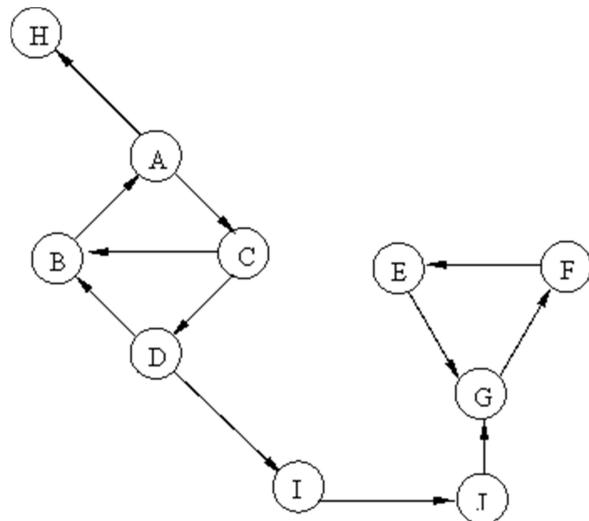
Ví dụ 4.5:

Xét graph hữu hướng trong hình 4.4. Các thành phần liên thông bền được xác định bởi

$$\{A B C D\} \quad \{E F G\} \quad \{H\} \quad \{\}\quad \{J\}$$

Các cung (A, H) , (D, I) , (I, J) và (J, G) không là một phần một thành phần liên thông bền nào cả. Xem graph trong hình 4.3 là một graph vô hướng (nghĩa là xem các cung là các liên kết vô hướng), thì graph này có một thành phần duy nhất, vì thế nó là một graph liên thông.

Cho một graph $G = (V, E)$, H là một **graph con** nếu $H = (V', E')$, trong đó V' là tập con của V and E' là tập hợp con của E . Các tập con này có thể có hoặc không tuân theo quy định đã nêu.



Hình 4.4. Graph hữu hướng

Một graph không hề chứa các chu trình gọi là **cây**. Một **cây bắc cầu** là một graph liên thông không có các chu trình. Những graph như vậy được gọi một cách đơn giản là **cây**. Khi graph không liên thông hoàn toàn được gọi là **rừng**. Chúng ta thường dễ cập các cây trong các graph vô hướng.

Trong các graph hữu hướng, có một cấu trúc tương tự với cây gọi là **cây phân nhánh**. Một cây phân nhánh là một graph hữu hướng có các đường đi từ một nút (gọi là **nút gốc của cây phân nhánh**) tới tất cả các nút khác hoặc nói một cách khác là graph hữu hướng có các

đường đi từ tất cả các nút khác đến nút gốc. Một cây phân nhánh sẽ trở thành một cây nếu nó là vô hướng.

Các cây bắc cầu có rất nhiều thuộc tính đáng quan tâm, những thuộc tính đó khiến cho các cây bắc cầu rất hữu ích trong quá trình thiết kế mạng truyền thông. Thứ nhất, các cây bắc cầu là liên thông tối thiểu có nghĩa là: chúng là các graph liên thông nhưng không tồn tại một tập con các cạnh nào trong cây tạo ra một graph liên thông. Chính vì vậy, nếu mục đích chỉ đơn giản là thiết kế một mạng liên thông có giá tối thiểu thì giải pháp tối ưu nhất là chọn một cây. Điều này có thể hiểu được vì trong một cây luôn có một và chỉ một đường đi giữa một cặp nút. Điều đó không gây khó khăn đáng kể trong việc định tuyến trong cây và làm đơn giản các thiết bị truyền thông liên quan đi rất nhiều.

Chú ý rằng một graph có N nút thì bất kỳ một cây nào bắc cầu tất cả các nút thì có đúng $(N-1)$ cạnh. Bất kỳ một rừng nào có k thành phần thì chứa đúng $(N-k)$ cạnh. Nhận xét này có thể suy ra từ lập luận sau: khi có một graph có N nút và không có cạnh nào thì có N thành phần, và cứ mỗi cạnh thêm vào nhằm kết nối hai thành phần thì số lượng thành phần giảm đi một.

Một tập hợp các cạnh mà sự biến mất của nó chia cắt một graph (hay nói một cách khác là làm tăng số lượng thành phần của graph) được gọi là một **tập chia cắt**. Một tập chia cắt nào đó chia cắt các nút thành hai tập X và Y được gọi là một **cutset** hoặc một **XY-cutset**. Hầu hết các vấn đề cần quan tâm đều liên quan đến các cutset tối thiểu (nghĩa là các cutset không phải là tập con của một cutset khác). Trong một cây, một cạnh bất kỳ là một cutset tối thiểu. Một tập tối thiểu các nút mà sự biến mất của nó phân chia các nút còn lại thành hai tập gọi là một **cut**. Các vấn đề cần quan tâm cũng thường liên quan đến các **cut** tối thiểu.

Ví dụ 4.6:

Hình 4.4 biểu diễn một graph vô hướng. Các tập các liên kết

$$\{(A, C), (B, D)\}$$

và

$$\{(C, E), (D, E), (E, F)\}$$

là các ví dụ của các cutset tối thiểu.

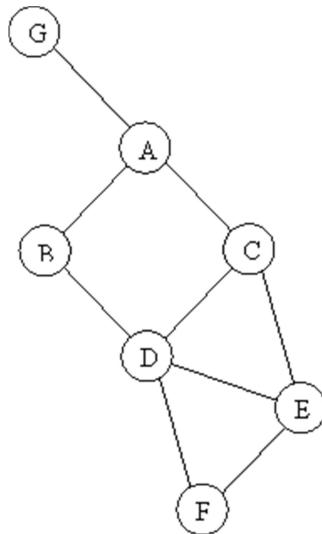
Tập cuối cùng là một ví dụ về một loại tập trong đó tập các liên kết đi tới một nút thành viên bất kỳ đều là một cutset và các cutset đó chia cắt nút đó ra khỏi các nút khác.

Tập (C, D) là một cut. Nút A cũng là một cut. Một nút duy nhất mà sự biến mất của nó chia cắt graph gọi là một điểm khớp nối.

Tập hợp các liên kết:

$$\{(A, B), (A, C), (A, G), (C, D), (C, E), (E, F)\}$$

là một cây. Bất kỳ tập con nào của tập này, kể cả tập đầy hay tập rỗng, đều là một rừng.



Hình 4.4. Các cutset, các cut, các cây

5.3. Các mô hình định tuyến thông dụng

5.3.1. Định tuyến ngắn nhất (Shortest path Routing)

Bài toán tìm các đường đi ngắn nhất là một bài toán khá quan trọng trong quá trình thiết kế và phân tích mạng. Hầu hết các bài toán định tuyến có thể giải quyết như giải quyết bài toán tìm đường đi ngắn nhất khi một "độ dài" thích hợp được gắn vào mỗi cạnh (hoặc cung) trong mạng. Trong khi các thuật toán thiết kế thì cố gắng tìm kiếm cách tạo ra các mạng thoả mãn tiêu chuẩn độ dài đường đi.

Bài toán đơn giản nhất của loại toán này là tìm đường đi ngắn nhất giữa hai nút cho trước. Loại bài toán này có thể là bài toán tìm đường đi ngắn nhất từ một nút tới tất cả các nút còn lại, tương đương bài toán tìm đường đi ngắn nhất từ tất cả các điểm đến một điểm. Đôi khi đòi hỏi phải tìm đường đi ngắn nhất giữa tất cả các cặp nút. Các đường đi đôi khi có những giới hạn nhất định (chẳng hạn như giới hạn số lượng các cạnh trong đường đi).

Tiếp theo, chúng ta xét các graph hữu hướng và giả sử rằng đã biết độ dài của một cung giữa mỗi cặp nút i và j là l_{ij} . Các độ dài này không cần phải đối xứng. Khi một cung không tồn tại thì độ dài l_{ij} được giả sử là rất lớn (chẳng hạn lớn gấp n lần độ dài cung lớn nhất trong mạng). Chú ý rằng có thể áp dụng quá trình này cho các mạng vô hướng bằng cách thay mỗi cạnh bằng hai cung có cùng độ dài. Ban đầu giả sử rằng l_{ij} là dương hoàn toàn; sau đó giả thiết này có thể được thay đổi.

Thuật toán Dijkstra

Tất cả các thuật toán tìm đường đi ngắn nhất đều dựa vào các nhận xét được minh họa trên hình 4.5, đó là việc lồng nhau giữa các đường đi ngắn nhất nghĩa là một nút k thuộc một đường đi ngắn

nhất từ i tới j thì đường đi ngắn nhất từ i tới j sẽ bằng đường đi ngắn nhất từ i tới k kết hợp với đường đi ngắn nhất từ j tới k. Vì thế, chúng ta có thể tìm đường đi ngắn nhất bằng công thức đệ quy sau:

$$d_{ij} = \min_k (d_{ik} + d_{kj})$$

d_{xy} là độ dài của đường đi ngắn nhất từ x tới y. Khó khăn của cách tiếp cận này là phải có một cách khởi động để quy nào đó vì chúng ta không thể khởi động với các giá trị bất kỳ ở về phải của phương trình 4.2. Có một số cách để thực hiện việc này, mỗi cách là cơ sở cho một thuật toán khác nhau.



Hình 4.5. Các đường ngắn nhất lồng nhau

Thuật toán Dijkstra phù hợp cho việc tìm đường đi ngắn nhất từ một nút i tới tất cả các nút khác. Bắt đầu bằng cách thiết lập

$$d_{ii} = 0$$

và

$$d_{ij} = \infty \quad \forall i \neq j$$

sau đó thiết lập

$$d_{ij} \leftarrow l_{ij} \quad \forall j \text{ là nút kề cận của } i$$

Sau đó tìm nút j có d_{ij} là bé nhất. Tiếp đó lấy chính nút j vừa chọn để khai triển các khoảng cách các nút khác, nghĩa là bằng cách thiết lập

$$d_{ik} \leftarrow \min (d_{ik}, d_{ij} + l_{jk})$$

Tại mỗi giai đoạn của quá trình, giá trị của d_{ik} là giá trị ước lượng hiện có của đường đi ngắn nhất từ i tới k; và thực ra là độ dài đường đi ngắn nhất đã được tìm cho tới thời điểm đó. Xem d_{ik} như là nhãn trên nút k. Quá trình sử dụng một nút để triển khai các nhãn cho các nút khác gọi là quá trình quét nút.

Thực hiện tương tự, tiếp tục tìm các nút chưa được quét có nhãn bé nhất và quét nó. Chú ý rằng, vì giả thiết rằng tất cả các l_{jk} đều dương do đó một nút không thể gán cho nút khác một nhãn bé hơn chính nhãn của nút đó. Vì vậy, khi một nút được quét thì việc quét lại nó nhất thiết không bao giờ xảy ra. Vì thế, mỗi nút chỉ cần được quét một lần. Nếu nhãn trên một nút thay đổi, nút đó phải được quét lại. Thuật toán Dijkstra có thể được viết như sau:

```

array[n] <- Dijkstra (n, root, dist)
    dcl dist[n,n], pred[n], sp_dist[n],
scanned[n]

    index <- FindMin( )
        d_min <- INFINITY
        for each (i , n )
            if (! (scanned[j]) && (sp_dist[i]<
d_min) )
                i_min <- i
                d_min <- sp_dist[i]
        return (i_min)

    void <- Scan( i )
        for each ( j , n)
            if((sp_dist[j] > sp_dist[i] +
dist[i,j]))
                sp_dist[j]<- sp_dist[i] +
dist[i,j]
                pred[j]<- i

    sp_dist<- INFINITY
    pred <- -1
    scanned <-FALSE
    sp_dist[root]<- 0
    #scanned <- 0

    while (#_scanned < n )
        i <- FindMin()
        Scan( i )
        #_scanned= #_scanned + 1
return ( pred )

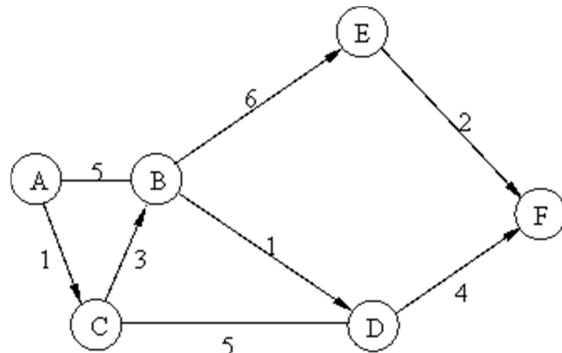
```

Trong thuật toán đã viết ở trên, hàm chỉ trả về dãy *pred* , dãy này chứa tất cả các đường đi. Hàm cũng có thể trả về dãy *sp_dist*, dãy này chứa độ dài của các đường đi, hoặc hàm trả về cả hai dãy nếu cần thiết.

Thuật toán trông rất quen thuộc. Nó gần giống với thuật toán tìm cây bắc cầu tối thiểu Prim. Chỉ khác nhau ở chỗ, các nút trong thuật toán này được gắn nhãn là độ dài của toàn bộ đường đi chứ không phải là độ dài của một cạnh. Chú ý rằng thuật toán này thực hiện với graph hữu hướng trong khi thuật toán Prim chỉ thực hiện với graph vô hướng. Tuy nhiên về mặt cấu trúc, các thuật toán là rất đơn giản. Độ phức tạp của thuật toán Dijkstra, cũng giống như độ phức tạp của thuật toán Prim , là $O(N^2)$.

Cũng giống như thuật toán Prim, thuật toán Dijkstra thích hợp với các mạng dày và đặc biệt thích hợp với các quá trình thực hiện song song (ở đây phép toán scan có thể được thực hiện song song, về bản chất độ phức tạp của quá trình đó là $O(1)$ chứ không phải là $O(N)$). Hạn chế

chủ yếu của thuật toán này là không có được nhiều ưu điểm khi mạng là mỏng và chỉ phù hợp với các mạng có độ dài các cạnh là dương.



Hình 4.6. Các đường đi ngắn nhất từ A

Ví dụ 4.7:

Xét một mạng trong hình 4.6. Mục tiêu ở đây là tìm các đường đi ngắn nhất từ nút A tới các nút khác. Khởi đầu, A được gán nhãn 0 và các nút khác được gán nhãn là vô cùng lớn. Quét nút A, B được gán bằng 5 và C được gán là 1. C là nút mang nhãn bé nhất nên sau đó C được quét và B được gán bằng 4 ($=1+3$), trong khi D được gán bằng 6. Tiếp theo B (có nhãn bằng 4) được quét; D và E được gán lần lượt là 5 và 10. Sau đó D (có nhãn bằng 5) được quét và F được gán bằng 9. E được quét và dẫn đến không có nhãn mới. F là nút có nhãn bé nhất nên không cần phải quét vì không có nút nào phải đánh nhãn lại. Mỗi nút chỉ được quét một lần. Chú ý rằng việc quét các nút có các nhãn theo thứ tự tăng dần là một điều cần lưu ý vì trong quá trình thực hiện thuật toán một số nút được đánh lại số. Các nút được quét ngay tức thì hoặc là phải được quét lại sau đó.

Chú ý rằng các đường đi từ A đến các nút khác (nghĩa là (A, C) , (C, B) , (B, D) , (B, E) và (D, F)) tạo ra một cây. Điều này không là một sự trùng hợp ngẫu nhiên. Nó là hệ quả trực tiếp từ việc lồng nhau của các đường đi ngắn nhất. Chẳng hạn, nếu k thuộc đường đi ngắn nhất từ i tới j thì đường đi ngắn nhất từ i tới j sẽ là tổng của đường đi ngắn nhất từ i tới k và đường đi ngắn nhất từ k tới j .

Tương tự như trong ví dụ minh họa cho thuật toán Prim, kết quả của ví dụ trên có thể được trình bày một cách ngắn gọn như bảng sau:

Bảng 4.2

Nút	init.	A(0)	C(1)	B(4)	D(5)	F(9)	E(10)
A	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)
B	∞ (-)	5(A)	4(C)	4(C)	4(C)	4(C)	4(C)
C	∞ (-)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)
D	∞ (-)	∞ (-)	6(C)	5(B)	5(B)	5(B)	5(B)
E	∞ (-)	∞ (-)	∞ (-)	10(B)	10(B)	10(B)	10(B)
F	∞ (-)	∞ (-)	∞ (-)	∞ (-)	9(D)	9(D)	9(D)

Thuật toán Bellman

Một thuật toán khác của dạng thuật toán Dijkstra do Bellman phát biểu và sau đó được Moore và Page phát triển, đó là việc quét các nút theo thứ tự mà chúng được đánh nhãn. Việc đó loại trừ việc phải tìm nhãn nhỏ nhất, nhưng tạo ra khả năng; một nút có thể cần quét nhiều hơn một lần.

Trong dạng đơn giản nhất, thuật toán Bellman duy trì một hàng đợi các nút để quét. Khi một nút được đánh nhãn nó được thêm vào hàng đợi trừ khi nó đã tồn tại trong hàng đợi. Hàng đợi được quản lý theo quy tắc vào trước, ra trước. Vì thế các nút được quét theo thứ tự mà chúng được đánh nhãn. Nếu một nút được đánh nhãn lại sau khi nút đó được quét thì nó được thêm vào sau hàng đợi và được quét lần nữa.

Ví dụ 4.8:

Trong ví dụ ở hình 4.6, chúng ta bắt đầu quá trình bằng các đặt nút A vào hàng đợi. Quét A các nhãn 5 và 1 lần lượt được gán cho nút B và C, đồng thời các nút B và C được đưa vào hàng đợi (vì các nút này nhận giá trị mới và chưa có mặt trong hàng đợi). Tiếp đó chúng ta quét nút B và các nút E và D được đánh nhãn lần lượt là 11 và 6. D và E cũng được đặt vào hàng đợi. Sau đó chúng ta quét C, khi đó B được gán nhãn là 4 và lại được đặt vào sau hàng đợi. E được quét và F được gán nhãn là 13 và đưa vào hàng đợi. D được quét và F được gán nhãn là 10; F vẫn còn ở trong hàng đợi nên F không được đưa vào hàng đợi. B được quét lần thứ hai. trong lần quét này E và D lần lượt được đánh nhãn là 10 và 5 đồng thời cả hai nút được đặt vào hàng đợi. F được quét và không đánh nhãn nút nào cả. E được quét không đánh nhãn nút nào cả. D được quét và F được đánh nhãn 9 và được đưa vào hàng đợi. F được quét và không đánh dấu nút nào cả.

Các nút B, C, D và F được quét hai lần. Đó là cái giá phải trả cho việc không quét các nút theo thứ tự. Một khác trong thuật toán này không cần thiết phải tìm kiếm các nút có nhãn nhỏ nhất.

Cũng như trong hai ví dụ 4.4 và 4.5 cho thuật toán Prim và thuật toán Dijkstra, chúng ta có thể trình bày kết quả của các quá trình trong ví dụ này như trong bảng sau

Bảng 4.3

Nút	init.	A(0)	B(5)	C(1)	E(11)	D(6)
A	0(-) A	0(-) B	0(-) C	0(-) E	0(-) D	0(-) B
B	∞ (-)	5(A)	C 5(A)	E 4(C)	D 4(C)	B 4(C)
C	∞ (-)	1(A)	1(A) D	1(A) B	1(A) F	1(A)
D	∞ (-)	∞ (-)	6(B)	6(B)	6(B)	6(B)
E	∞ (-)	∞ (-)	11(B)	11(B)	11(B)	11(B)
F	∞ (-)	∞ (-)	∞ (-)	∞ (-)	13(E)	10(D)
		B(4)	F(10)	E(10)	D(5)	F(9)
A		0(-) F	0(-) E	0(-) D	0(-) F	0(-)
B		4(C)	E 4(C)	D 4(C)	4(C)	4(C)
C		1(A) D	1(A)	1(A)	1(A)	1(A)
D		5(B)	5(B)	5(B)	5(B)	5(B)
E		10(B)	10(B)	10(B)	10(B)	10(B)
F		10(D)	10(D)	10(D)	9(D)	9(D)

Thuật toán có thể viết như sau:

```

array[n]<-Bellman (n, root, dist)
dcl dist[n][n], pred[n], sp_dist[n],
in_queue[n]
    scan_queue[queue]

    void <- Scan( i )
        in_queue[i]<- FALSE
for j=1 to n
    if((sp_dist[j] > sp_diat[i] +
dist[i,j]))
        sp_dist[j]<- sp_diat[i] +
dist[i,j]
        pred[j]<- i
        if ( not ( in_queue[j] ) )
            Push(scan_queue, j )
            in_queue[j]<- TRUE

sp_dist<- INFINITY
pred <- -1
in_queue <-FALSE
initialize_queue( scan_queue )
sp_dist[root]<- 0
Push(scan_queue , root )
in_queue <-TRUE

```

```

while (not (Empty( scan_queue ))
       i <- Pop(scan_queue)
       Scan( i )

return ( pred )

```

Một hàng đợi chuẩn được sử dụng quá trình trên. Có thể sử dụng dãy *in_queue* để theo dõi nút nào đang hiện có trong hàng đợi.

Theo quá trình được viết ở trên thì thuật toán Bellman là một quá trình tìm kiếm theo chiều rộng. Người ta đã chứng minh được rằng trong trường hợp xấu nhất, một nút được quét $n-1$ lần. Vì vậy quá trình quét trong trường hợp xấu nhất có độ phức tạp là $O(n)$ với n là số lượng các nút. Từ đó suy ra rằng độ phức tạp của toàn bộ thuật toán là $O(n^3)$. Tuy nhiên trong thực tế các nút không thường xuyên được quét lại nhiều lần.

Trong hầu hết các trường hợp thực tế, số lần quét trung bình trên một nút là rất nhỏ, tối đa là 3 hoặc 4, ngay cả khi mạng có hàng ngàn nút. Nếu bậc trung bình của nút nhỏ, điều này thường xảy ra trong các mạng thực tế, thì thời gian cho việc tìm kiếm nút chưa quét bé nhất là phần có ảnh hưởng nhất của thuật toán Dijkstra. Vì vậy trong thực tế thuật toán Bellman được xem là nhanh hơn so với thuật toán Dijkstra mặc dù độ phức tạp trong trường hợp xấu nhất của thuật toán Bellman lớn hơn.

Tương tự có thể cải tiến độ phức tạp của thủ tục *Scan* bằng cách duy trì một danh sách kè cận cho mỗi nút. Độ phức tạp của *Scan* trở thành $O(d)$ thay vì $O(n)$ với d là bậc của nút đang quét. Vì vậy, trên thực tế độ phức tạp của thuật toán Bellman thường bằng $O(E)$ với E là số cạnh của graph.

Ngoài việc có thể cải thiện chất lượng trung bình của thuật toán trong nhiều trường hợp, thuật toán Bellman còn có một ưu điểm nữa đó là thuật toán hoạt động ngay cả khi độ dài các cạnh là các giá trị âm. Thuật toán Dijkstra dựa vào quy tắc: một nút không thể gán cho nút khác một nhãn bé hơn nhãn của chính nút. Điều đó chỉ đúng khi không có các cung có độ dài là âm trong khi thuật toán Bellman không cần phải giả thiết như vậy và quét lại các nút mỗi khi nút đó được gán nhãn lại. Vì thế, thuật toán này rất phù hợp khi xuất hiện các cung có độ dài âm. Tuy nhiên cần chú ý rằng khi graph có một chu trình có tổng độ dài âm thì thậm chí thuật toán Bellman cũng không khả dụng. Trong trường hợp này, thuật toán không kết thúc và các nút tiếp tục đánh nhãn các nút khác một cách vô hạn. Có một số dạng khác nhau của thuật toán Bellman, ngoài thuật toán này ra còn có một số các thuật toán tìm đường đi ngắn nhất từ một điểm tới các điểm khác trong trường hợp khác nhau.

Thuật toán Floyd

Có thể thấy rằng bài toán tìm kiếm đường ngắn nhất giữa mọi cặp nút nặng nề gấp N lần bài toán tìm đường đi ngắn nhất từ một nút đến tất

cả các nút khác. Một khả năng có thể đó là sử dụng thuật toán Bellman hoặc thuật toán Dijkstra N lần, bắt đầu từ mỗi nút nguồn. Một khả năng khác, đặc biệt thích hợp với các mạng dày, là sử dụng thuật toán Floyd.

Thuật toán Floyd dựa vào quan hệ đệ quy đã được trình bày trong phần giới thiệu thuật toán Dijkstra, nhưng thuật toán này sử dụng quan hệ đệ quy đó theo một cách khác. Lúc này, $d_{ij}(k)$ được định nghĩa là đường đi ngắn nhất từ i tới j sử dụng các nút được đánh số là k hoặc thấp hơn như là các nút trung gian. Vì thế $d_{ij}(0)$ được định nghĩa như là l_{ij} , độ dài của liên kết từ nút i tới nút j , nếu liên kết đó tồn tại hoặc $d_{ij}(0)$ sẽ bằng vô cùng nếu liên kết đó không tồn tại. Vì vậy,

$$d_{ij}(k) = \min (d_{ij}(k-1), d_{ik}(k-1) + d_{kj}(k-1))$$

nghĩa là, chúng ta chỉ quan tâm đến việc sử dụng nút k như là một điểm quá giang cho mỗi đường đi từ i tới j . Thuật toán có thể được viết như sau:

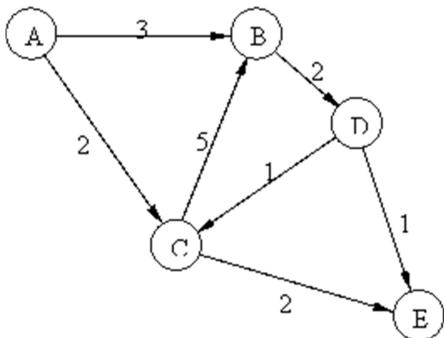
```
array[n] <-Floyd (n, dist)
dcl dist[n][n], pred[n][n], sp_dist[n,n]

for each (i , n )
    for each (i , n )
        sp_dist[i,j] <- dist[i, j]
        pred[i, j]<- i

    for each (k , n )
        for each (i , n )
            for each (j , n )
                if((sp_dist[i,j]> sp_dist[i,k] +
dist[k,j]))
                    sp_dist[i,j]<- sp_dist[i,k] +
dist[k,j]
                    pred[i, j]<- pred[k,j]

return ( pred )
```

$pred[i,j]$ chứa nút trung gian cuối cùng của đường đi từ i tới j và có thể được sử dụng để khôi phục đường đi từ i tới j . Giống như thuật toán Bellman, thuật toán Floyd hoạt động cả với các độ dài cung là âm. Nếu xuất hiện các chu trình có tổng độ dài âm thì thuật toán Floyd dừng lại nhưng không bảo đảm các đường đi là ngắn nhất. Các chu trình có tổng độ dài âm có thể được nhận biết nhờ sự xuất hiện của các con số âm trên đường chéo chính của dãy sp_dist .



Hình 4.7: Ví dụ graph

Ví dụ 4.9:

Xét graph trong hình 4.7. Mảng chứa khoảng cách ban đầu và mảng chứa nút trung gian cuối cùng của mỗi đường đi được cho trước như sau:

		Đến							Đến				
		A	B	C	D	E			A	B	C	D	E
T ù r r o w	A	0	3	2	-	-			A	A	A	A	A
	B	-	0	-	2	-			B	B	B	B	B
	C	-	5	0	-	2			C	C	C	C	C
	D	-	-	1	0	1			D	D	D	D	D
	E	-	-	-	-	0			E	E	E	E	E
	sp_dist							pred					

Chú ý rằng sp_dist có các giá trị 0 trên đường chéo chính và vô cùng lớn (được biểu diễn là dấu "-") nếu giữa hai nút không tồn tại một liên kết. Ngoài ra vì graph là graph hữu hướng và không đối xứng nên sp_dist cũng không đối xứng.

Xét A ta thấy A là một nút trung gian không ảnh hưởng đến các dãy này vì không có cung nào đi tới nó và vì thế không có đường đi nào đi qua A. Tuy nhiên, xét nút B ta thấy nút B gây nên sự thay đổi ở vị trí (A, D) và (C, D) trong các dãy trên, cụ thể như sau :

		Đến							Đến				
		A	B	C	D	E			A	B	C	D	E
T ù r r o w	A	0	3	2	5	-			A	A	A	A	B
	B	-	0	-	2	-			B	B	B	B	B
	C	-	5	0	7	2			C	C	C	C	B
	sp_dist							pred					

	D	-	-	1	0	1		D	D	D	D	D	D	
	E	-	-	-	-	0		E	D	D	D	D	D	
	<i>sp_dist</i>										<i>pred</i>			

Tiếp tục xét các nút C, D và E thì gây nên sự thay đổi cụ thể như sau:

		Đến					Đến				
		A	B	C	D	E	A	B	C	D	E
Tùy	A	0	3	2	5	4	A	A	A	B	C
	B	-	0	3	2	3	B	B	B	D	B
	C	-	5	0	7	2	C	C	C	B	C
	D	-	6	1	0	1	D	D	C	D	D
	E	-	-	-	-	0	E	E	E	E	E
	<i>sp_dist</i>						<i>pred</i>				

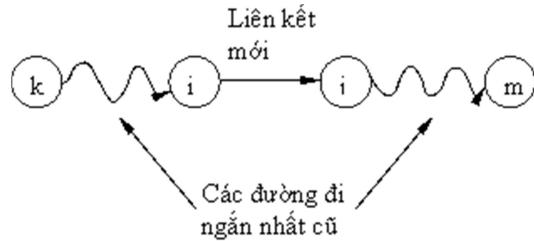
Các thuật toán tìm đường đi ngắn nhất mở rộng

Trong quá trình thiết kế và phân tích mạng đôi khi chúng ta phải tìm đường đi ngắn nhất giữa mọi cặp các nút (hoặc một số cặp) sau khi có sự thay đổi độ dài một cung. Việc thay đổi này bao gồm cả việc thêm hoặc loại bỏ một cung (trong trường hợp đó độ dài của cung có thể được xem như là chuyển từ không xác định thành xác định hoặc ngược lại). Vì thế ta giả thiết rằng đường đi ngắn nhất giữa tất cả các cặp nút là biết trước và bài toán đặt ra ở đây là xác định (nếu có) những sự thay đổi do việc thay đổi độ dài của một cung nào đó. Thuật toán sau đây được Murchland phát biểu, trong đó xem xét riêng rẽ cho từng trường hợp: tăng và giảm độ dài của các cung. Những thuật toán này hoạt động với các graph hữu hướng và có thể hoạt động với các độ dài cung là âm, tuy nhiên thuật toán này vẫn không giải quyết các chu trình có tổng độ dài là âm.

Độ dài cung giảm

Giả sử rằng độ dài cung (i, j) được giảm. Vì sự lồng nhau trong các đường đi ngắn nhất (nghĩa là một nút k thuộc một đường đi ngắn nhất từ i tới j thì đường đi ngắn nhất từ i tới j sẽ bằng đường đi ngắn nhất từ i tới k hợp với đường đi ngắn nhất từ j tới k) nên nếu cung (i, j) không phải là đường đi ngắn nhất sau khi cung này được làm ngắn (trong trường hợp này cung (i, j) có thể không phải là đường đi ngắn nhất trước khi độ dài của cung (i, j) bị thay đổi) thì nó không phải là một phần của đường đi ngắn nhất nào cả và sự thay đổi được bỏ qua. Tương tự, nếu (i, j) là một phần của đường đi ngắn nhất từ k tới m thì nó phải là một phần của đường đi ngắn nhất từ k tới j và đường đi ngắn nhất từ i tới m . Thực ra, đường đi ngắn nhất từ k tới m mới phải

chuỗi các đường đi từ k tới i cũ, liên kết (i, j) và đường đi từ j tới m . Điều này được biểu diễn trong hình 4.8.



Hình 4.8. Đường đi ngắn nhất mở rộng khi (i, j) được làm ngắn

Vì thế cần phải quét các nút i và j để tìm các tập K và M thích hợp chứa các nút k và m như trong hình 4.8 và thực hiện việc xét các cặp nút, mỗi nút từ một tập (K hoặc M đã nói ở trên). Với i thuộc K và j thuộc M thực hiện việc kiểm tra điều kiện sau

$$d_{km} > d_{ki} + l_{ij} + d_{jm}$$

nếu đúng, cập nhật d_{km} và nút trung gian cuối cùng của đường đi này. Thuật toán này có thể được viết như sau:

```

(array[n,n], array[n,n]) <-
sp_decrease(n,i,j,length,*dist,sp_dist,pred)

dcl dist[n,n], pred[n,n], sp_dist[n,n],
setk[set], setm[set]

dist[i, j]<- length
if(length >=sp_dist[i,j])
    return( sp_dist, pred )

setk <- Φ
setm <- Φ
for each (k, n)
    if(sp_dist[k,j]> sp_dist[k,i] + length)
        append(k, setk )

for each (m, n)
    if(sp_dist[i,m]> sp_dist[j,m] + length)
        append(m, setm )

for each (k , setk )
    for each (m , setm )
        if(sp_dist[k,m]>
sp_dist[k,i] + length + sp_dist[j,m])
            sp_dist[k,m]<-
sp_dist[k,i] + length + sp_dist[j,m]
            if ( j = m )
pred[k, m]<- i

```

```

        else
pred[k, m] <- pred[j, m]
return ( sp_dist , pred )

```

Thuật toán trả về sp_dist và $pred$, đường đi ngắn nhất đã được cập nhật và các dãy chứa nút trung gian cuối cùng của mỗi đường đi ngắn nhất. Hàm được xây dựng trong đoạn giả mã trên có đầu vào là dãy chứa các độ dài của các liên kết hiện có $dist$, điểm cuối (i và j) của liên kết mới được làm ngắn và độ dài mới của liên kết được làm ngắn $length$. Φ là danh sách rỗng.

Có thể thấy rằng, trong trường hợp xấu nhất độ phức tạp của thủ tục trên là $O(n^2)$ vì trong thủ tục trên có hai vòng lặp có độ phức tạp trong trường hợp xấu nhất là $O(n)$. Trong thực tế, trường hợp cả hai tập đều có độ phức tạp là $O(n)$ là ít khi gặp và vì thế độ phức tạp thực tế của thuật toán thường thấp hơn nhiều.

Độ dài cung tăng

Bây giờ xét trường hợp một liên kết (i,j) được kéo dài hoặc bị loại bỏ khỏi graph (trong trường hợp này độ dài của liên kết được xem là vô cùng lớn). Nếu (i, j) không phải là một phần của đường đi ngắn nhất từ k tới m trước khi độ dài liên kết (i,j) được tăng lên thì sau đó liên kết này chắc chắn cũng không thuộc đường đi ngắn nhất từ k tới m . Vì vậy cần kiểm tra cặp (k, m) có đường đi ngắn nhất thoả mãn điều kiện:

$$d_{km} = d_{ki} + l_{ij} + d_{jm}$$

Chú ý rằng, nếu l_{ij} không phải là một phần của đường đi ngắn nhất từ i tới j thì không có thay đổi nào xảy ra. Thuật toán này có thể được viết như sau:

```

(array[n,n], array[n,n]) <-
    sp_increase(n,i,j,*dist,length,
    sp_dist,pred )

dcl dist[n,n], pred[n,n], pairs[set]

if(length > sp_dist[i,j])
    dist[i,j] <- length
return( sp_dist, pred )

pairs <-  $\Phi$ 
for each (k, n)
    for each (m, n)
if(sp_dist[k,m]=
    sp_dist[k,i] + dist[i,j]+ sp_dist[i,m])
        append( (k,m), pairs )
        sp_dist[k,m] <- dist[k,m]

dist[i,j] <- length

```

```

for each (a , n )
    for each ((k,m) , pairs )
        if(sp_dist[k,m] > sp_dist[k,a]+
sp_dist[a,m])
            sp_dist[k,m]<- sp_dist[k,a]+
sp_dist[a,m]
            pred[k, m]<- pred[a, m]

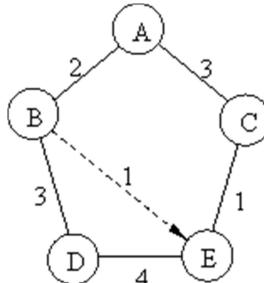
return ( sp_dist , pred )

```

Trong trường hợp này, *pairs* là một tập các cặp nút cần phải được kiểm tra. Vì vậy, các phần tử của *pairs* là các cặp nút. Thuật toán này có các tham số vào ra giống như thuật toán cập nhật các đường đi ngắn nhất khi giảm độ dài một cung.

Về bản chất thuật toán này giống như thuật toán Floyd, chỉ khác nhau ở chỗ thuật toán này chỉ hoạt động với các cặp được chọn chứa liên kết bị thay đổi trước khi liên kết này được kéo dài.

Độ phức tạp của thủ tục này là $O(np)$ với p là số cặp nút trong tập *pairs*. Trong trường hợp xấu nhất, tập *pairs* có thể chứa $n(n-1)$ cặp nút và vì thế độ phức tạp là $O(n^3)$, giống với độ phức tạp của thuật toán Floyd. Tuy nhiên trong thực tế p thường rất bé.



Hình 4.9

Ví dụ 4.10: (ví dụ cho trường hợp độ dài cung giảm)

Xét một mạng trong hình 4.9. Các cạnh trong mạng này là các liên kết hai hướng. Độ dài của các đường đi ngắn nhất giữa các cặp nút được cho trước trong bảng 4.4.

Bây giờ thêm một cung (B, E) có $l_{BE} = 1$. Vì

$$d_{BE} > l_{BE}$$

chúng ta thực hiện quá trình. Ngoài ra vì

$$d_{BC} > l_{BE} + d_{EC}$$

nhưng

$$d_{BX} \leq l_{BE} + d_{Ex}$$

đối với tất cả các nút khác. Vì vậy

$$set_m = C, E$$

Tương tự,

$$set_k = A, B$$

Bảng 4.4

	A	B	C	D	E
A	0	2	3	5	4
B	2	0	5	3	6
C	3	5	0	5	1
D	5	3	5	0	4
E	4	6	1	4	0

Bây giờ chúng ta xét các cặp nút

(k, m) với $k \in set_k$ và $m \in set_m$, (nghĩa là các cặp (A, C) , (A, E) , (B, C) và (B, E)). Chúng ta thấy rằng tất cả các cặp trừ cặp (A, C) đều bị thay đổi nên chúng ta cập nhật các đường đi ngắn nhất và nút trung gian cuối cùng của mỗi đường đi ngắn nhất giữa các cặp nút này. Ma trận đường đi ngắn nhất bây giờ được biểu diễn trong bảng 4.3

Bảng 4.5

	A	B	C	D	E
A	0	2	3	5	3
B	2	0	2	3	1
C	3	5	0	5	1
D	5	3	5	0	4
E	4	6	1	4	0

Chú ý rằng, ma trận này không còn đối xứng nữa vì một cung (B, E) vừa mới được thêm vào mạng.

Bây giờ giả sử rằng $I_{BE} = 5$ (ví dụ cho bài toán có sự tăng độ dài một cung). Kiểm tra ma trận đường đi ngắn nhất, ta thấy rằng trước khi thay đổi I_{BE} thì

$$d_{BE} = I_{BE}$$

Chúng ta kiểm tra tất cả các cặp nút (k, m) và thấy rằng điều kiện

$$d_{km} = d_{ki} + l_{ij} + d_{jm}$$

chỉ có các cặp (A, E) , (B, C) và (B, E) thoả mãn. Vì thế chúng ta thực hiện phép gán sau

```
pairs <- { (A, E), (B, C), (B, E) }
```

và sau đó thực hiện lặp trên tất cả các nút trung gian, kiểm tra các đường đi ngắn nhất đối với các cặp này. Khi thực hiện quá trình này, chú ý rằng đường đi ngắn nhất từ A tới E trở thành 4 (qua nút C) và đường đi ngắn nhất từ B tới C trở thành 5 (qua A). Tuy nhiên, đối với đường đi ngắn nhất từ B tới E thì cung (B, E) được giữ nguyên. Độ dài các đường đi ngắn nhất giữa các cặp nút được chỉ ra trong bảng 4.6.

Bảng 4.6

	A	B	C	D	E
A	0	2	3	5	4
B	2	0	5	3	5
C	3	5	0	5	1
D	5	3	5	0	4
E	4	6	1	4	0

Flow Network

Cho một tô-pô mạng và một yêu cầu duy nhất từ một nút nguồn s tới một nút đích d , yêu cầu đặt ra ở đây là tìm một dạng luồng khả thi, nghĩa là tìm một tập các luồng liên kết thoả mãn yêu cầu duy nhất nói trên mà không có bất kỳ luồng của liên kết nào có giá trị vượt quá dung lượng của chính liên kết đó. Tô-pô mạng được biểu diễn dưới dạng tập các liên kết l_{ij} , đi cùng với các dung lượng c_{ij} . Vì trong thực tế các mạng là các mạng thưa nên có thể lưu trữ topo mạng dưới dạng các danh sách hiện và khai thác các tính chất của mạng thưa. Ngoài ra có thể lưu trữ các dung lượng dưới dạng một ma trận, trong đó c_{ij} được gán bằng 0 khi l_{ij} không tồn tại.

Bài toán vì thế trở thành bài toán tìm một hoặc nhiều đường đi từ s tới d rồi gửi luồng đi qua các đường này đồng thời đảm bảo yêu cầu đã cho. Tổng các luồng bằng với giá trị yêu cầu và tổng luồng trên mỗi liên kết không vượt quá dung lượng của liên kết.

Có một số dạng của bài toán này. Dạng đầu tiên, như đã nói ở trên, là bài toán tìm các luồng thoả mãn một yêu cầu nào đó. Một dạng khác đó là bài toán tối đa hoá giá trị luồng từ s tới d đồng thời thoả mãn điều kiện dung lượng. Dạng cuối cùng là khi chúng ta biết được giá trên một đơn vị luồng dành cho mỗi liên kết, bài toán đặt ra là tìm một luồng thoả mãn yêu cầu cho trước có giá tối thiểu. Các lời giải cho các bài toán này liên hệ chặt chẽ với nhau và sẽ được xem xét sâu hơn. Hơn nữa, lời giải cho bài toán này là cơ sở cho việc giải quyết các bài toán phức tạp hơn gọi là bài toán luồng đa hạng, bài toán có rất nhiều yêu

cầu giữa các nguồn và các đích. Đây là bài toán hết sức quan trọng trong việc thiết kế mạng và sẽ được nói kỹ ở chương sau.

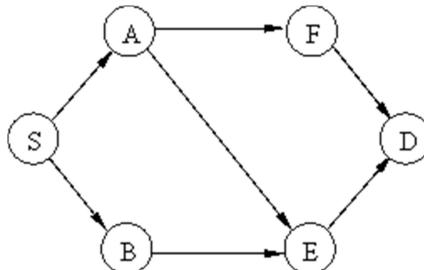
Chú ý rằng trong trường hợp này ta đang xét các liên kết hữu hướng (nghĩa là có sự khác nhau giữa c_{ij} và c_{ji}). Tuy nhiên có thể giải quyết các mạng vô hướng bằng cách thay thế mỗi liên kết vô hướng I_{ij} bằng hai liên kết hữu hướng có các dung lượng riêng rẽ. Như chúng ta sẽ thấy, trong bất kỳ liên kết nào và ở đâu trong quá trình tìm lời giải cho bài toán này, chỉ có luồng theo một hướng.

Có thể biểu diễn bài toán này dưới dạng bài toán tìm các luồng f_{ij} thoả mãn các điều kiện sau:

$$\begin{aligned} \sum_j f_{ij} - \sum_j f_{ji} &= r_{ij}; \forall i = s \\ \sum_j f_{ij} - \sum_j f_{ji} &= -r_{ij}; \forall i = d \\ \sum_j f_{ij} - \sum_j f_{ji} &= 0; \forall i \neq s \\ f_{ij} &\leq c_{ij} \\ f_{ij} &\geq 0; \forall i, j \end{aligned}$$

Thuật toán Ford-Fulkerson

Thuật toán tốt nhất cho việc giải bài toán luồng đơn hướng là thuật toán Ford-Fulkerson. Thuật toán này chỉ ra các đường đi từ nguồn s tới đích d và gửi các luồng lớn nhất có thể qua mỗi đường mà không vi phạm giới hạn dung lượng. Thực ra thuật toán được điều khiển nhằm chỉ ra các đường đi và điền đầy chúng bằng các luồng.



Hình 4.10. Mạng đơn giản

Chẳng hạn xét một mạng trong hình 4.10. Giả sử tất cả các liên kết có dung lượng là 1. Chúng ta có thể gửi một đơn vị luồng trên đường đi $SABD$ và một trên đường đi $SEFD$. Vì tổng dung lượng của các liên kết rời S là 2 và mỗi đơn vị luồng từ S tới D phải sử dụng một đơn vị dung lượng rời khỏi S này do đó không có luồng nào khác nữa thỏa mãn yêu cầu. Ngoài ra, vì mỗi đơn vị luồng phải sử dụng ít nhất một đơn vị dung lượng của một SD -cut bất kỳ (với SD -cut là một tập các liên kết mà sự biến mất của nó phân tách S khỏi D) nên luồng từ S tới D lớn nhất không thể lớn hơn dung lượng của bất kỳ cut nào (dung

lượng của cut là tổng dung lượng của tất cả các liên kết thuộc cut). Do đó ta có bồ đề sau:

Bồ đề 4.1 (Ford-Fulkerson)

Luồng từ S tới D lớn nhất không thể lớn hơn dung lượng của cut có dung lượng nhỏ nhất

Thực ra, luồng từ S tới D lớn nhất chính bằng dung lượng của $SD\text{-cut}$ có dung lượng bé nhất. Đó chính là định lý **Luồng Lớn nhất- Cutset Bé nhất** nổi tiếng của Ford-Fulkerson.

Giới hạn (1) đã nêu trên gọi là điều kiện giới hạn bảo toàn luồng. Điều kiện này đảm bảo rằng với các nút khác với nút nguồn và nút đích thì luồng vào bằng với luồng ra. Trong trường hợp này, các nút nguồn (đích) có luồng ra (vào) phải bằng luồng từ nguồn tới đích. Bất kỳ $SD\text{-cut}$ nào cũng phân chia các nút thành hai tập X và Y với S thuộc X và D thuộc Y . Nếu điều kiện (1) đổi với tất cả các nút thuộc tập X được cộng lại thì ta thấy rằng luồng tổng từ X tới Y trừ đi luồng tổng từ Y tới X có kết quả bằng luồng từ S tới D . Chú ý rằng tổng các phần ở về trái chính bằng tổng các luồng trong các liên kết có một đầu thuộc X còn đầu kia thuộc Y , trừ đi tổng các luồng trong các liên kết có một đầu thuộc Y còn đầu kia thuộc X . Các liên kết có hai đầu cùng thuộc X không tham gia vào tổng này vì chúng xuất hiện trong tổng nhưng có dấu ngược nhau. Các liên kết không có đầu nào thuộc X cũng không xuất hiện ở trong tổng. S tham gia vào về phải của điều kiện; tất cả các nút khác không tham gia.

Vì thế, để thoả mãn định lý trên cần phải:

Luồng tổng đi qua cut có dung lượng bé nhất phải bằng dung lượng của cut đó nghĩa là tất cả các liên kết thuộc cắt phải ở trạng thái bão hoà (luồng bằng dung lượng).

Luồng đi ngược cut này phải bằng 0.

Thực ra, tất các cut có dung lượng bé nhất phải là bão hoà và điều đó xảy ra vào cuối thuật toán. Thuật toán thực hiện bằng cách chỉ ra các đường đi có dung lượng bé và gửi luồng đi qua toàn bộ các đường đi đó. Khi không tìm ra một đường đi nào cả có dung lượng bé, một cut bão hoà được chỉ ra và thuật toán kết thúc. Các cut có dung lượng bé khác cũng bão hoà nhưng chúng không được thuật toán chỉ ra.

```
number <- FFflow( n , s , d , cap , *flow )
dcl cap[n][n] , flow[n][n], pred[n],
sign[n] , mxf[n] , scan_queue[queue]

void <-Scan( i )
    for each( j , n )
        if( predd[j] = U )
            if(flow[i,j] < cap[i,j])
```

```

         $mxf[i] \leftarrow \min(mxf[i], cap[i,j] -$ 
 $flow[i,j])$ 
 $mxf[j], pred[j], sign[j] \leftarrow$ 
 $mxf[i], +$ 
 $\text{else if } ($ 
 $flow[j,i] > 0)$ 
 $mxf[i] \leftarrow \min(mxf[i], flow[j,i])$ 
 $mxf[j], pred[j], sign[j] \leftarrow$ 
 $mxf[i], -$ 
 $\text{Push}(scan\_queue, j)$ 

void <-Backtrack( )
n <- d
tot_flow <- tot_flow + mxf[d]
while ( n != s )
    p <- pred[n]
    if (sign[n] = + )
        flow[p,n] <- flow[p,n] +
mxf[d]
    else
        flow[n,p] <- flow[n,p] +
mxf[d]

tot_flow <- 0
flow <- 0
flag <- TRUE

while ( flag )
    pred <- U
Initialize_queue ( scan_queue )
Push( scan_queue , s )
mxf[s] <-INFINITY
while( ! (Empty(scan_queue) && (pred[d] = U) ) )
    i<- Pop(scan_queue)
    Scan( i )
if( pred[d] != U )
    Backtrack( )
flag <- (pred[d] !=U)
return( tot_flow )

```

Trong trường hợp đơn giản nhất, thuật toán Ford-Fulkerson được viết như trong đoạn giả mã trên với n là số nút, m là số liên kết. Mỗi nút có một nhãn:

$(maxflow, pred, sign)$

Nhãn này biểu diễn giá trị luồng lớn nhất có thể trên đường đi hiện hành tính cho tới thời điểm đang xét, nút liền trước của nút đang xét trong đường đi hiện hành và chiều của luồng đi qua liên kết. Giá trị tương trưng U là không xác định; giá trị thực sự của U nên được phân biệt với bất kỳ giá trị hợp lệ nào khác.

Thuật toán trả về luồng trong mỗi liên kết. Tổng luồng đi từ nguồn tới đích có thể được tính bằng tổng các luồng đi ra khỏi nguồn (hoặc đi tới đích). Thuật toán chỉ ra đường đi từ nguồn tới đích bằng cách sử dụng một thuật toán được cải biến từ thuật toán Bellman. Thuật toán này cho phép sử dụng một liên kết (i,j) theo hướng tới (nghĩa là từ i tới j) nếu luồng từ i tới j là f_{ij} bé hơn dung lượng của liên kết đó c_{ij} . Nó cũng cho phép sử dụng liên kết theo chiều ngược lại (nghĩa là liên kết (i,j) được sử dụng để đưa luồng từ j tới i), nhưng điều này chỉ xảy ra nếu trước đó có một luồng từ i tới j là dương. Trong trường hợp này, luồng được loại ra khỏi liên kết (i,j) .

Luồng lớn nhất theo chiều từ i tới j là $c_{ij} - f_{ij}$. Luồng lớn nhất theo chiều từ j tới i là f_{ji} . Đại lượng mxf , trong các nhãn của mỗi nút, chỉ ra luồng lớn nhất có thể được gửi đi trên một đường đi.

Bên trong vòng `while` ở trên, chúng ta bắt đầu từ nút nguồn s và thực hiện việc tìm kiếm nhãn d . Nếu thành công, chúng ta có thể quan sát ngược từ d về s theo `pred` từ d . Thực ra quá trình này bao gồm việc tăng luồng trong mỗi liên kết theo hướng thuận và giảm luồng trong mỗi liên kết theo hướng ngược lại. Nếu không có nhãn cho d , thuật toán kết thúc. Khi đó thuật toán chỉ ra luồng lớn nhất; các liên kết (i, j) có i được gán nhãn và j không được gán nhãn tạo ra các cut bão hoà.

Hàm `Scan` có độ phức tạp là $O(n)$. Một dạng khác của thuật toán này hoạt động có hiệu quả hơn, đó là dạng có hàm `Scan` có độ phức tạp là $O(d)$ với d là bậc của nút, hàm này tạo ra một danh sách chứa các nút kề cận cho mỗi nút. Trong `Scan(i)` thay thế

`for j=1 to n`

bằng

`for each (j , adj_set[i])`

Khi thuật toán dừng lại, một cut hoàn toàn được định nghĩa. Các nút có nhãn khác U thì thuộc tập X và các nút còn lại thì thuộc Y , với X và Y được định nghĩa như trước đây. Việc đánh nhãn bảo đảm rằng tất cả các cung trong X , Y -cut là bão hoà, và tất cả các cung trong Y , X -cut có luồng bằng 0. Điều này có thể thấy rõ khi chú ý rằng thuật toán dừng lại khi việc đánh nhãn không được tiếp tục nữa. Bất kỳ cung chưa bão hoà nào thuộc S, D cut hoặc bất kỳ cung nào thuộc D, S cut có luồng khác không thì có thể được sử dụng để tiếp tục việc đánh nhãn. Khi chúng ta không tiếp tục đánh nhãn nghĩa là khi đó không có những cung như vậy. Vì vậy, luồng từ S tới D bằng với dung lượng của X, Y -cut và định lý **Luồng lớn nhất - Cut bé nhất** đã ngầm được chứng minh.

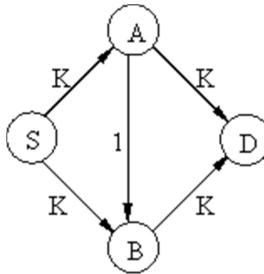
Ví dụ 4.11:

Xem xét việc sử dụng các cung theo chiều ngược cũng là một việc quan trọng. Nếu việc này không được thực hiện thì sẽ không đảm bảo rằng luồng là lớn nhất. Xét một mạng trong hình 4.10. Giả sử rằng, đường đi đầu tiên là $SAFD$, một đơn vị luồng được gửi đi trên toàn bộ đường đi. Tiếp đó một đường đi khác được tìm kiếm. S không thể đánh nhãn A bởi vì SA là một cung bão hoà. S đánh nhãn E và E đánh

nhân F , F không thể đánh nhân D vì FD là một cung bão hoà. Chú ý rằng, không tồn tại một cung từ F tới A ; cung FA chỉ có hướng từ A tới F . Điều cần chú ý ở đây là thuật toán phải sử dụng cung FA theo chiều ngược, do đó loại bỏ một đơn vị luồng khỏi cung đó. Điều đó cho phép F đánh nhân A , A đánh nhân B và B đánh nhân D . Vì thế một đường đi thứ hai được tìm thấy, đó là đường đi $SEFABD$ có cung FA được sử dụng theo chiều ngược. Kết quả của việc gửi luồng trên hai đường đi là gửi một đơn vị luồng từ S tới E , tới F , tới D và một đơn vị luồng như vậy từ S tới A , tới B và tới D . Đơn vị luồng ban đầu trên liên kết AF được loại trừ trong đường đi thứ hai và luồng mạng trên cung này bằng 0. Hai đường đi được tìm thấy bằng thuật toán có thể kết hợp tạo thành hai đường đi mới.

Như đã trình bày ở trên, đối với một mạng có N nút và E cạnh, một lần sử dụng thuật toán này để tìm một đường đi đơn thì có độ phức tạp bằng $O(N^2)$ vì mỗi nút được quét tối đa một lần (các nút không được đánh lại nhân), và độ phức tạp của phép quét là $O(N)$. Với thuật toán đã được sửa đổi từ thuật toán Bellman có sử dụng danh sách kề cận, mỗi nút được kiểm tra tối đa một lần từ mỗi đầu và một lần thực hiện việc đó có độ phức tạp bằng $O(E)$. Độ phức tạp trong việc thiết lập danh sách kề cận là $O(E)$ vì chỉ cần đi qua các cung một lần duy nhất cùng với việc chèn các nút vào danh sách kề cận. Vì vậy, đối với các mạng thừa, độ phức tạp không quá lớn.

Có thể thấy rằng độ phức tạp của toàn bộ thuật toán bằng tích của độ phức tạp khi tìm một đường đi đơn và số đường đi tìm được. Nếu dung lượng của các cung là các số nguyên thì mỗi đường đi cộng thêm ít nhất một đơn vị luồng vào mạng. Vì thế số lượng đường đi được giới hạn bởi luồng cuối cùng F . Do đó độ phức tạp toàn bộ của thuật toán là $O(EF)$.



Hình 4.11. Mạng đơn giản

Nói chung, F có thể rất lớn. Xét một mạng trong hình 4.11. Tất cả các cung ngoại trừ cung (A, B) đều có dung lượng bằng K , một số rất lớn. (A, B) có dung lượng bằng 1. Giả sử đường đi đầu tiên là $SABD$. Vì cung (A, B) có dung lượng bằng 1, nên chỉ có một đơn vị luồng có thể chuyển qua đường đi này. Tiếp đó, giả sử rằng $SBAD$ là đường đi được tìm thấy. Vì chỉ có một đơn vị luồng được loại khỏi (A, B) nên cũng chỉ có duy nhất một đơn vị luồng được gửi trên đường đi này. Thuật toán thực hiện tìm kiếm được $2K$ đường đi, các đường đi $SABD$ và $SBAD$ được lặp đi lặp lại, trong đó mỗi đường đi có một đơn vị

luồng được gửi đi, vì thế độ phức tạp đạt tới độ phức tạp trong trường hợp xấu nhất.

Các bài toán tương tự như bài toán nêu trên sẽ không thể xảy ra nếu thuật toán tìm các đường đi tìm được các đường đi có số bước tối thiểu. Thuật toán tìm kiếm theo chiều rộng sẽ thực hiện việc này. Từ trước tới nay, bài toán luồng lớn nhất đã được tìm hiểu khá kỹ và có rất nhiều thuật toán cũng như các thuật toán cải tiến từ các thuật toán đó dùng để giải quyết bài toán này.

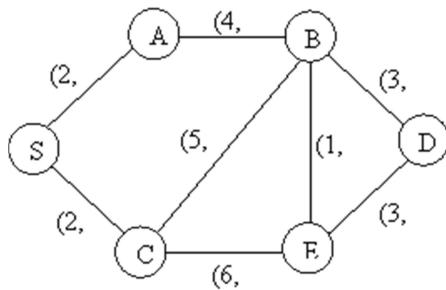
Trong thực tế, quá trình thực hiện thuật toán đã nêu trên có thể hoạt động để giải quyết hoặc là bài toán luồng lớn nhất hoặc là bài toán tìm một luồng có một giá trị cụ thể nào đó. Bây giờ chúng ta sẽ xem xét bài toán tìm các luồng có giá nhỏ nhất.

Các luồng có giá nhỏ nhất

Giả thiết rằng chúng ta đã biết giá của một đơn vị luồng c_{ij} trên mỗi liên kết. Yêu cầu đặt ra là tìm một luồng từ nguồn tới đích với giá trị cho trước có giá bé nhất, trong đó giá của một luồng được định nghĩa bằng tổng tất cả các tích của luồng trên mỗi liên kết nhân với giá của một đơn vị luồng trên liên kết đó. Tương tự, có thể chúng ta cần tìm một luồng với trị số lớn nhất có giá bé nhất. Chẳng hạn, chúng ta cần tìm một giá tối thiểu, nhưng vẫn đảm bảo là có thể tạo ra một luồng có trị số lớn nhất.

Cách đơn giản nhất để tìm một luồng có giá tối thiểu đó là sửa đổi thuật toán Ford-Fulkerson để tìm các đường đi ngắn nhất thay vì tìm các đường đi có bước nhỏ nhất với giá của một đơn vị luồng được sử dụng như các độ dài. Thuật toán Bellman hoặc bất kỳ thuật toán tìm đường ngắn nhất nào cũng có thể được làm cho tương thích với mục đích này. Yêu cầu đặt ra là phải theo dõi được luồng trên mỗi liên kết và giống như trong thuật toán Ford-Fulkerson, ở đây chỉ sử dụng các liên kết chưa bão hòa theo chiều thuận, và chỉ sử dụng các liên kết theo chiều ngược nếu các liên kết đó đang có luồng theo chiều thuận dương.

Cách thực hiện trên có thể được xem như là việc thực hiện thuật toán Ford-Fulkerson với một vài sửa đổi. Lúc này, mỗi nhãn có thêm một đại lượng thứ tư p , đó là độ dài của đường đi. Giá trị đó được cập nhật giống như cách đã làm trong thuật toán Bellman. Chẳng hạn, một nút có độ dài là p sẽ gán cho nút láng giềng của nó một độ dài đường đi là q với q bằng tổng của p và độ dài của liên kết nối hai nút.



Hình 4.12. Luồng có giá thấp nhất

Ví dụ 4.12:

Trong hình 4.12 mỗi liên kết được gán một nhãn (giá của một đơn vị luồng, dung lượng). Các liên kết là các liên kết hai hướng. Chẳng hạn, giá của việc chuyển một đơn vị luồng giữa A và B theo một trong hai hướng là 4. Sử dụng thuật toán Ford-Fulkerson, sửa đổi cách theo dõi độ dài các đường đi và cho phép một nút được đánh nhãn lại nếu độ dài đường đi trong nhãn của nút này được cải tiến (tích cực hơn) để giải quyết bài toán. Vì thế, mỗi nút có một nhãn

(pathlength, maxflow, pred, sign)

S có nhãn *(0, INFINITY, PHI, PHI)*, nhãn này chỉ ra rằng có một giá (độ dài đường đi) bằng 0 tính từ nguồn, không có giới hạn về luồng, và không có nút liền trước. Tất cả các nút khác ban đầu không có nhãn hoặc có nhãn gần giống với nhãn sau

(INFINITY, INFINITY, PHI, PHI)

Một nhãn có độ dài đường đi không xác định tương đương với việc không có nhãn nào vì bất kỳ khi nào đánh nhãn, cũng có một nhãn có độ dài đường đi xác định thay thế một nhãn như vậy.

S được đặt vào danh sách quét và nó là nút đầu tiên được quét, S đánh nhãn C bằng *(2, 4, S, +)* và C được đặt vào danh sách quét. Vì độ dài giữa S và chính nó bằng 0 và không có giới hạn về luồng mà nó có thể chuyển qua, nên độ dài đường đi chỉ đơn giản là độ dài của liên kết từ S tới C và luồng lớn nhất chính là dung lượng của liên kết (S, C). S gán nhãn A bằng *(2, 3, S, +)* và A được đặt vào danh sách quét. Việc chọn nút nào được đánh nhãn trước mang tính ngẫu nhiên. Điều này tuỳ thuộc vào thứ tự được thiết lập trong danh sách kè cận.

Sau đó C được quét, C thử đánh nhãn S nhưng điều đó là không thể vì S đã có một nhãn có độ dài đường đi bằng 0, trong khi C được gán một nhãn có độ dài đường đi bằng 4. Tuy nhiên C có thể đánh nhãn E bằng *(8, 3, C, +)*. Độ dài đường đi bằng 8 chính là tổng của 2 (độ dài đường đi trong nhãn hiện có của C) và 6 (độ dài của liên kết từ C tới E). Luồng lớn nhất chính là giá trị bé nhất của 4 (luồng lớn nhất trong nhãn của C) và 3 (dung lượng của liên kết từ C tới E trừ đi luồng

hiện tại là 0). E được đưa vào danh sách quét. Tương tự C gán nhãn B bằng $(11, 4, C, +)$ và B được đưa vào danh sách quét.

Sau đó A được quét. A có thể gán lại nhãn cho B bằng nhãn có độ dài đường đi bé hơn và B có nhãn bằng $(6, 2, A, +)$. Chú ý rằng B được gán lại nhãn có độ dài đường đi bé hơn, mặc dù điều đó dẫn đến luồng lớn nhất trong nhãn bé hơn. Điều này có thể giảm luồng trên đường đi đó nhưng không làm giảm tổng luồng được gửi tới D ; sự đánh nhãn kiểu này chỉ đơn giản là yêu cầu cần thêm đường đi để chuyển luồng đó. Mặc dù B được gán lại nhãn nhưng không được đưa vào danh sách quét vì B đã tồn tại trong danh sách quét.

E sau đó được quét, nút này gán nhãn D bằng $(11, 3, C, +)$. D là nút đích nên không cần phải đưa vào danh sách quét. Mặc dù D được gán nhãn nhưng vẫn phải tiếp tục đánh nhãn cho đến khi danh sách thành rỗng bởi vì vẫn có thể có một đường đi tốt hơn. E không thể gán nhãn B lần nữa vì nhãn của B có độ dài đường đi bằng 6 trong khi E chỉ có thể gán 9 cho B . Tiếp đó B được quét và D được đánh nhãn bằng $(9, 2, B, +)$.

Lúc này, danh sách quét đã rỗng. Đi ngược đường đi từ D , đường đi này có các nút sau: B (nút trước của D), A (nút trước của B) và S . Thêm 2 đơn vị luồng (luồng lớn nhất trong nhãn của D) vào các liên kết (S, A) , (A, B) và (B, D) . Lúc này cả ba liên kết đó có các luồng có luồng dương, vì thế chúng đủ điều kiện để sử dụng theo chiều ngược lại. Liên kết (A, B) bão hòa theo chiều thuận và chỉ đủ điều kiện để sử dụng theo chiều ngược.

Lần tìm thứ hai có kết quả là đường đi $SCED$ có độ dài là 11 và luồng bằng 3. Lần tìm thứ ba có kết quả là đường đi $SCEBD$ có độ dài là 12 và luồng bằng 1.

Trong lần tìm thứ tư, tất cả mọi nút đều được gán nhãn trừ nút D , nhưng D không thể được gán nhãn nên thuật toán kết thúc. Điều này tương ứng với các cut có dung lượng bằng 6 giữa các nút còn lại và nút D . Vì thế có một luồng lớn nhất bằng với dung lượng của một cut tối thiểu. Điều này tạo ra tổng giá trị bằng

$$(9 \times 2 + 11 \times 3 + 12 \times 1) = 63$$

Nếu chỉ muốn gửi ba đơn vị luồng thì điều đó có thể thực hiện với giá bằng

$$(9 \times 2 + 11 \times 1) = 29$$

với đường đi đầu tiên và đường đi thứ hai. Chính vì vậy, thuật toán này có thể được sử dụng để giải quyết bài toán luồng lớn nhất, giá bé nhất lần bài toán tìm luồng với giá trị cho trước có giá bé nhất. Trong bài toán tìm luồng với giá trị cho trước có giá bé nhất, thuật toán có thể dừng lại khi luồng đạt tới giá trị mong muốn. Trong bài toán luồng lớn nhất, giá bé nhất, như đã nói ở trên, thuật toán được thực hiện cho đến khi không có đường đi nào nữa được tìm thấy.

Sự mở rộng thuật toán Ford-Fulkerson là đúng đắn. Điều bất lợi duy nhất đó là việc phải mất sự đảm bảo về độ phức tạp tính toán. Không

còn có việc tìm kiếm theo chiều sâu nữa, và có thể phải tìm một đường đi mà phép tìm kiếm có độ phức tạp bằng $O(L)$ với luồng có độ lớn là L . Trong thực tế, các đường đi có độ dài bé nhất có xu hướng có bước nhỏ nhất và ít khi có sự thay đổi đáng kể về thời gian hoạt động. Thế nhưng, theo định lý điều đó có thể xảy ra. Điều này đặt ra yêu cầu về sự phát triển các thuật toán phức tạp hơn có độ phức tạp trong trường hợp xấu nhất bé hơn. Những thuật toán như thế gọi là thuật toán kép, rất nhiều trong số chúng bắt đầu bằng việc sử dụng thuật toán Ford-Fulkerson để tìm một luồng tối đa (hoặc một luồng có giá trị cho trước) và sau đó tìm kiếm đường chuyển luồng khác theo một chương trình có độ dài âm, chuyển luồng khỏi đường đi có giá cao hơn tới đường đi có giá thấp hơn.

5.4. Bài tập (Pending)

Chương 6 Điều khiển luồng và chống tắc nghẽn

6.1. Tổng quan

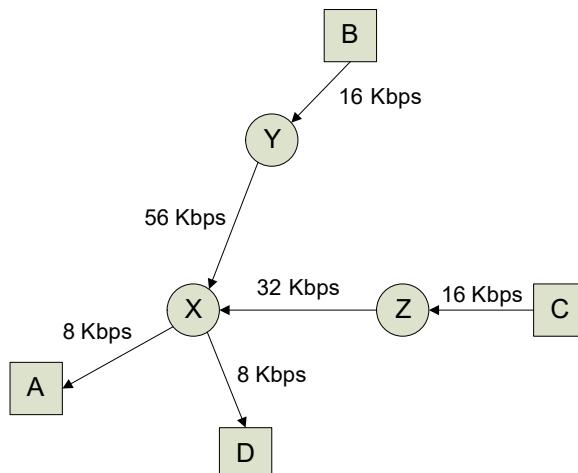
6.1.1. Mở đầu

Trong trao đổi thông tin, khi phía phát truyền dữ liệu đến phía thu thì dữ liệu đầu tiên được lưu trong bộ đệm phía thu. Dữ liệu trong bộ đệm này sau khi được xử lý và chuyển lên các lớp phía trên thì sẽ được xóa đi, để dành bộ đệm cho các dữ liệu kế tiếp.

Trên thực tế trao đổi thông tin trong mạng, có thể xảy ra tình trạng phía phát truyền dữ liệu với tốc độ cao hơn khả năng xử lý của phía thu, dẫn đến bộ đệm của phía thu sẽ đầy dần và bị tràn. Trong trường hợp này, phía thu không thể nhận thêm các gói dữ liệu từ phía phát dẫn đến việc phía phát phải thực hiện truyền lại dữ liệu, gây lãng phí băng thông trên đường truyền.

Nhằm giảm thiểu việc phải truyền lại thông tin vì mất gói do tràn hàng đợi, cần có cơ chế thực hiện kiểm soát và điều khiển lưu lượng thông tin đi đến một thiết bị/mạng. Chức năng này được thực hiện bởi kỹ thuật điều khiển luồng và kiểm soát tắc nghẽn.

Ví dụ 5.1: hoạt động của mạng khi không có sự kiểm soát



Hình: Hoạt động của mạng khi không có sự kiểm soát

Trên hình vẽ này các số trên mỗi liên kết thể hiện tốc độ truyền dữ liệu trên đường đó. Giả sử có hai kết nối từ B đến A (theo đường B – Y – X – A, tốc độ λ_{BA} Kbps) và từ C đến D (theo đường C – Z – X – D, tốc độ λ_{CD} Kbps).

Giả thiết hệ thống mạng không được kiểm soát, nghĩa là tất cả các gói tin đều có thể truy cập tài nguyên của mạng, và bộ đệm tại các nút X, Y và Z có thể được sử dụng bởi bất kỳ gói tin nào. Giả thiết môi trường truyền không có lỗi, lúc này các gói tin không bị sai nhưng vẫn có thể phải được truyền lại nếu nó bị nút mạng hủy do không còn dung lượng bộ đệm để lưu gói tin tạm thời trước khi xử lý. Giả thiết khi gói tin bị mất vì không được lưu trong bộ đệm thì nút phát nó sẽ thực hiện phát lại nhằm đảm bảo việc truyền tin tin cậy.

Để minh họa cho việc điều khiển trong mạng, ta tìm hiểu các trường hợp sau:

1) Trường hợp 1: $\lambda_{BA} = 7 \text{ Kbps}$ và $\lambda_{CD} = 0$.

Trong trường hợp này không xảy ra tắc nghẽn vì lưu lượng từ B đến A sẽ được mạng trung chuyển hết. Tốc độ thông tin đến nút A chính bằng tốc độ thông tin nút B đưa vào mạng, các đường B-Y, Y-X và X-A đều có tốc độ 7 Kbps

2) Trường hợp 2: $\lambda_{BA} = 8 + \delta \text{ Kbps}$ ($\delta > 0$) và $\lambda_{CD} = 0$

Trong trường hợp này, tốc độ thông tin từ B đến A lớn hơn tốc độ hoạt động của đường từ X đến A. Vì lý do này, tốc độ thông tin từ Y đến X lớn hơn từ X đến A, lượng thông tin dư thừa sẽ phải được lưu trong bộ đệm của X. Bộ đệm của X sẽ dần bị đầy và tràn dần đến các gói thông tin từ Y đến sẽ không được lưu và bị hủy. Vì bộ đệm của Y lưu lại các gói tin chưa được báo nhận (để truyền lại) nên bộ đệm của Y cũng dần bị đầy và tràn.

Nút X có thể chuyển 8 Kbps khi lưu lượng đầu vào của nó là $8 + \delta$ Kbps (X hủy δ Kbps). Lúc này, đường Y – X sẽ có tốc độ $8 + 2\delta$ Kbps (trong đó $8 + \delta$ Kbps là thông tin từ B đến và δ Kbps là thông tin phát lại). Nhưng vì nút X chỉ có thể truyền 8 Kbps nên nó hủy 2δ Kbps và Y lại phải truyền lại lượng thông tin này. Quá trình này cứ tiếp diễn và cuối cùng đường nối Y – X sẽ hoạt động với tốc độ 56 Kbps. Tương tự như vậy, đường liên kết từ B đến Y cũng sẽ hoạt động với tốc độ 16 Kbps (bao gồm cả các gói mới và các gói được phát lại)

Để giải quyết vấn đề này, có thể làm theo hai cách:

- Xây dựng hệ thống mạng có khả năng đáp ứng tốc độ của thông tin từ X đến A ($8 + \delta$ Kbps) nhằm đáp ứng với yêu cầu về tốc độ của B – giải pháp này chỉ thực sự khả thi và hiệu quả khi tốc độ phát tin của B là ổn định trong một thời gian dài, nếu không hiệu quả sử dụng tài nguyên rất thấp nếu xây dựng hệ thống mạng có khả năng đáp ứng lưu lượng lớn nhưng lại chỉ hoạt động với các yêu cầu trao đổi lưu lượng nhỏ.
- Giới hạn tốc độ truyền tin của B xuống còn 8 Kbps – phương án này khả thi khi yêu cầu truyền tin của B trong phần lớn thời gian < 8 Kbps và tốc độ vượt 8 Kbps chỉ diễn ra trong thời gian ngắn.

Trong hai phương án này, trên thực tế người ta sử dụng phương án 2 với sự hỗ trợ của các giao thức mạng.

3) Trường hợp 3: $\lambda_{BA} = 7 \text{ Kbps}$ và $\lambda_{CD} = 7 \text{ Kbps}$

Tương tự như trường hợp 1, trường hợp 3 không xảy ra tắc nghẽn trong mạng. Thông tin được chuyển đến A và D với tốc độ 7Kbps cho mỗi nút. Mỗi một liên kết trong mạng sẽ hoạt động với tốc độ 7Kbps

4) Trường hợp 4: $\lambda_{BA} = 8 + \delta \text{ Kbps}$ và $\lambda_{CD} = 7 \text{ Kbps}$ ($\delta > 0$)

Trong trường hợp này, đường đi từ C đến D có đủ dung lượng (tốc độ) để đáp ứng yêu cầu cho kết nối C – D; tuy nhiên yêu cầu truyền thông tin trên đường B – A vượt quá khả năng xử lý của tuyến truyền này.

Trong trường hợp này, hai kết nối B – A và C – D chia sẻ bộ đệm của nút X. Như đã xét trong trường hợp 2, lưu lượng thông tin từ B đến A làm tràn bộ đệm của X, điều này dẫn đến thông tin từ B và C khi đến X đều bị hủy. Hiện tượng này xảy ra đối với tất cả các gói tin (cả B và C) cho dù nguyên nhân gây ra là do B. Hệ quả là nút Y và Z cũng bị tràn bộ đệm và tất cả các đường liên kết sẽ hoạt động với tốc độ cực đại của chúng.

Do trước khi chuyển gói tin từ B và C đến A và D tương ứng, nút X phải lưu các gói tin này vào bộ đệm để xử lý nên trong trường hợp bộ đệm X bị tràn, X sẽ phải hủy các gói tin này. Do tốc độ thông tin Y – X gấp đôi tốc độ thông tin Z – X (khi các liên kết này hoạt động với tốc độ cố định) nên số lượng gói tin từ Y đến X sẽ gấp đôi từ Z đến X. Nói một cách khác, X sẽ hủy (hay chấp nhận) các gói tin từ Y và Z đến theo tỷ lệ 2:1. Lúc này thông tin từ B đến A hoạt động với tốc độ 8 Kbps trong khi thông tin từ C đến D chỉ hoạt động với tốc độ 4 Kbps.

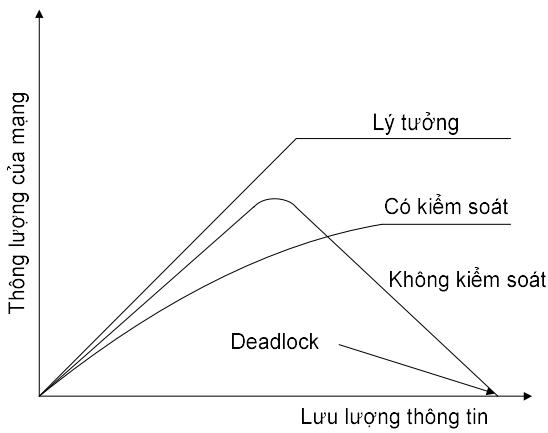
So với trường hợp 3, ta thấy:

- Thông lượng tổng cộng của mạng giảm từ 14 Kbps xuống còn 12 Kbps.
- Nút C bị đổi xử không công bằng vì tốc độ truyền thông tin của nó đến D bị giảm từ 7 Kbps xuống còn 4 Kbps trong khi nút B không bị ảnh hưởng nhiều (giảm từ $8 + \delta$ Kbps xuống 8 Kbps). Ngoài ra, nguyên nhân gây ra tắc nghẽn lại là do nút B.

Để giải quyết vấn đề này, người ta có thể dành một phần dung lượng bộ đệm tại X cho các gói tin từ C đi đến. Việc dành trước tài nguyên này có vẻ như trái ngược với nguyên tắc của chuyển mạch gói khi tài nguyên trong mạng được chia sẻ bởi tất cả các nút và người dùng. Tuy nhiên, trên thực tế người ta có thể đánh đổi điều này để đảm bảo tính công bằng ở trong mạng.

Hình dưới đây mô tả thông lượng của mạng trong mối quan hệ với lưu lượng đầu vào.

- **Thông lượng:** là tốc độ chuyển thông tin của mạng tính theo gói /s
- **Lưu lượng:** là tốc độ thông tin đi đến mạng (bao gồm cả thông tin mới và thông tin được truyền lại)



Hình: Thông lượng của mạng trong mối quan hệ với lưu lượng đầu vào

Trong trường hợp lý tưởng, mạng sẽ thực hiện chuyển tất cả các gói đi vào mạng trong trường hợp tốc độ đến của các gói này nhỏ hơn khả năng trung chuyển của mạng (lưu lượng nhỏ hơn thông lượng). Khi lưu lượng thông tin đến vượt quá thông lượng của mạng, trong trường hợp lý tưởng thì mạng phải có khả năng chuyển các gói với tốc độ bằng thông lượng của mạng (theo đường lý tưởng trên hình vẽ)

Trong trường hợp thực tế, nếu hệ thống mạng không được kiểm soát và có các cơ chế điều khiển, mạng sẽ thực hiện chuyển tất cả các gói tin khi lưu lượng nhỏ hơn một ngưỡng nào đó. Khi lưu lượng vượt quá giá trị ngưỡng thì thông lượng bắt đầu giảm. Lưu lượng đến càng nhiều thì thông lượng càng giảm. Trong một số trường hợp dẫn đến tình trạng deadlock nghĩa là mạng hầu như không chuyển được gói tin nào nữa.

Trong trường hợp có thực hiện điều khiển luồng và điều khiển tắc nghẽn, hệ thống mạng sẽ được kiểm soát và có khả năng hoạt động tốt ngay cả khi có trường hợp quá tải xảy ra (lưu lượng đi vào mạng lớn hơn thông lượng của mạng). Tuy nhiên, do việc thực hiện điều khiển luồng và tắc nghẽn đòi hỏi phải có các thông tin điều khiển nên thông lượng thực tế (trong trường hợp mạng chưa quá tải) sẽ nhỏ hơn trường hợp lý tưởng, thậm chí nhỏ hơn so với trường hợp không có điều khiển.

6.1.2. Khái niệm điều khiển luồng

Định nghĩa – Điều khiển luồng là cơ chế nhằm đảm bảo việc truyền thông tin của phía phát không vượt quá khả năng xử lý của phía thu.

Trong kỹ thuật mạng, điều khiển luồng được chia làm hai loại.

- Điều khiển luồng giữa hai nút đầu cuối (end-to-end): nhằm đảm bảo nút nguồn (nơi khởi tạo phiên thông tin) thực hiện truyền

thông tin không vượt quá khả năng xử lý của nút đích (nơi kết thúc phiên thông tin).

- Điều khiển luồng giữa hai nút trong mạng (hop-by-hop): là việc thực hiện điều khiển luồng giữa hai nút liên tiếp trên đường đi từ nguồn đến đích.

6.1.3. Khái niệm chống tắc nghẽn

Định nghĩa – Chống tắc nghẽn là cơ chế kiểm soát thông tin đi vào mạng nhằm đảm bảo tổng lưu lượng thông tin đi vào mạng không vượt quá khả năng xử lý của toàn mạng.

Chống tắc nghẽn được chia làm hai loại:

- Điều khiển truy nhập mạng** (network access): kiểm soát và điều khiển lượng thông tin có thể đi vào trong mạng.
- Điều khiển cấp bộ đệm** (buffer allocation): là cơ chế thực hiện tại các nút mạng nhằm đảm bảo việc sử dụng bộ đệm là công bằng và tránh việc không truyền tin được do bộ đệm của tất cả các nút bị tràn (deadlock).

Chống tắc nghẽn liên quan đến việc kiểm soát thông tin trên toàn mạng, trong khi điều khiển luồng là việc kiểm soát thông tin giữa hai đầu cuối cụ thể. Hai kỹ thuật này có điểm tương đồng là phải giới hạn lưu lượng thông tin nhằm tránh khả năng quá tải của hệ thống đích. Do tính chất gắn kết của hai khái niệm này, đa phần các tài liệu đều sử dụng lẫn (hoặc kết hợp) các khái niệm điều khiển luồng (flow control) và điều khiển tắc nghẽn (congestion control).

Vì lý do đó, trong tài liệu này, chúng tôi sử dụng khái niệm điều khiển luồng để diễn tả cả hai phạm trù. Trong những trường hợp cụ thể cần phân biệt làm rõ hai khái niệm, chúng tôi sẽ có những chú thích rõ ràng.

6.1.4. Nhiệm vụ chủ yếu của điều khiển luồng và chống tắc nghẽn

Điều khiển luồng và chống tắc nghẽn được sử dụng khi có sự giới hạn về tài nguyên (thường là băng thông) giữa điểm truy nhập thông tin. Khái niệm điểm truy nhập ở đây có thể là giữa hai người sử dụng, giữa người sử dụng với điểm truy nhập mạng hay giữa hai thiết bị mạng.

Mục đích chính của việc sử dụng điều khiển luồng và chống tắc nghẽn trong mạng là nhằm:

- Tối ưu hóa thông lượng sử dụng của mạng:** trong trường hợp thông tin chỉ truyền giữa hai người dùng, việc tối ưu hóa tốc độ truyền tin không cần đặt ra. Tuy nhiên, trong một hệ thống mạng với sự tham gia trao đổi thông tin của nhiều nút mạng, việc tối ưu hóa thông lượng của hệ thống mạng phức tạp hơn nhiều.

- **Giảm trễ gói khi đi qua mạng:** đứng trên phương diện người sử dụng, trễ gói từ đầu cuối đến đầu cuối càng nhỏ càng tốt. Tuy nhiên, điều khiển luồng (ở lớp mạng) không nhằm thực hiện điều đó. Điều khiển luồng chỉ đảm bảo trễ của gói tin khi đi qua mạng nằm ở một mức chấp nhận được thông qua việc giới hạn số lượng gói tin đi vào mạng (và do đó, giảm trễ hàng đợi). Vì lý do đó, điều khiển luồng không có tác dụng với những ứng dụng đòi hỏi trễ nhỏ trong khi lại truyền trên hệ thống hạ tầng tốc độ thấp. Trong trường hợp này, việc đáp ứng yêu cầu của người sử dụng chỉ có thể được thực hiện thông qua việc nâng cấp hệ thống hay sử dụng các giải thuật định tuyến tối ưu hơn. Mục đích chính của việc giảm trễ gói là để giảm sự lãng phí tài nguyên khi phải truyền lại gói. Việc truyền lại có thể do hai nguyên nhân: (1) hàng đợi của các nút mạng bị đầy dẫn đến gói thông tin bị hủy và phải truyền lại; (2) thông tin báo nhận quay trở lại nút nguồn quá trễ khiến phía phát cho rằng thông tin truyền đi đã bị mất và phải truyền lại.
- **Đảm bảo tính công bằng cho việc trao đổi thông tin trên mạng:** đảm bảo tính công bằng trong trao đổi thông tin là một trong những yếu tố tiên quyết của kỹ thuật mạng. Việc đảm bảo tính công bằng cho phép người sử dụng được dùng tài nguyên mạng với cơ hội như nhau. Trong trường hợp người sử dụng được chia thành các nhóm với mức độ ưu tiên khác nhau thì bảo đảm tính công bằng được thực hiện đối với các người dùng trong cùng một nhóm.
- **Đảm bảo tránh tắc nghẽn trong mạng:** tắc nghẽn là hiện tượng thông lượng của mạng giảm và trễ tăng lên khi lượng thông tin đi vào mạng tăng. Điều khiển luồng cung cấp cơ chế giới hạn lượng thông tin đi vào mạng nhằm tránh hiện tượng tắc nghẽn kể trên. Có thể hình dung điều khiển luồng như hoạt động của cảnh sát giao thông trên đường phố vào giờ cao điểm.

Như trên đã trình bày, điều khiển luồng và tránh tắc nghẽn thường được sử dụng kết hợp với nhau để kiểm soát thông tin trên mạng. Điều khiển luồng và tránh tắc nghẽn được sử dụng nhiều nhất tại các lớp liên kết dữ liệu (data link), lớp mạng (network) và lớp giao vận (transport) trong đó điều khiển luồng hop-by-hop được sử dụng ở lớp liên kết dữ liệu, điều khiển luồng end-to-end được sử dụng ở lớp giao vận và điều khiển tắc nghẽn được sử dụng ở lớp mạng.

6.1.5. Phân loại điều khiển luồng và tránh tắc nghẽn

Trong các phần tới, chúng ta sẽ lần lượt tìm hiểu các cơ chế và chính sách thực hiện điều khiển luồng và tránh tắc nghẽn. Các cơ chế này được phân ra làm ba loại chính:

- Các cơ chế cấp phát bộ đệm
- Các cơ chế cửa sổ

- Các cơ chế điều khiển truy nhập mạng

6.2. Tính công bằng

6.2.1. Định nghĩa

Định nghĩa – Tính công bằng là khả năng đảm bảo cho các người dùng, các ứng dụng khác nhau được sử dụng tài nguyên mạng với cơ hội như nhau.

Đảm bảo tính công bằng là một trong những tiêu chí hàng đầu của kỹ thuật mạng.

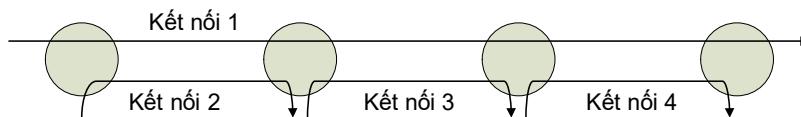
Ví dụ: xem lại ví dụ đầu chương (ví dụ số...) để thấy được tính công bằng.

6.2.2. Tính công bằng về mặt băng truyền

Định nghĩa – Tính công bằng về mặt băng truyền thể hiện ở khả năng chia sẻ băng truyền công bằng cho tất cả người dùng hoặc kết nối.

Ví dụ 5.2: Xét mô hình mạng như trên hình vẽ dưới đây. Liên kết giữa các nút có tốc độ 1Mbps.

- Thông lượng của mạng sẽ đạt cực đại (bằng 3Mbps) nếu các kết nối 2, 3 và 4 được sử dụng toàn bộ 1 Mbps băng thông và kết nối 1 không được cung cấp lượng băng thông nào cả
- Một khái niệm khác của tính công bằng là cho mỗi kết nối sử dụng 0,5Mbps băng thông. Lúc này tổng thông lượng của mạng sẽ là 2Mbps.
- Nếu cung cấp lượng tài nguyên mạng (băng thông) cho tất cả các kết nối là như nhau, lúc ấy các kết nối 2, 3, 4 sẽ được sử dụng 0,75Mbps và kết nối 1 sử dụng 0,25 Mbps (và được sử dụng trên toàn bộ đường truyền)



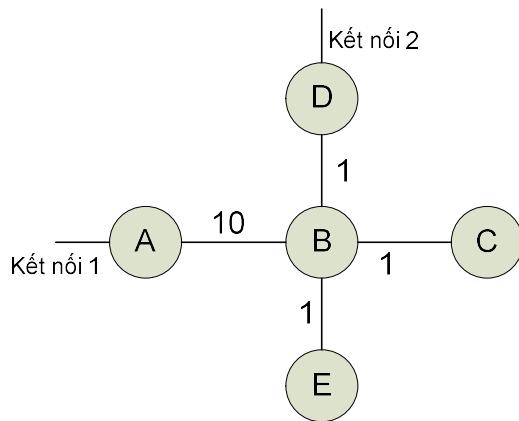
Hình: Minh họa sự đánh đổi giữa thông lượng và tính công bằng

6.2.3. Tính công bằng về mặt bộ đệm

Hình vẽ dưới minh họa khái niệm sử dụng bộ đệm

- Giả sử nút mạng B có dung lượng bộ đệm hữu hạn

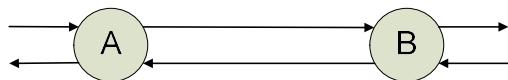
- Liên kết 1 (từ A đến B) có tốc độ 10Mbps, liên kết 2 (từ D đến B) có tốc độ 1 Mbps.
- Nếu không có cơ chế điều khiển luồng và quản lý bộ đệm, tỷ lệ sử dụng dung lượng bộ đệm tại B của hai liên kết 1 và 2 sẽ là 10:1 (do tốc độ thông tin đến B tương ứng là 10Mbps và 1Mbps)



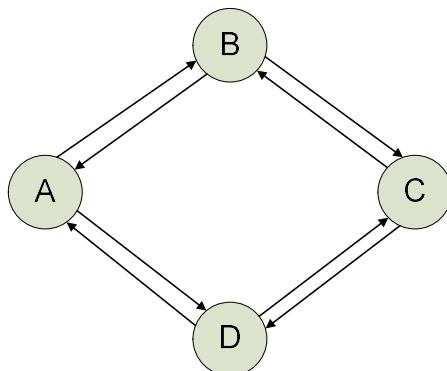
Hình: Minh họa về sự không công bằng khi sử dụng bộ đệm

Hình vẽ dưới minh họa hiện tượng tắc nghẽn xảy ra do tràn bộ đệm.

- Trong hình (a), bộ đệm của nút A đã được điền đầy bởi thông tin đến từ B và ngược lại. Kết quả là A và B không nhận được thêm thông tin từ nhau và việc truyền thông tin là không thực hiện được (deadlock).
- Trong hình (b), giả sử bộ đệm của A đầy các gói thông tin của B, bộ đệm của B đầy thông tin của C và bộ đệm của C đầy các thông tin của A. Tương tự như trường hợp hình A, trong trường hợp này, việc truyền tin cũng không thực hiện được do tràn bộ đệm.



(a): Direct Deadlock



(b): Indirect Deadlock

Hình: Tắc nghẽn do tràn bộ đệm

Định nghĩa – Tính công bằng về mặt bộ đệm là khả năng đảm bảo việc sử dụng bộ đệm của các người dùng, các ứng dụng hay kết nối là công bằng.

Với việc sử dụng cơ chế điều khiển luồng và các cơ chế quản lý bộ đệm, việc phân chia sử dụng bộ đệm giữa các người dùng, ứng dụng hay các kết nối sẽ được thực hiện công bằng hơn.

6.2.4. Cơ chế phát lại ARQ

Các cơ chế điều khiển luồng và điều khiển tắc nghẽn theo phương pháp cửa sổ được hoạt động tương tự như các cơ chế phát lại ARQ (Automatic Repeat Request). Vì lý do đó, trong phần này, chúng tôi trình bày các khái niệm cơ bản về các cơ chế ARQ làm nền tảng cho việc tìm hiểu về điều khiển luồng và điều khiển tắc nghẽn ở các phần sau.

Khi truyền thông tin trong mạng, thông tin truyền từ phía phát sang phía thu có thể bị sai lỗi hoặc mất. Trong trường hợp thông tin bị mất, cần phải thực hiện truyền lại thông tin. Với trường hợp thông tin bị sai, có thể sửa sai bằng một trong hai cách:

- Sửa lỗi trực tiếp bên thu: phía thu sau khi phát hiện lỗi có thể sửa lỗi trực tiếp ngay bên thu mà không yêu cầu phải phát lại. Để có thể thực hiện được điều này, thông tin trước khi truyền đi phải được cài các mã sửa lỗi (bên cạnh việc có khả năng phát hiện lỗi, cần có khả năng sửa lỗi).

- Yêu cầu phía phát truyền lại: phía thu sau khi kiểm tra và phát hiện có lỗi sẽ yêu cầu phía phát truyền lại thông tin.

Đặc điểm của hai phương pháp sửa lỗi trên:

- Sửa lỗi trực tiếp bên thu (Forward Error Correction – FEC): chỉ cần truyền thông tin một lần, không yêu cầu phải truyền lại thông tin trong trường hợp có lỗi. Tuy nhiên, số lượng bit thông tin có thể sửa sai phụ thuộc vào số loại mã sửa sai và số bit thông tin thêm vào cho mục đích sửa sai. Nhìn chung, số bit thông tin thêm vào càng lớn thì số bit có thể sửa sai càng nhiều, tuy nhiên hiệu suất thông tin (số bit thông tin hữu ích trên tổng số bit truyền đi) lại thấp.
- Sửa lỗi bằng cách truyền lại: khác với sửa lỗi trực tiếp bên thu, trong trường hợp sửa lỗi bằng cách truyền lại, thông tin trước khi phát chỉ cần thêm các bit thông tin phục vụ cho mục đích phát hiện lỗi (số bit thêm vào ít hơn so với trường hợp sửa lỗi) do đó hiệu suất truyền thông tin cao hơn so với trường hợp trên. Tuy nhiên, trong trường hợp có lỗi xảy ra với khung thông tin thì toàn bộ khung thông tin phải được truyền lại (giảm hiệu suất truyền tin).
- Với ưu nhược điểm của các phương pháp trên, sửa lỗi bằng cách truyền lại thường được dùng trong môi trường có tỷ lệ lỗi bit thấp (truyền dẫn hữu tuyến) trong khi sửa lỗi bên thu thường được dùng trong trường hợp môi trường truyền dẫn có tỷ lệ lỗi bit cao (vô tuyến). Để có thể đối phó với trường hợp lỗi chùm (burst noise), có thể áp dụng một số cơ chế như ghép xen kẽ thông tin (interleaving).

Trong khuôn khổ chương này, chúng tôi trình bày việc điều khiển lỗi theo cơ chế phát lại. Các cơ chế này được gọi là ARQ (Automatic Repeat Request). Cơ chế sửa lỗi trực tiếp bên thu được trình bày trong các nội dung của môn học khác.

Các cơ chế phát lại được chia ra làm 3 loại chính:

- Cơ chế phát lại dừng và đợi** (Stop-and-Wait ARQ)
- Cơ chế phát lại theo nhóm** (Go-back-N ARQ)
- Cơ chế phát lại có lựa chọn** (Selective repeat ARQ)

Phản dưới đây sẽ là lần lượt trình bày nguyên tắc hoạt động cũng như đánh giá hiệu năng của mỗi phương pháp.

6.2.5. Stop-and-Wait ARQ

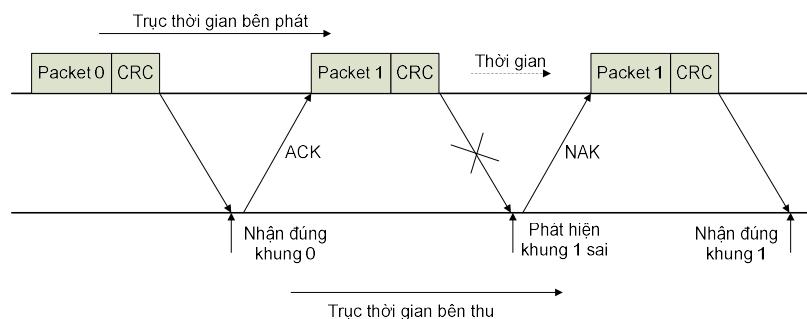
Cơ chế hoạt động

Trong cơ chế phát lại theo phương pháp dừng và đợi (Stop-and-Wait ARQ), phía phát sẽ thực hiện phát một khung thông tin sau đó dừng lại, chờ phía thu báo nhận.

- Phía thu khi nhận đúng khung thông tin và xử lý xong sẽ gửi báo nhận lại cho phía phát. Phía phát sau khi nhận được báo nhận sẽ phát khung thông tin tiếp theo.
- Phía thu khi nhận khung thông tin và phát hiện sai sẽ gửi báo sai lại cho phía phát. Phía phát sau khi nhận được báo sai sẽ thực hiện phát lại khung thông tin.

Báo nhận được sử dụng cho khung thông tin đúng và được gọi là **ACK** (viết tắt của chữ Acknowledgement). Báo sai được sử dụng cho khung thông tin bị sai và được gọi là **NAK** (viết tắt của chữ Negative Acknowledgement).

Hình vẽ dưới đây mô tả nguyên tắc hoạt động cơ bản của cơ chế phát lại dừng và đợi.



Hình: Phát lại theo cơ chế dừng và đợi

- 5) Câu hỏi:** Trong trường hợp phía phát không nhận được thông tin gì từ phía thu, phía phát sẽ làm gì?

Phía phát không nhận được thông tin từ phía thu trong hai trường hợp:

- Khung thông tin bị mất, phía thu không nhận được gì và cũng không gửi thông báo cho phía phát.
- Phía thu đã nhận được đúng khung thông tin và gửi ACK rồi, nhưng ACK bị mất; hoặc phía thu nhận được khung thông tin và phát hiện sai và đã gửi NAK nhưng khung này bị mất.

Để tránh tình trạng phía phát không phát thông tin do chờ ACK (hoặc NAK) từ phía thu, mỗi khi phát một khung thông tin, phía phát sẽ đặt một đồng hồ đếm ngược (time-out) cho khung thông tin đó. Hết khoảng thời gian time-out, nếu phía phát ko nhận được thông tin gì từ phía thu thì nó sẽ chủ động phát lại khung thông tin bị time-out.

- 6) Câu hỏi:** Trong trường hợp phía phát phải phát lại khung thông tin do time-out, nhưng khung thông tin đó đã được nhận đúng ở phía thu rồi (time-out xảy ra do ACK bị mất), phía thu làm thế nào để có thể phân biệt là khung thông tin này là khung phát lại hay khung thông tin mới?

Để có thể phân biệt được các khung thông tin với nhau, cần đánh số khác khung. Trong trường hợp này, chỉ cần dùng một bit để đánh số khung (0 hoặc 1).

Để tránh tình trạng các nhầm lẫn giữa các khung thông tin được phát và báo nhận tương ứng, tất cả các khung đều được truyền đi giữa hai phía phát – thu đều được đánh số (0, 1) luân phiên. Số thứ tự khung thông tin từ phía phát sang phía thu nằm trong trường *SN* (Sequence Number) và số thứ tự của báo nhận từ phía thu sang phía phát nằm trong trường *RN* (Request Number). *SN* là số thứ tự được khởi tạo ở bên phát, trong khi đó, *RN* là số thứ tự của khung tiếp theo mà phía thu muốn nhận. $RN = SN + 1$ trong trường hợp khung đúng (ứng với ACK), $RN = SN$ trong trường hợp phía thu yêu cầu phát lại do khung sai (ứng với NAK).

Trên thực tế, thông tin trao đổi giữa hai điểm thường được truyền theo hai chiều, nghĩa là đồng thời tồn tại hai kênh truyền từ phát đến thu và ngược lại. Trong trường hợp này, khung ACK/NAK (hay trường *RN*) không cần nằm trong một khung báo nhận độc lập mà có thể nằm ngay trong tiêu đề của khung thông tin được truyền theo chiều từ thu đến phát. Một số giao thức có khung thông tin báo nhận độc lập (ACK/NAK) trong khi một số giao thức khác lại sử dụng luôn khung thông tin truyền theo chiều ngược lại (từ thu sang phát) để thực hiện báo nhận (hay báo lỗi) cho khung thông tin từ phát sang thu

Tóm tắt cơ chế hoạt động của Stop-and-Wait ARQ

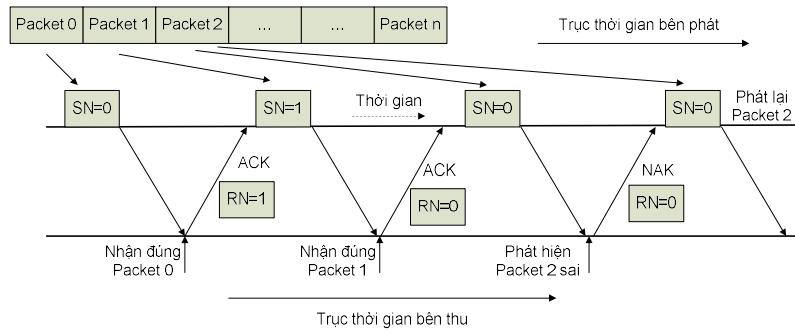
Phía phát – giả sử tại thời điểm đầu $SN = 0$

- 1) Nhận gói tin từ lớp phía trên và gán *SN* cho gói tin này
- 2) Gửi gói tin *SN* này trong một khung thông tin có số thứ tự là *SN*
- 3) Chờ khung thông tin (không có lỗi, đóng vai trò là khung báo nhận) từ phía thu.
 - Nếu khung nhận được không có lỗi, và trong trường *Request* có $RN > SN$ thì đặt giá trị $SN = RN$ và quay lại bước 1
 - Nếu không nhận được khung thông tin trong một khoảng thời gian định trước (time-out), thì thực hiện bước 2

Phía thu – giả sử tại thời điểm đầu $RN = 0$

- 4) Khi nhận được một khung thông tin (không có lỗi) từ phía phát, chuyển khung này lên lớp phía trên và tăng giá trị *RN* lên 1
- 5) Trong trường hợp nhận được khung thông tin có lỗi, gửi lại một khung thông tin cho phía phát với *RN* được giữ nguyên (khung báo sai - NAK). Khung được gửi từ phía thu này có thể chứa cả thông tin từ phía thu sáng phía phát chứ không đơn thuần chỉ dùng cho mục đích báo sai.

Hình dưới đây mô tả nguyên tắc hoạt động của cơ chế Stop-and-Wait ARQ khi có sử dụng *SN* và *RN*.



Hình: Stop-and-Wait ARQ có sử dụng SN/RN

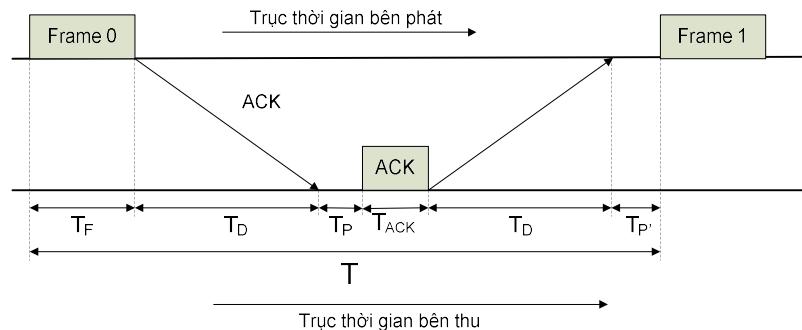
Hiệu suất của phương pháp Stop-and-Wait ARQ

Định nghĩa – Hiệu suất của việc truyền tin giữa phía phát và thu là tỷ lệ giữa thời gian phía phát cần để phát xong lượng thông tin đó trên tổng thời gian cần thiết để truyền lượng thông tin đó.

Tổng thời gian cần thiết ở đây bao gồm thời gian trễ khi truyền tín hiệu từ phát sang thu (và ngược lại) cũng như thời gian xử lý thông tin và thời gian chờ báo nhận từ phía thu.

Để tính hiệu suất tính cho phương pháp ARQ dừng và đợi, người ta tính cho một khung thông tin điển hình, hiệu suất của cả một phiên truyền cho nhiều khung thông tin về bản chất chính bằng hiệu suất khi tính cho một khung (vì cả tử số và mẫu số cùng nhân với một hệ số tỷ lệ là số khung thông tin được truyền)

Trường hợp 1: Giả thiết môi trường không có lỗi, thông tin từ truyền từ phía phát sang phía thu chỉ chịu ảnh hưởng của trễ



Hình: Giản đồ thời gian khi truyền tin từ phát sang thu, không có lỗi

Trong đó:

- T_F = thời gian phát khung thông tin
- T_D = trễ truyền sóng giữa phía phát và phía thu

- T_P = thời gian xử lý khung thông tin ở phía thu
- T_{ACK} = thời gian phát khung ACK
- $T_{P'}$ = thời gian xử lý khung ACK ở phía phát

Ta có:

- Thời gian phía phát cần để phát xong khung thông tin là T_F
- Tổng thời gian cần thiết để truyền khung thông tin là $T = T_F + T_D + T_P + T_{ACK} + T_D + T_{P'}$. Vì thời gian xử lý khung thông tin T_P và $T_{P'}$ là khá nhỏ nên có thể bỏ qua. Trong trường hợp kích thước khung thông tin F lớn hơn khung báo nhận ACK rất nhiều thì có thể bỏ qua cả T_{ACK} . Như vậy $T = T_F + 2T_D$.

Hiệu suất truyền:

$$\eta = \frac{T_F}{T_F + 2T_D} = \frac{1}{1+2a} \text{ với } a = \frac{T_D}{T_F}$$

Trong đó:

$T_D = \frac{d}{v}$ với d là khoảng cách giữa hai trạm phát và thu; v là vận tốc truyền sóng trong môi trường. $v = 3.10^8$ m/s khi truyền trong không gian tự do.

$T_F = \frac{L}{R}$ với L là kích thước khung thông tin và R là tốc độ đường truyền

Khi đó $a = \frac{Rd}{vL}$, a càng nhỏ thì hiệu suất truyền càng lớn

Ví dụ 5.3: tính hiệu suất của phương pháp phát lại theo cơ chế ARQ dùng và đợi cho tuyến thông tin vệ tinh. Giả thiết khoảng cách từ vệ tinh tới mặt đất là 36.000 km, vận tốc truyền sóng trong không khí là 3.10^8 m/s, tốc độ thông tin là 56 Kbps và khung có kích thước 4000 bits.

Giải: Ta có

$$a = \frac{Rd}{vL} = \frac{56.10^3.36.10^6}{3.10^8.4.10^3} = 1,68,$$

Do đó hiệu suất

$$\eta = \frac{1}{1+2a} = \frac{1}{1+2.1,68} = 22,94\%.$$

Hiện tại, các dịch vụ thông tin vệ tinh có tốc độ lớn hơn nhiều (R lớn) nên hệ số a càng lớn và hiệu suất sẽ còn nhỏ hơn trường hợp ví dụ này.

Ví dụ 5.4: tính hiệu suất của phương pháp phát lại theo ví dụ trên nhưng sử dụng co kết nối trong mạng LAN với khoảng cách giữa hai

trạm là 100 m, vận tốc truyền sóng trên cáp đồng là $2 \cdot 10^8$ m/s, tốc độ truyền thông tin là 10 Mbps và khung có kích thước 500 bits.

Giải: tính tương tự như trường hợp trên, ta có

$$a = \frac{Rd}{vL} = \frac{10 \cdot 10^6 \cdot 100}{2 \cdot 10^8 \cdot 500} = 0,01, \text{ hiệu suất } \eta = \frac{1}{1+2a} = \frac{1}{1+2 \cdot 0,01} = 98,04\%$$

Như vậy, với thông tin trong mạng LAN, do cự ly nhỏ nên hiệu suất được cải thiện so với trường hợp truyền thông tin vệ tinh.

- 6) **Trường hợp 2:** ở phần trên, để tính toán hiệu suất, chúng ta đã giả thiết môi trường truyền lý tưởng (không có lỗi). Tuy nhiên, môi trường truyền thực tế luôn có lỗi và được đặc trưng bởi xác suất lỗi p , do đó, hiệu suất truyền trên thực tế sẽ nhỏ hơn so với trường hợp lý tưởng.

Định nghĩa xác suất lỗi – Xác suất lỗi p ($0 \leq p \leq 1$) là xác suất phía thu nhận được bit 0 khi phía phát truyền bit 1 (hoặc ngược lại).

Xác suất lỗi càng lớn thì môi trường truyền càng không tốt, khi $p = 0$ thì môi trường truyền không có lỗi (lý tưởng); $p = 1$ là khi môi trường truyền luôn luôn có lỗi (sẽ không dùng để truyền tin).

Khi $0,5 < p < 1$ tức là khả năng phía thu nhận được thông tin có lỗi sẽ lớn hơn nhận được thông tin đúng, trong trường hợp này, chỉ cần đảo bit luồng thông tin thu được là ta có thể chuyển thành trường hợp $0 < p < 0,5$. Vì lý do đó, trong lý thuyết thông tin, người ta chỉ tìm hiểu các môi trường truyền dẫn có xác suất lỗi $0 \leq p \leq 0,5$.

Như trên đã trình bày, khi truyền thông tin trong môi trường có lỗi, có thể xảy ra trường hợp phải truyền lại khung thông tin (do lỗi), do đó, hiệu suất trong trường hợp này nhỏ hơn trường hợp lý tưởng. Gọi N_R là số khung thông tin phải truyền cho đến khi đúng ($1 \leq N_R \leq \infty$), khi ấy, hiệu suất của trường hợp không lý tưởng sẽ là $\eta'_{\text{reality}} = \frac{\eta_{\text{ideal}}}{N_R}$. Vấn đề ở đây là tính được giá trị N_R . Để đơn giản hóa, ta giả thiết ACK và NAK không bị lỗi. Ta thấy, với xác suất lỗi là p thì:

- Xác suất để truyền khung thành công ngay lần đầu là $1-p$
- Xác suất để truyền khung đến lần thứ hai mới thành công là $p(1-p)$
- Tổng quát hóa: xác suất để truyền khung đến lần thứ i mới thành công là $p^{i-1}(1-p)$

Vậy:

$$N_R = \sum_{i=1}^{\infty} ip^{i-1}(1-p) = \frac{1}{1-p}.$$

Hiệu suất của phương pháp ARQ dừng và đợi trong trường hợp thực tế:

$$\eta_{\text{reality}} = \frac{\eta_{\text{ideal}}}{N_R} = \frac{1-p}{1+2a}$$

Nhận xét

Như phần trên đã trình bày, hiệu suất của phương pháp truyền theo cơ chế dừng và đợi phụ thuộc vào hệ số $a = \frac{Rd}{vL}$, a càng nhỏ thì hiệu suất càng lớn. Ta thấy a sẽ nhỏ khi $v.L$ lớn hoặc khi $R.d$ nhỏ.

- R nhỏ – đây là điều không mong muốn khi truyền thông tin vì trên thực tế, người ta mong muốn truyền tin với tốc độ đường truyền càng cao càng tốt.
- d nhỏ – tham số khoảng cách giữa phía phát và phía thu thường không thay đổi được do phụ thuộc vào những yêu cầu khách quan bên ngoài.
- v lớn – vận tốc truyền sóng trong môi trường có các giá trị nhất định và rất khó có thể thay đổi.
- L lớn – có thể tăng kích thước khung để tăng hiệu suất. Tuy nhiên phương pháp này có nhược điểm là thông tin truyền lại sẽ lớn nếu khung thông tin ban đầu bị sai. Cũng vì lý do này mà mỗi môi trường truyền dẫn nhất định sẽ có kích thước khung tối ưu tương ứng.

Như vậy, hệ số a gần như không thể thay đổi dẫn đến phương pháp truyền lại theo cơ chế dừng và đợi không được sử dụng phổ biến do hiệu quả sử dụng đường truyền không cao. Để nâng hiệu suất lên, cần có những cơ chế mới nhằm đảm bảo phía phát có thể tận dụng được thời gian rảnh rỗi trong khi chờ báo nhận từ phía thu. Người ta đã dựa trên cơ chế dừng và đợi này để tạo ra các cơ chế mới cho hiệu quả truyền cao hơn, cụ thể là cơ chế truyền lại theo nhóm (Go-back-N ARQ) và cơ chế phát lại yêu cầu (Selective Repeat ARQ).

6.2.6. Go-back-N ARQ

Cơ chế hoạt động

Với cơ chế phát lại Go-back-N, phía phát sẽ được phát nhiều hơn một khung thông tin trước khi nhận được báo nhận từ phía thu. Số khung thông tin cực đại mà phía phát có thể phát (ký hiệu là W) được gọi là kích thước cửa sổ. Với cơ chế hoạt động này, Go-back-N (và cả phương pháp selective repeat trình bày ở phần sau) được gọi là cơ chế cửa sổ trượt (sliding window)

Mỗi khi phát xong một khung thông tin, phía phát giảm kích thước cửa sổ đi 1, khi kích thước cửa sổ bằng 0, phía phát sẽ không được phát thêm khung thông tin nào nữa (điều này đảm bảo số khung thông tin

đồng thời đến phía thu không vượt quá W , và do đó, không vượt quá khả năng xử lý của phía thu).

Mỗi khi phía thu nhận được một khung thông tin đúng và xử lý xong, phía thu sẽ gửi lại một báo nhận ACK cho phía phát. Khi nhận được báo nhận này, phía phát sẽ tăng kích thước cửa sổ W lên 1. Điều này đồng nghĩa với việc phía phát sẽ được phép thêm một khung nữa, ngoài W khung đã phát trước đó, vì phía thu đã xử lý xong một khung, và như vậy, tổng số khung mà phía thu phải xử lý tại một thời điểm vẫn không vượt quá W .

Để có thể phân biệt các khung trên đường truyền, các khung cần được đánh số thứ tự. Nếu dùng k bit để đánh số thì tổng số khung được đánh số sẽ là 2^k (từ 0 đến $2^k - 1$) và do đó, kích thước cửa sổ tối đa $W_{max} = 2^k$ (về mặt lý thuyết).

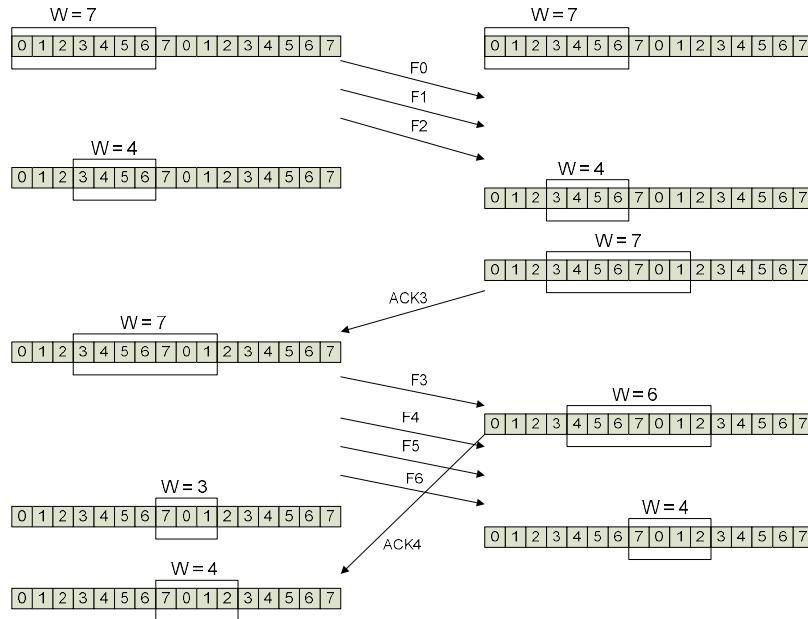
Ví dụ sử dụng 3 bit để đánh số thứ tự cho các khung thông tin. Lúc này kích thước cửa sổ cực đại sẽ là 7 (yêu cầu sinh viên giải thích lý do). Tại thời điểm ban đầu, cả phía phát và thu đều có kích thước cửa sổ là 7 thể hiện rằng phía phát được phép phát tối đa là 7 khung (bắt đầu từ khung $F0$). Sau khi phía phát đã phát được ba khung ($F0, F1, F2$) và chưa nhận được ACK, phía phát giảm kích thước cửa sổ xuống còn 4. Lúc này cửa sổ phía phát bao gồm các khung từ $F3$ đến $F6$ thể hiện rằng phía phát còn được phép truyền tối đa là 4 khung nữa, bắt đầu từ khung $F3$.

Ở phía thu, sau khi đã nhận đúng và xử lý xong ba khung $F0, F1$ và $F2$ thì sẽ gửi lại ACK3 cho phía phát. ACK3 nhằm ám chỉ rằng: "Phía thu đã nhận và xử lý xong các khung cho đến $F2$ và phía thu đang sẵn sàng nhận khung 3." Thực tế, phía thu sẵn sàng nhận 7 khung bắt đầu từ khung $F3$. Phía thu đồng thời tăng kích thước cửa sổ bên thu lên 7, bao các khung từ $F3$ cho đến $F1$.

Phía phát sau khi nhận được ACK3 sẽ tăng kích thước cửa sổ thêm 3 đơn vị. Lúc này cửa sổ phía phát $W = 7$ và bao các khung từ $F3$ đến $F1$. Giả sử lúc này phía phát thực hiện phát các khung từ $F3$ đến $F6$ (4 khung). Sau khi phát, phía phát sẽ giảm kích thước cửa sổ đi 4 ($W = 3$), lúc này cửa sổ chỉ còn bao các khung $F7, F0$ và $F1$.

Phía thu gửi lại ACK4, báo rằng nó đã nhận và xử lý xong khung $F3$, ACK4 ám chỉ rằng phía phát được phép phát tối đa là 7 khung bắt đầu từ $F4$. Tuy nhiên khi ACK4 về đến phía phát thì phía phát đã thực hiện phát các khung $F4, F5$ và $F6$ rồi, như vậy, phía phát sẽ chỉ còn phát được tối đa là 4 khung bắt đầu từ $F7$.

Hình dưới đây minh họa nguyên tắc hoạt động của cơ chế cửa sổ trượt.



Hình: Nguyên tắc hoạt động của cơ chế cửa sổ trượt

Trong trường hợp lý tưởng (không có lỗi xảy ra) thì cơ chế cửa sổ trượt đảm bảo số khung thông tin từ phía phát đến phía thu không vượt quá kích thước cửa sổ. Trong trường hợp này, không có sự phân biệt giữa Go-back-N và selective repeat (và chúng được gọi chung là sliding window).

Khi có lỗi xảy ra, việc truyền lại các khung lỗi của cơ chế cửa sổ trượt được thực hiện theo hai cách khác nhau:

- Go-back-N: phía phát sẽ thực hiện phát lại khung thông tin bị sai và tất cả các khung thông tin khác đã được truyền, tính từ khung bị sai.
- Selective repeat: phía phát sẽ chỉ phát lại các khung thông tin bị sai

Để có thể hiểu rõ hơn về cơ chế hoạt động của Go-back-N, ta xét một số trường hợp cụ thể sau:

1) Khung thông tin bị lỗi – có thể xảy ra một trong ba trường hợp:

- Phía phát đã phát khung i , phía thu đã thu đúng các khung từ $i - 1$ trở về trước. Lúc này phía thu sẽ gửi $NAK i$ ($RN = i$) cho phía phát để báo lỗi cho khung i . Khi phía phát nhận được $NAK i$, nó sẽ thực hiện phát lại khung i và tất cả các khung sau i (nếu các khung đó đã được phát).
- Khung thông tin i bị mất trên đường truyền, giả sử phía thu nhận được khung $i+1$, lúc này phía thu thấy các khung đến không theo thứ tự (nhận được $i+1$ trước khi nhận được i) và

hiểu rằng khung i đã mất. Phía thu sẽ gửi lại $NAK\ i$ cho phía phát.

- Khung thông tin i bị mất trên đường truyền và phía phát không gửi thêm khung thông tin nào nữa. Lúc này phía thu không nhận được gì và không gửi lại ACK hay NAK. Phía phát chờ đến time-out của khung thông tin i và thực hiện truyền lại khung này.

2) Khung ACK bị lỗi – ACK bị lỗi có thể xảy ra một trong hai trường hợp:

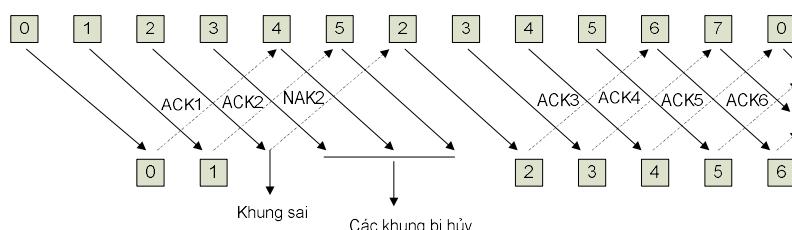
- Phía thu nhận được khung i và gửi $ACK(i+1)$ về phía phát và ACK này bị mất trên đường truyền. Giả sử trước khi time-out của khung i xảy ra, phía phát nhận được $ACK(i+2)$ (hoặc $ACK(i+n)$ với $n > 1$) thì phía phát hiểu rằng khung i đã được nhận đúng. Kết luận này được giải thích như sau: khi phía thu gửi $ACK(i+2)$ nghĩa là phía thu đã nhận đúng (và chấp nhận) khung $i+1$, điều đó cũng đồng nghĩa với việc phía thu đã nhận đúng khung i . Người ta nói cơ chế của Go-back-N sử dụng **cummulative ACK** (nghĩa là các ACK sau cũng đồng thời báo nhận cho các khung trước đó)

- Nếu trong khoảng thời gian time-out của khung i , phía phát không nhận được $ACK(i+n)$ nào cả thì sau time-out, phía phát sẽ phải phát lại khung i (và tất cả các khung sau đó).

3) Khung NAK bị lỗi – trong trường hợp NAK bị lỗi, nghĩa là khung i bị lỗi, lúc này phía thu sẽ không nhận thêm một khung nào sau khung i (và cũng sẽ không gửi báo nhận). Với trường hợp này phía phát bắt buộc phải chờ đến time-out và thực hiện phát lại khung thông tin i .

4) Để đơn giản vấn đề, chúng ta không xét trường hợp ACK và NAK bị sai (nếu xét thì sẽ như thế nào???)

Hình 1-8 đây trình bày nguyên tắc phát lại của ARQ Go-back-N khi có lỗi xảy ra với khung thông tin



Hình 5-8: Minh họa cơ chế Go-back-N ARQ

Một số chú ý của cơ chế hoạt động ARQ Go-back-N

Bên cạnh nguyên tắc hoạt động và minh họa đã trình bày trên đây, cần chú ý một số điểm sau khi tìm hiểu hoạt động của Go-back-N

- Trong trường hợp phía thu có khả năng xử lý W khung thông tin thì không cần bộ đệm. Phía thu chỉ nhận và xử lý thông tin theo đúng thứ tự (dựa trên số thứ tự đánh trên các khung)
 - Phía thu chuyển các gói thông tin lên lớp cao hơn theo thứ tự
 - Phía thu sẽ không nhận khung $i+1$ nếu chưa nhận được khung i . Điều này là nguyên nhân khiến phía thu không cần phải có bộ đệm
 - Phía phát phải lưu tối đa là W khung thông tin trong bộ đệm để chờ ACK

Hiệu suất của cơ chế ARQ Go-back-N

Tương tự như trường hợp ARQ dừng và đợi, khi tính hiệu suất của phương pháp phát lại ARQ Go-back-N, chúng ta cũng tính trong hai trường hợp: trường hợp lý tưởng và và trường hợp thực tế.

1) Trường hợp 1: trong điều kiện lý tưởng

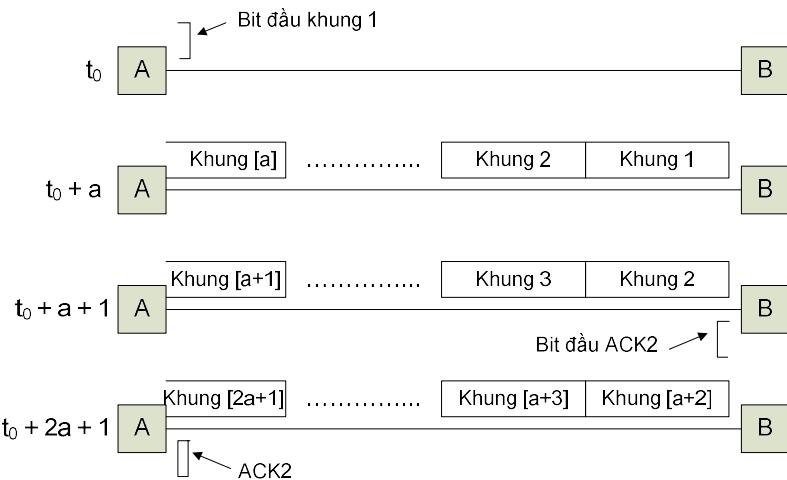
Để có thể tính được hiệu suất của phương pháp ARQ Go-back-N trong trường hợp lý tưởng, chúng ta dựa trên hiệu suất của phương pháp dừng và đợi đã biết. Đó là: $\eta = \frac{1}{1 + 2a}$ trong đó $a = \frac{T_D}{T_F}$. Nếu chuẩn

hóa đơn vị $T_F = 1$ đơn vị thời gian (giả thiết thời gian phát khung là 1 đơn vị chuẩn) thì trễ truyền sóng từ giữa hai trạm thu phát là a đơn vị thời gian. Nói một cách khác, khung thông tin truyền từ phát sang thu và khung ACK/NAK truyền từ thu về phát mất một khoảng thời gian là a đơn vị thời gian.

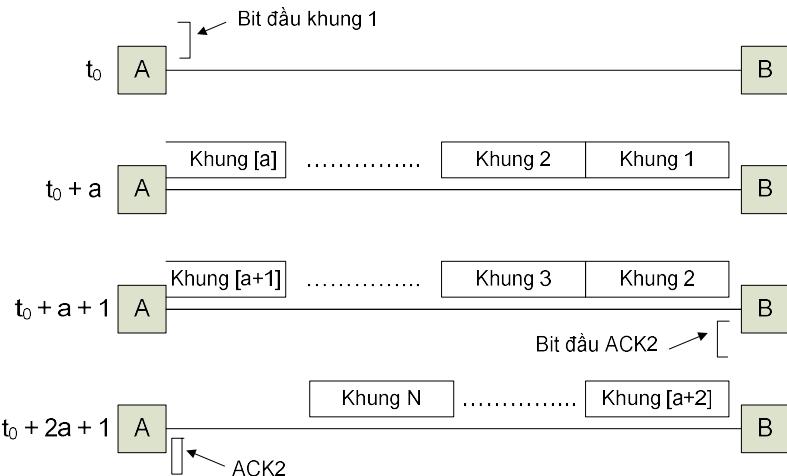
Hình 1-9 trình bày về giản đồ thời gian của phương pháp cửa sổ trượt. Hình 1-9(a) minh họa trong trường hợp kích thước cửa sổ $W > 2a + 1$ và hình 1-9 (b) minh họa trong trường hợp kích thước cửa sổ $W < 2a + 1$.

Quy ước:

- [X] là số nguyên nhỏ nhất lớn hơn hay bằng X .
- A là phía phát, B là phía thu



Hình 5-9(a): Giản đồ thời gian phương pháp cửa sổ trượt, $W > 2a+1$



Hình 5-9(b): Giản đồ thời gian phương pháp cửa sổ trượt, $W < 2a+1$

Hiệu suất của phương pháp này phụ thuộc vào kích thước cửa sổ W và giá trị a . Trên hình 1-9(a) và 1-9(b), phía phát A thực hiện truyền các khung tại thời điểm t_0 (bit đầu tiên của khung đầu tiên). Bit đầu tiên này đến phía thu B tại thời điểm t_0+a . Toàn bộ khung đầu tiên đến B tại thời điểm t_0+a+1 . Giả thiết bỏ qua thời gian xử lý, như vậy B cũng có thể gửi báo nhận ACK tại thời điểm t_0+a+1 . Trong trường hợp kích thước báo nhận nhỏ thì đây cũng là thời điểm toàn bộ báo nhận ACK rời khỏi phía thu. Báo nhận này đến phía phát A tại thời điểm t_0+2a+1 . Giả thiết phía phát luôn có dữ liệu để có thể truyền liên tục, khi ấy có hai trường hợp xảy ra.

- Nếu $W \geq 2a+1$: báo nhận đầu tiên đến phía phát trước khi $W=0$. Kể từ thời điểm A nhận được báo nhận đầu tiên, cứ mỗi một đơn vị thời gian A phát được một khung thông tin và cũng đồng thời nhận được một báo nhận, như vậy A có thể phát tin liên tục

- Nếu $W < 2a+1$: kích thước cửa sổ phía phát $W = 0$ đạt tại thời điểm t_0+W (xảy ra trước thời điểm t_0+2a+1) và phía phát không thể phát khung trong khoảng thời gian từ t_0+W đến t_0+2a+1 .

Hiệu suất của phương pháp cửa sổ trượt lúc này:

$$\eta_{window} = \frac{W}{2a+1} \text{ khi } W < 2a+1 \text{ và } \eta_{window} = 1 \text{ khi } W \geq 2a+1$$

2) Trường hợp 2: trong trường hợp thực tế, do có lỗi xảy ra nên hiệu suất thực tế nhỏ hơn hiệu suất trong trường hợp lý tưởng

$$\eta_{Go-back-N} = \frac{\eta_{window}}{N_R} \text{ trong đó } N_R \text{ là số lần truyền thành công}$$

Với trường hợp Go-back-N, mỗi khi có lỗi xảy ra, phía phát sẽ phải phát lại K khung (việc xác định K sẽ được tính ở phần sau).

Xác suất để khung thông tin được truyền đến lần thứ i thì đúng

$p(i) = p^{i-1} \cdot (1-p)$ (trong đó p^{i-1} là xác suất để $i-1$ lần truyền đầu tiên bị sai) và $1-p$ là xác suất để lần truyền thứ i đúng.

Với trường hợp này, tổng số khung phải truyền lại sẽ là $f(i) = 1 + (i-1)K$ trong đó $(i-1)K$ là tổng số khung phải truyền lại cho $i-1$ lần truyền sai.

Vậy số khung trung bình cần truyền trong trường hợp truyền đến lần thứ i mới đúng là $N(i) = f(i) \cdot p(i)$

Số khung trung bình cần truyền cho đến khi thành công:

$$N_R = \sum_{i=1}^{\infty} f(i) \cdot p^{i-1} (1-p) = \sum_{i=1}^{\infty} [(1-K) + Ki] p^{i-1} (1-p)$$

$$N_R = (1-K) \sum_{i=1}^{\infty} p^{i-1} (1-p) + K \sum_{i=1}^{\infty} i p^{i-1} (1-p)$$

Sử dụng các kết quả sau:

$$\sum_{i=0}^{\infty} r^i = \sum_{i=1}^{\infty} r^{i-1} = \frac{1}{1-r}$$

Và:

$$\sum_{i=1}^{\infty} i \cdot r^{i-1} = \frac{1}{(1-r)^2}$$

Ta có:

$$N_R = 1 - K + \frac{K}{1-p} = \frac{1-p+Kp}{1-p}$$

Tính K :

Để tính hiệu suất của phương pháp Go-back-N, ta giả thiết phía phát luôn có dữ liệu để phát (thực hiện phát liên tục, trừ khi phải dừng lại do kích thước cửa sổ = 0). Như vậy,

- Nếu $W \geq 2a + 1$ thì $K \approx 2a + 1$ – do khi NAK của khung i về thì phía phát đã phát thêm được $\approx 2a + 1$ khung
- Nếu $W < 2a + 1$ thì $K = W$ – do khi NAK của khung i về thì phía phát đã phát xong kích thước cửa sổ (W khung) và đang chờ báo nhận cho khung i để phát tiếp.

Hiệu suất:

$$\eta_{Go-back-N} = \frac{1-p}{1+2ap} \text{ khi } W \geq 2a+1$$

Và:

$$\eta_{Go-back-N} = \frac{W(1-p)}{(2a+1)(1-p+Wp)} \text{ khi } W < 2a+1$$

Nhận xét

Ưu điểm của phương pháp ARQ Go-back-N là hiệu suất cao hơn so với phương pháp ARQ dừng và đợi. Bên cạnh đó, cơ chế xử lý thông tin ở phía thu khá đơn giản và không cần bộ đệm.

Tuy nhiên, phương pháp này có nhược điểm là cần truyền lại quá nhiều khung thông tin trong trường hợp khung thông tin bị lỗi. Để khắc phục nhược điểm này, người ta đề xuất sử dụng cơ chế ARQ phát lại theo yêu cầu (Selective repeat ARQ)

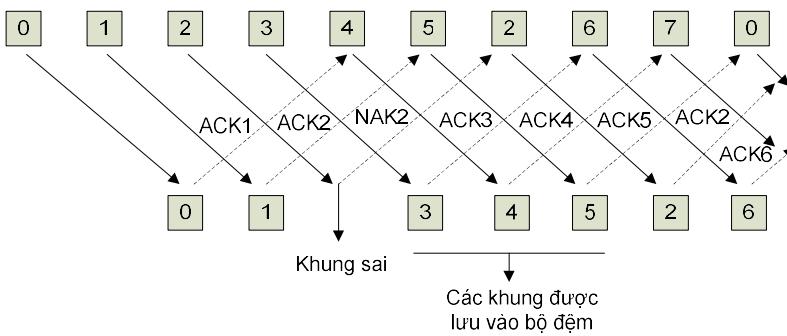
6.2.7. Selective repeat ARQ

Cơ chế hoạt động

Tương tự như cơ chế phát lại Go-back-N, cơ chế phát lại có lựa chọn (selective repeat ARQ) cũng dựa trên phương pháp cửa sổ trượt. Phía phát được phép phát tối đa W khung thông tin (kích thước cửa sổ) trước khi nhận được báo nhận.

Điểm khác biệt giữa selective repeat và Go-back-N nằm ở cách hai phương thức này xử lý khung thông tin bị lỗi. Với trường hợp selective repeat, phía phát sẽ chỉ thực hiện phát lại khung thông tin bị lỗi mà không cần phát lại tất cả các khung khác sau khung lỗi nếu như các khung đó không bị sai. Cơ chế này giúp tăng hiệu quả sử dụng đường truyền so với cơ chế Go-back-N.

Hình 5-10 mô tả nguyên tắc hoạt động của selective repeat



Hình 1-10: Nguyên tắc hoạt động của selective repeat

Một số chú ý của selective repeat ARQ

Do phía phát chỉ thực hiện phát lại các khung bị lỗi, do đó các khung đến phía thu có thể không theo thứ tự như khi được phát đi ở phía phát

- Phía thu phải có khả năng xử lý các khung thông tin không theo thứ tự.
- Do các khung thông tin phải được đưa lên lớp trên theo đúng thứ tự nên phía thu phải có bộ đệm để lưu tạm các khung thông tin trong khi chờ các khung bị mất hoặc lỗi được phát lại.

Phía phát phải thực hiện báo nhận cho tất cả các khung thông tin mà nó nhận đúng. Các khung thông tin không được báo nhận trong khoảng thời gian time-out tương ứng sẽ được coi là bị mất hoặc lỗi

Trong trường hợp phía thu nhận được một khung thông tin sai, phía thu có thể gửi NAK để báo lỗi và yêu cầu truyền lại khung đó (selective reject)

Hiệu suất của cơ chế selective repeat ARQ

Tương tự như trường hợp Go-back-N, hiệu suất của cơ chế selective repeat cũng được tính cho hai trường hợp: lý tưởng và không lý tưởng

1) Trường hợp 1: lý tưởng.

Đo bản chất của selective repeat là cũng hoạt động dựa trên phương pháp cửa sổ trượt (giống Go-back-N) nên trong trường hợp lý tưởng (không có lỗi), hiệu suất của selective repeat cũng chính là hiệu suất của Go-back-N và là hiệu suất của phương pháp cửa sổ trượt khi môi trường không có lỗi.

Hiệu suất:

$$\eta_{window} = \frac{W}{2a+1} \text{ khi } W < 2a+1$$

và

$$\eta_{window} = 1 \text{ khi } W \geq 2a+1$$

2) Trường hợp 2: không lý tưởng

Trong trường hợp này, hiệu suất của phương pháp selective repeat sẽ bằng hiệu suất của phương pháp cửa sổ trượt trong trường hợp lý tưởng chia cho số lần phát lại trung bình N_R (tương tự như trường hợp Go-back-N). Hiệu suất $\eta_{selective-repeat} = \frac{\eta_{window}}{N_R}$. Tuy nhiên N_R trong trường hợp selective repeat khác với trường hợp Go-back-N.

Tính N_R – do bản chất của việc truyền lại trong selective repeat hoàn toàn tương tự như trong phương pháp dừng và đợi nên với cách tính tương tự, $N_R = \frac{1}{1-p}$.

Hiệu suất:

$$\eta_{selective-repeat} = \frac{W(1-p)}{2a+1} \text{ khi } W < 2a+1$$

và

$$\eta_{selective-repeat} = 1-p \text{ khi } W \geq 2a+1$$

Nhận xét

Cơ chế selective repeat cho hiệu suất hoạt động trên đường truyền cao hơn so với Go-back-N do cơ chế này sử dụng đường truyền hiệu quả hơn. Tuy nhiên, cơ chế selective repeat hoạt động phức tạp hơn do nó yêu cầu phía thu phải có khả năng xử lý các khung thông tin đến phía thu không theo thứ tự. Ngoài ra, phía thu cần phải có bộ đệm để có thể lưu tạm thời các khung thông tin này.

6.3. Điều khiển luồng và tránh tắc nghẽn theo phương pháp cửa sổ

Cơ chế điều khiển luồng và chống tắc nghẽn dựa trên phương pháp cửa sổ được thực hiện bởi việc giới hạn số lượng gói tin được truyền ở phía phát nhằm đảm bảo thông tin này không vượt quá khả năng xử lý của phía thu.

Theo cơ chế này, phía phát sẽ không thực hiện phát tin chừng nào phía thu còn chưa xử lý xong gói tin (hoặc một số gói tin) trước đó. Khi phía thu xử lý xong thông tin do phía phát gửi đến thì nó sẽ báo cho phía phát biết và lúc này, phía phát sẽ tiếp tục gửi các gói tin tiếp theo. Cơ chế này đảm bảo việc truyền tin không bao giờ vượt quá khả năng xử lý của phía thu.

Với việc kết hợp hoạt động nhịp nhàng giữa phía phát và phía thu (có sử dụng báo nhận), số lượng gói tin đồng thời tồn tại trên đường truyền nằm trong giới hạn nhất định. Nếu phía thu có bộ đệm với dung lượng lớn hơn tổng kích thước các gói tin này thì bộ đệm phía thu sẽ không bao giờ bị tràn.

Các tiến trình thông tin có thể chịu sự ảnh hưởng của điều khiển luồng gồm có các kênh ảo độc lập, một nhóm các kênh ảo hay toàn bộ luồng thông tin từ một nút mạng này đến một nút mạng khác.

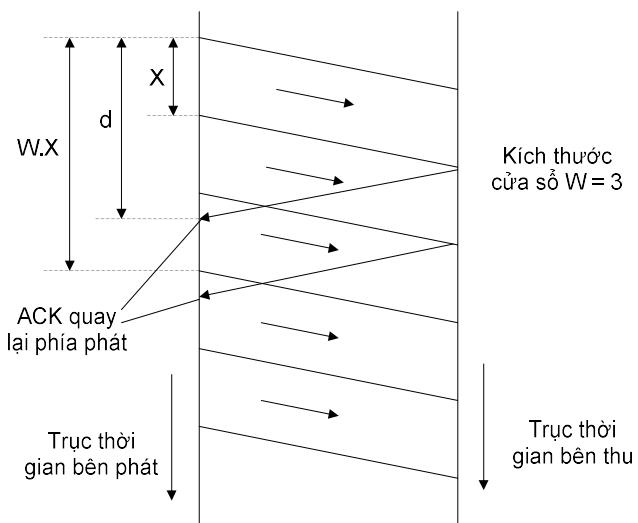
Các phương pháp điều khiển luồng được dựa trên các phương pháp điều khiển lỗi và phát lại ARQ ở lớp 2 của mô hình OSI (đã được trình bày ở phần trên).

6.3.1. Điều khiển luồng theo cửa sổ (Window Flow Control)

Phương pháp điều khiển luồng theo cửa sổ trượt là phương pháp được sử dụng phổ biến nhất ở thời điểm hiện tại. Trong phần này, chúng tôi sẽ lần lượt trình bày việc điều khiển luồng theo cửa sổ trượt theo hai cơ chế end-to-end (điều khiển luồng giữa điểm phát và điểm thu trong mạng) và hop-by-hop (điều khiển luồng giữa hai nút mạng liên tiếp).

Cửa sổ End-to-End

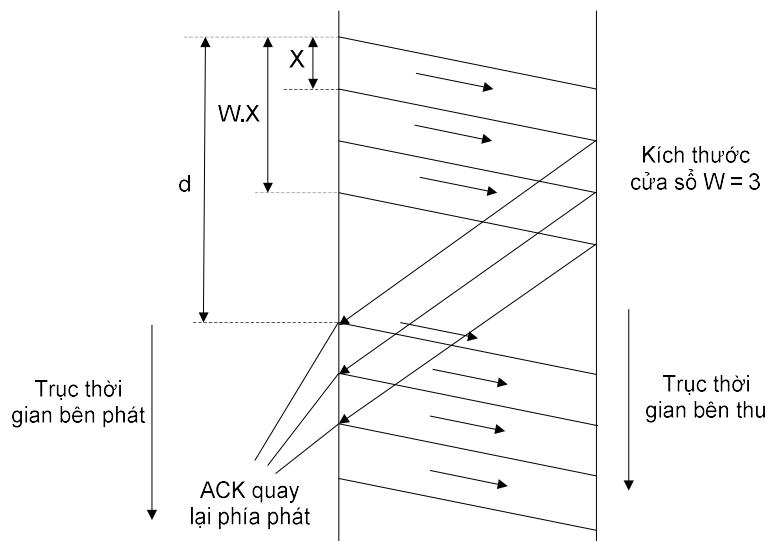
Như phần trên đã nói, phương pháp điều khiển luồng theo cửa sổ dựa trên cơ sở phương pháp cửa sổ trượt ARQ làm việc tại lớp liên kết dữ liệu. Các khung thông tin từ phát sang thu và khung báo nhận, báo lỗi truyền từ thu sang phát được đánh số thứ tự để phân biệt, kích thước cửa sổ $W < 2^k$ với k là số bit dùng đánh số phân biệt các khung.



Hình 5-11: Ví dụ phía phát truyền tin liên tục khi $W = 3$

Hình 5-11 trình bày mối liên hệ giữa kích thước cửa sổ và tốc độ truyền thông tin. Gọi X là thời gian phát một khung thông tin, W là kích thước cửa sổ và d là tổng trễ từ phát đến thu (dùng cho khung thông tin) và từ thu đến phát (dùng cho báo nhận), round-trip delay.

Trong hình vẽ này, kích thước cửa sổ $W = 3$, $d \leq W.X$. Như lý luận trong phần ARQ, lúc này phía phát có thể truyền thông tin liên tục mà không cần phải dừng lại đợi. Tốc độ phát thông tin $r = 1/X$ và trong trường hợp này, điều khiển luồng không có ý nghĩa (vì phía phát có thể phát tin với tốc độ cao nhất mà không bị hạn chế)

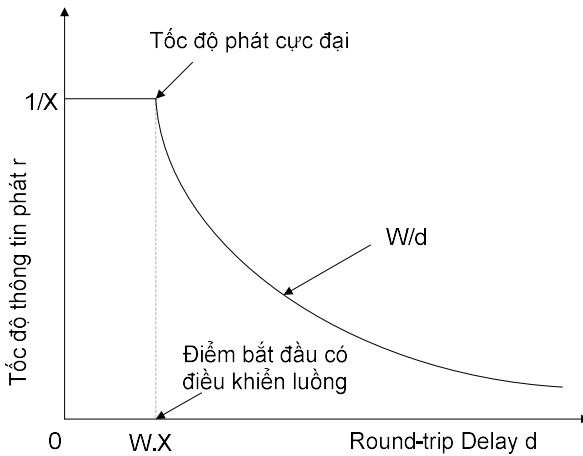


Hình 5-12: Ví dụ phía phát truyền tin không liên tục khi $W = 3$

Hình 1-12 trình bày trường hợp $d > W.X$, trong trường hợp này, ta thấy được vai trò của điều khiển luồng. Phía phát thực hiện phát W khung thông tin sau đó dừng lại chờ báo nhận ở phía thu, rồi mới được phát tiếp. Nói một cách khác, lượng thông tin đến phía thu (hay lượng thông tin đi vào mạng) đã bị hạn chế nhỏ hơn khả năng phát cực đại của phía phát. Điều này xảy ra khi round-trip delay lớn nên khi phía phát thực hiện phát xong W gói tin rồi nhưng báo nhận đầu tiên vẫn chưa quay trở lại. Lúc này phía phát phải ngừng phát và chờ báo nhận vì W đã giảm xuống 0 (xem lại phần nguyên tắc hoạt động của cửa sổ trượt). Nếu phía phát luôn có thông tin để phát thì tốc độ phát tin trung bình sẽ là W/d gói/s

Kết hợp cả hai trường hợp hình 5-11 và 5-12, ta tính được tốc độ phát tin cực đại khi kể đến round-trip delay sẽ là $r = \left\{ \frac{1}{X}, \frac{W}{d} \right\}$

- Khi d tăng (có tắc nghẽn), điều khiển luồng sẽ thực hiện vai trò của nó và giới hạn tốc độ truyền tin
- Khi không có tắc nghẽn xảy ra, d giảm và r tăng lên



Hình 5-13: Quan hệ giữa tốc độ truyền dẫn và round-trip delay trong điều khiển luồng

Hình 5-13 trình bày quan hệ của tốc độ truyền dẫn và round-trip delay trong cơ chế điều khiển luồng. Tốc độ truyền tin sẽ bị giảm khi xảy ra tắc nghẽn (trễ tăng). Ngoài ra, cơ chế cửa sổ phản ứng khá nhanh với tắc nghẽn (trong khoảng thời gian truyền W gói). Sự phản ứng nhanh với tắc nghẽn kết hợp với thông tin điều khiển ít là ưu điểm chính của cơ chế cửa sổ so với các cơ chế khác.

Nguyên tắc chọn kích thước cửa sổ:

- Trong trường hợp không có tắc nghẽn xảy ra, kích thước cửa sổ được chọn đủ lớn để đảm bảo tốc độ truyền thông tin đạt $r = 1/X$ gói/s.

Quy ước:

- d' = round-trip delay khi trễ hàng đợi xấp xỉ 0 (không có tắc nghẽn) – đây là trễ tính từ lúc phát gói thông tin ở bên phát và nhận ACK từ phía thu
- N = số nút mạng dọc theo đường truyền từ phát đến thu
- D = trễ truyền sóng dọc theo đường truyền

$$d' = 2.N.X + 2.D$$

Để đảm bảo tốc độ truyền tin tối đa (khi không có tắc nghẽn), cần đảm bảo $W.X \geq d'$ hay $W \geq 2.N + 2.D/X$. Ta nhận thấy:

- Khi $D < X$ thì $W \approx 2.N$ – kích thước cửa sổ không phụ thuộc vào trễ truyền sóng.
- Khi $D \gg X$ thì $W \approx 2.D/X$ – kích thước cửa sổ không phụ thuộc vào chiều dài đường đi.

- Trong trường hợp có tắc nghẽn xảy ra, thì trễ round-trip $d > d'$ (bao gồm cả trễ hàng đợi do tắc nghẽn)

Phương pháp cửa sổ End-to-End có những hạn chế nhất định. Đó là:

- Khó đảm bảo trễ nằm trong giới hạn cho phép khi lưu lượng vào mạng tăng

Giả sử trong mạng có n tiến trình điều khiển luồng với kích thước cửa sổ tương ứng là W_1, W_2, \dots, W_n . Lúc này, tổng số gói tin trong mạng sẽ là $\sum_{i=1}^n \beta_i \cdot W_i$ trong đó β_i là một hệ số trong khoảng 0 đến 1 phụ thuộc vào thời gian trễ của ACK. Theo định luật Little's thì trễ trung bình của

$$\text{gói tin trong mạng sẽ là } T = \frac{\sum_{i=1}^n \beta_i \cdot W_i}{\lambda} \text{ trong đó } \lambda \text{ là thông lượng.}$$

Khi số lượng các tiến trình cần điều khiển luồng tăng lên (n tăng) thì λ tiến đến giá trị cực đại là tốc độ của các đường liên kết và do đó, là giá trị không đổi (giá trị này phụ thuộc vào mạng, vị trí của điểm phát và thu cũng như giải thuật định tuyến). Như vậy giá trị trễ T sẽ tăng tỷ lệ với số lượng tiến trình được điều khiển luồng (chính xác ra là kích thước cửa sổ của chúng). Như vậy, nếu số lượng tiến trình là rất lớn thì hệ thống mạng không đảm bảo giữ giá trị T nằm trong một giới hạn nhất định và không có khả năng tránh tắc nghẽn một cách triệt để.

Một giải pháp có thể sử dụng là giảm kích thước cửa sổ để có thể giảm trễ khi mạng hoạt động ở tình trạng nặng tải (có thể xảy ra tắc nghẽn). Giải pháp này có thể áp dụng ở một mức độ nào đó tuy nhiên nó nếu giá trị này quá nhỏ thì việc truyền thông tin lại không hiệu quả.

Trên thực tế, người ta sử dụng phương pháp cửa sổ thích ứng (adaptive window) để thực hiện truyền tin. Trong phương pháp này, kích thước cửa sổ có thể thay đổi tùy thuộc tình trạng của mạng. Trong trường hợp mạng ít tải, kích thước cửa sổ có thể lớn để cho phép truyền thông tin với tốc độ cao. Khi tải trên mạng tăng, kích thước cửa sổ được giảm đi nhằm tránh tắc nghẽn. Phương pháp cửa sổ thích ứng sẽ được trình bày trong phần sau.

- Khó đảm bảo tính công bằng cho tất cả người dùng.

Một hạn chế nữa của phương pháp cửa sổ end-to-end là chưa đảm bảo được tính công bằng cho người dùng trong tất cả các trường hợp. Như phần trên đã nói, để đảm bảo truyền tin tốt nhất cho một kết nối, kích thước cửa sổ tỷ lệ với số nút mạng trên đường đi từ nguồn đến đích cũng như tỷ lệ với trễ truyền sóng dọc theo đường truyền (cũng phụ thuộc vào khoảng cách). Như vậy, trong trường hợp có tắc nghẽn, nếu trên một đường truyền có nhiều kết nối cùng hoạt động thì kết nối nào có khoảng cách nguồn – đích lớn sẽ được sử dụng tài nguyên nhiều hơn (do kích thước cửa sổ lớn hơn và số lượng gói tin đến nút đó và được chấp nhận sẽ nhiều hơn).

Để đảm bảo được tính công bằng, người ta dùng cơ chế round-robin (xử lý vòng) cho tất cả các kết nối cùng sử dụng tài nguyên của một nút mạng. Lúc này, các kết nối được coi như có độ ưu tiên như nhau và được xử lý luân phiên dựa theo kết nối chứ không dựa trên tỷ lệ gói tin đến.

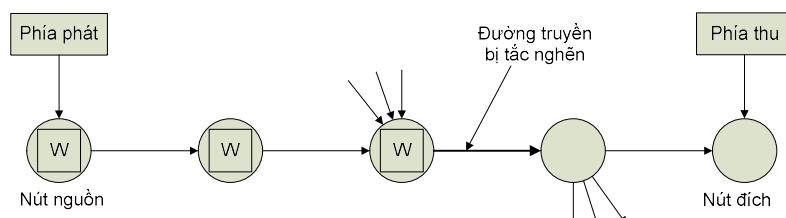
Cửa sổ Hop-by-Hop

Trong cơ chế điều khiển luồng hop-by-hop, việc điều khiển luồng được thực hiện giữa hai nút mạng kế tiếp trên đường truyền. Mỗi nút mạng có các cửa sổ độc lập dùng cho các kênh làm việc khác nhau (kênh ảo). Nguyên tắc hoạt động của cơ chế này tương tự như điều khiển luồng kiểu end-to-end nhưng chỉ áp dụng cho một chặng. Trong trường hợp truyền thông tin cự ly không quá xa (với đa phần các cơ chế truyền tin, trừ thông tin vệ tinh) kích thước cửa sổ thường là 2 hoặc 3 (do số nút mạng thông tin phải đi qua là 1, trễ truyền sóng không đáng kể).

Ta tạm gọi nút có thông tin cần truyền là nút nguồn, nút có nhận thông tin là nút đích (các nút đọc trên đường truyền, và có thể bao gồm cả phía phát và phía thu). Mục đích chính của điều khiển luồng hop-by-hop là đảm bảo bộ đệm của nút đích không bị quá tải bởi quá nhiều gói tin đến (như trong trường hợp end-to-end). Điều này được thực hiện với việc nút đích giảm tốc độ gửi ACK về cho nút nguồn. Trong trường hợp tổng quát, nút đích có bộ đệm với dung lượng W gói cho mỗi liên kết và nó sẽ gửi ACK cho nút nguồn nếu trong bộ đệm còn chỗ trống. Nút đích sẽ xóa gói tin trong bộ đệm nếu nó đã được truyền thành công đến nút kế tiếp trên đường truyền hay đã đi ra khỏi mạng.

Giả sử có ba nút liên tiếp trên mạng là $(i-1, i, i+1)$. Giả sử bộ đệm của i đã bị đầy với W gói tin. Nút i sẽ gửi ACK cho nút $i-1$ nếu nó đã gửi thành công một gói tin cho nút $i+1$ (lúc đó bộ đệm của nút i mới được giải phóng và có chỗ cho một gói tin). Nút i thực hiện điều này nếu nó nhận được một ACK từ nút $i+1$.

Trong trường hợp có tắc nghẽn xảy ra tại một nút nào đó, bộ đệm của nút này bị đầy bởi W gói tin và theo hệ quả, bộ đệm của các nút phía trước nút đó cũng sẽ dần dần bị đầy. Hiện tượng này được gọi là **backpressure** và được trình bày trên hình 1-14.



Hình 5-14: Cơ chế backpressure trong điều khiển luồng hop-by-hop

Ưu điểm của phương pháp hop-by-hop được trình bày trên hình 1-14. Trong trường hợp xấu nhất, giả sử tắc nghẽn xảy ra tại đường nối cuối cùng của tuyến truyền (đường nối thứ n) thì tổng số gói tin nằm trong mạng sẽ là $n.W$ (bộ đệm của mỗi nút sẽ bị điền đầy bởi W gói tin). Trong trường hợp này, số lượng gói tin sẽ được phân bổ đều ở bộ đệm của các nút và do đó dung lượng bộ đệm cần thiết ở mỗi nút sẽ nhỏ hơn trường hợp end-to-end rất nhiều (chú ý rằng trong trường hợp end-to-end, nếu tổng số gói tin vào mạng, hay kích thước cửa sổ, là $n.W$ thì dung lượng bộ đệm tương ứng ở mỗi nút cũng phải là $n.W$).

Một ưu điểm khác nữa của phương pháp hop-by-hop chính là cho phép thực hiện tính công bằng. Với việc phân các gói tin của một kết nối dọc theo các nút mạng mà kết nối phải đi qua, ta có thể tránh được tình trạng ở tại một nút, kết nối với khoảng cách nguồn – đích lớn sẽ chiếm hết tài nguyên của các kết nối khác. Trong trường hợp hop-by-hop, kích thước cửa sổ của các kết nối là xấp xỉ bằng nhau do đó tốc độ thông tin đến là không chênh lệch và việc sử dụng tài nguyên được đảm bảo công bằng. Điều này không đúng trong trường hợp kết nối giữa hai nút dùng cho truyền vệ tinh. Trong trường hợp này, do trễ truyền dẫn khá lớn nên kích thước cửa sổ của kết nối vệ tinh có thể lớn hơn kích thước cửa sổ của các kết nối khác dẫn đến tình trạng không công bằng.

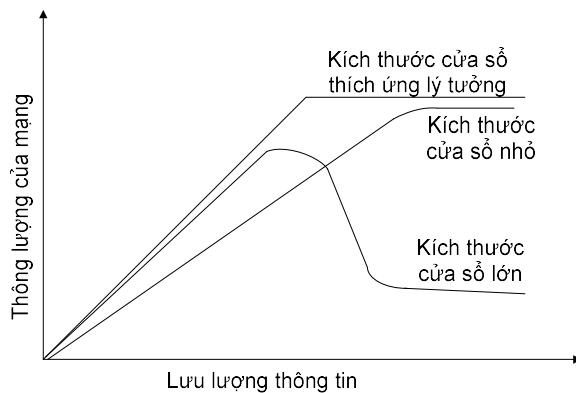
Phương thức Isarithmic

Phương thức này cũng được coi là một biến thể của cơ chế điều khiển luồng theo cửa sổ với một cửa sổ duy nhất được dùng cho toàn mạng. Việc điều khiển luồng được thực hiện bởi việc giới hạn số lượng gói tin đi vào mạng thông qua việc cấp phát một số lượng hạn chế thẻ bài. Mỗi một gói tin muốn đi vào mạng cần nhận được một thẻ bài ở nút mà gói tin đó vào và trả lại thẻ bài ở nút mà gói tin đó ra khỏi mạng. Như vậy, tổng số gói tin tồn tại đồng thời trong mạng luôn nhỏ hơn hoặc bằng tổng số lượng thẻ bài, và việc điều khiển luồng được thực hiện.

Tuy nhiên, phương pháp này có những hạn chế nhất định. Nó không đảm bảo tính công bằng cho tất cả người dùng vì không có những cơ chế nhất định để quản lý vị phân phối thẻ bài. Ngoài ra, các thẻ bài có thể bị mất vì những lý do nhất định mà hiện tại chưa có cơ chế để quản lý số lượng thẻ bài tồn tại trong mạng. Vì những lý do đó, phương thức Isarithmic ít được sử dụng trong thực tế.

6.3.2. Điều khiển tắc nghẽn sử dụng cửa sổ thích ứng (adaptive window)

Bên cạnh việc sử dụng cơ chế cửa sổ để thực hiện điều khiển luồng, người ta có thể sử dụng cơ chế cửa sổ để thực hiện điều khiển và tránh tắc nghẽn ở trong mạng. Khi mạng có khả năng mang thông tin của người dùng, kích thước cửa sổ sẽ được đặt ở một mức nào đó. Khi mạng năng tải và có tắc nghẽn xảy ra, phía phát sẽ giảm kích thước cửa sổ để giảm số lượng gói tin đi vào mạng, do đó, thực hiện chức năng điều khiển tắc nghẽn cho mạng. Kích thước cửa sổ chính là nhân tố quyết định tốc độ thông tin từ phía phát đi vào mạng.



Hình: Mối quan hệ giữa kích thước cửa sổ và lưu lượng mạng

Hình trên đây trình bày mối quan hệ giữa kích thước cửa sổ và thông lượng của mạng. Khi lưu lượng vào mạng nhỏ, kích thước cửa sổ lớn tỏ ra tối ưu do tận dụng được thời gian truyền gói tin, tuy nhiên, khi lưu lượng vào mạng tăng lên, việc sử dụng kích thước cửa sổ lớn sẽ gây ra tắc nghẽn do có quá nhiều gói tin có thể được gửi cùng lúc vào mạng. Trong trường hợp này, người ta sử dụng các cửa sổ có kích thước nhỏ để đáp ứng với tình trạng của mạng.

Việc thay đổi kích thước cửa sổ một cách mềm dẻo cho phù hợp với tình trạng lưu lượng của mạng chính là cách thức điều khiển tắc nghẽn của các thiết bị đầu cuối (phía phát và phía thu). Cơ chế thay đổi kích thước cửa sổ theo tình trạng lưu lượng mạng được gọi là cơ chế cửa sổ thích ứng (adaptive window).

Vấn đề của điều khiển tắc nghẽn theo phương pháp cửa sổ thích ứng là điều kiện quyết định việc tăng và giảm kích thước cửa sổ. Để có thể thực hiện được điều này, phía phát dựa trên các thông tin phản hồi từ phía thu hoặc các thiết bị trên đường truyền từ phát đến thu để thực hiện điều chỉnh kích thước cửa sổ.

Khi xét đến các thiết bị mạng trung gian giữa phát và thu (tạm gọi là thiết bị mạng), người ta chia làm hai loại:

- Thiết bị mạng thông minh (active intermediate system) – là các thiết bị mạng có khả năng phát hiện tắc nghẽn đang xảy ra hoặc có thể xảy ra và có khả năng thông báo cho phía phát
- Thiết bị mạng không thông minh(passive intermediate system) – các thiết bị này không có khả năng phát hiện tắc nghẽn, việc xác định tình trạng tắc nghẽn hoàn toàn được thực hiện bởi phía phát.

Trong các phần dưới đây, chúng tôi sẽ trình bày hoạt động của cơ chế cửa sổ thích ứng cho cả hai loại thiết bị mạng này

Thiết bị mạng thông minh

Kỹ thuật điều khiển tắc nghẽn sử dụng thiết bị mạng thông minh hoạt động như sau:

- Thiết bị mạng phát hiện tình trạng tắc nghẽn xảy ra hoặc sắp xảy ra (ví dụ: dung lượng bộ đệm vượt quá một ngưỡng nào đó)
- Khi phát hiện tắc nghẽn, thiết bị mạng thông báo cho tất cả các nút nguồn (phía phát) thực hiện phát thông tin qua thiết bị mạng này
- Các nút nguồn thực hiện giảm kích thước cửa sổ để giảm tắc nghẽn (với việc giảm kích thước cửa sổ, phía phát giảm số lượng gói tin có thể đi vào mạng)
- Các nút nguồn có thể tăng kích thước cửa sổ nếu chúng xác định được rằng tình trạng tắc nghẽn đã được giải quyết.

Chú ý rằng, khái niệm kích thước cửa sổ ở đây là kích thước cửa sổ cực đại, hay số lượng gói tin có thể đồng thời được phát đi mà không cần báo nhận. Trên thực tế, kích thước cửa sổ hoạt động của nút nguồn luôn thay đổi (giảm nếu phía nguồn phát gói tin và tăng nếu nút nguồn nhận được báo nhận).

Các tham số có thể dùng để xác định tắc nghẽn tại nút mạng là dung lượng bộ đệm (còn trống nhiều hay ít), khả năng hoạt động của CPU (nhiều hay ít) hoặc mức độ sử dụng băng thông của đường truyền.

Để có thể cảnh báo cho phía phát, nút mạng có thể sử dụng một trong hai cơ chế:

- Sử dụng một gói tin cảnh báo độc lập – phương pháp này cho phép phía phát nhanh chóng nhận được thông tin tắc nghẽn và phản ứng kịp thời. Tuy nhiên, hạn chế của phương pháp này là phải sử dụng gói tin độc lập gây lãng phí băng thông và phức tạp hóa việc quản lý
- Sử dụng một bit chỉ thị tắc nghẽn nằm trong trường điều khiển của gói tin mang dữ liệu từ phía thu sang phía phát. Bit chỉ thị tắc nghẽn bằng 0 thể hiện tắc nghẽn không xảy ra và bit này bằng 1 khi tắc nghẽn xảy ra.

Phía phát sẽ dựa trên thông tin cảnh báo nào để quyết định việc tăng giảm kích thước cửa sổ.

Nếu việc thay đổi kích thước cửa sổ chỉ được dựa trên một gói tin phản hồi thì có thể xảy ra tình trạng hệ thống hoạt động không hiệu quả (nếu nút mạng gửi một gói tin cảnh báo tắc nghẽn rồi lại gửi một gói thông báo không tắc nghẽn). Vì vậy, trên thực tế, phía phát sẽ dựa trên một số lượng thông báo nhất định từ phía nút mạng rồi mới kết luận về tình trạng tắc nghẽn. Thông thường, với một số lượng thông báo nhận được, nếu số gói tin cảnh báo tắc nghẽn vượt quá một giới hạn nào đó thì phía phát sẽ coi là có tắc nghẽn xảy ra và giảm kích

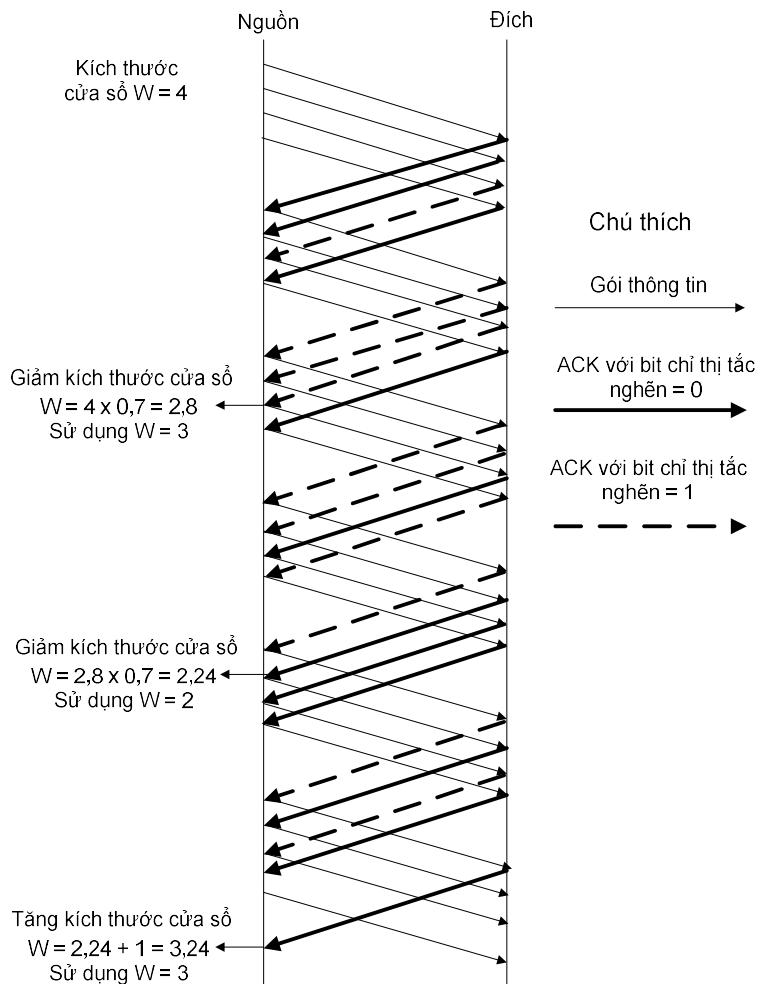
thước cửa sổ. Nếu số lượng cảnh báo này nhỏ hơn giới hạn cho phép thì phía phát sẽ coi là không có tắc nghẽn và tăng kích thước cửa sổ.

Việc tăng và giảm kích thước cửa sổ có thể tuân theo một trong hai quy tắc: phép cộng và phép nhân.

- Phép cộng: $W_{new} = W_{old} + I$ trong đó W_{new} và W_{old} là kích thước cửa sổ mới và cũ, I là hệ số tăng giảm. Khi $I > 0$ là tăng kích thước cửa sổ và $I < 0$ là giảm kích thước cửa sổ
- Phép nhân: $W_{new} = W_{old} \times \alpha$ với các quy ước tương tự như trên. Khi $\alpha > 1$ là tăng kích thước cửa sổ và $\alpha < 1$ là giảm kích thước cửa sổ. Trong trường hợp kích thước cửa sổ không phải số nguyên thì kích thước đó sẽ được quy về số nguyên gần nhất.

Trong ứng dụng cụ thể, người ta thường dùng phép cộng khi tăng và dùng phép nhân khi giảm.

Hình dưới đây trình bày nguyên tắc tăng giảm kích thước cửa sổ dựa trên bit chỉ thị tắc nghẽn được gửi đi từ nút mạng có tắc nghẽn. Trong ví dụ này, kích thước cửa sổ ban đầu là $W = 4$, việc kết luận về tình trạng tắc nghẽn được dựa trên các nhóm 7 báo nhận gửi về. Trong 7 báo nhận đó, nếu có lớn hơn hoặc bằng 4 báo nhận có bit chỉ thị tắc nghẽn bằng 1 thì nút nguồn coi là có tắc nghẽn và giảm kích thước cửa sổ, ngược lại nút nguồn coi là không có tắc nghẽn và tăng kích thước cửa sổ. Trong trường hợp này, việc giảm được thực hiện theo phép nhân với $\alpha = 0,7$ và việc tăng được thực hiện theo phép cộng với $I = 1$.



Hình: Sử dụng bit chỉ thi tắc nghẽn để thay đổi kích thước cửa sổ

Thiết bị mang không thông minh

Trong trường hợp này, các thiết bị mạng không có khả năng cảnh báo cho phía phát về tình trạng tắc nghẽn và việc xác định tắc nghẽn trong mạng hoàn toàn dựa trên việc suy đoán của nút nguồn. Thiết bị mạng không thông minh là các thiết bị mạng đơn giản, không có khả năng xác định trạng thái bộ đệm, trạng thái CPU hay trạng thái sử dụng đường truyền. Trong một số trường hợp khác, do yêu cầu hoạt động với tốc độ cao nên các thiết bị mạng có thể cũng không kiểm tra về trình trạng tắc nghẽn có thể xảy ra mỗi khi gói tin đi qua thiết bị.

Khi không có sự hỗ trợ của thiết bị mạng, nút nguồn kết luận về trạng thái tắc nghẽn hoàn toàn dựa trên báo nhận được gửi về. Trong trường hợp mạng bị tắc nghẽn, báo nhận có thể bị trễ lớn (trễ báo nhận hoặc trễ gói đến phía thu) hoặc có thể bị mất (mất báo nhận hoặc mất gói nên không có báo nhận). Trong trường hợp mất báo nhận

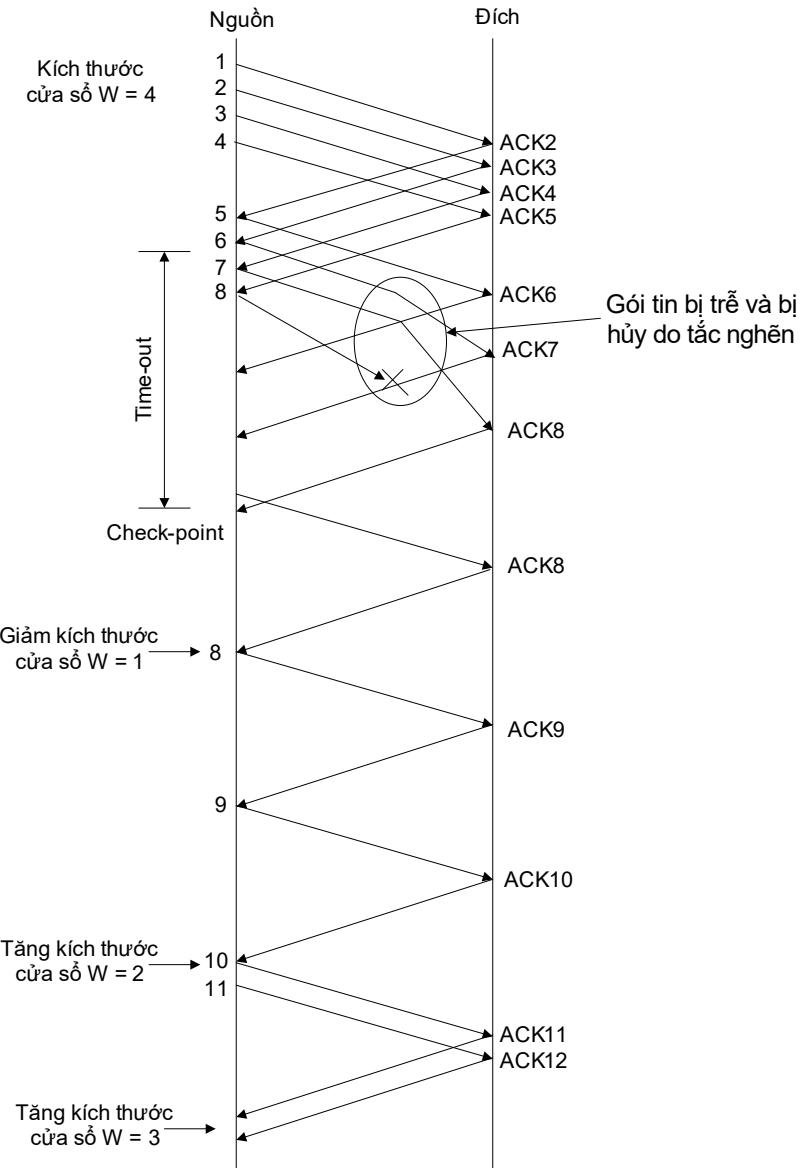
hoặc báo nhận đến quá trễ, nút nguồn sẽ phải phát lại gói và việc phát lại này có thể coi là một tín hiệu để kết luận về tình trạng tắc nghẽn.

Cơ chế tắc nghẽn này gọi là cơ chế điều khiển tắc nghẽn dùng cửa sổ thích ứng dựa trên time-out và hoạt động như sau:

- Tại thời điểm ban đầu, nút nguồn đặt kích thước cửa sổ bằng W_{max}
- Mỗi khi có time-out xảy ra và phía phát phải thực hiện phát lại gói tin thì nút nguồn sẽ đặt $W = 1$
- Mỗi khi nhận được n báo nhận từ nút đích, phía phát lại tăng kích thước cửa sổ lên 1. Kích thước cửa sổ sẽ không bao giờ vượt quá kích thước cửa sổ W_{max} . Với việc thay đổi giá trị n , người ta có thể thực hiện điều khiển tắc nghẽn ở nhiều mức độ khác nhau.

Trong trường hợp này, chúng ta giả thiết tỷ lệ lỗi bit là khá nhỏ và time-out xảy ra hoàn toàn là do trễ chứ không phải do mất gói vì lỗi bit.

Ví dụ trên hình dưới đây minh họa cơ chế điều khiển tắc nghẽn theo cửa sổ thích ứng dựa trên time-out. Trong ví dụ này, kích thước cửa sổ ban đầu $W_{max} = 4$, và giá trị $n = 2$. Giả thiết rằng các nút mạng trung gian có thể gây ra trễ hoặc hủy gói tin hoặc báo nhận nếu tắc nghẽn xảy ra. Điều này dẫn đến hệ quả là có time-out xảy ra tại nút nguồn cho các gói tin đó.



Hình: Sử dụng time-out và ACK để tăng/giảm kích thước cửa sổ

6.4. Điều khiển luồng và chống tắc nghẽn dựa trên băng thông (rate-based flow control)

6.4.1. Khái niệm

Trong phần trên, chúng ta đã thấy hạn chế cơ bản của điều khiển luồng theo phương pháp cửa sổ là trễ gói sẽ tăng tỷ lệ với số lượng kết nối cần thực hiện điều khiển luồng. Mặc dù có thể giảm kích thước

cửa sổ để có thể giảm trễ gói tuy nhiên phương pháp này không dễ thực hiện.

Để có thể đáp ứng được yêu cầu của điều khiển luồng, người ta để xuất các phương pháp thực hiện điều khiển luồng và chống tắc nghẽn dựa trên việc hạn chế băng thông. Cơ chế kiểm soát băng thông đảm bảo lượng thông tin của người dùng đưa vào mạng không vượt quá một mức nào đó nhằm tránh tắc nghẽn trong mạng. Trong một số trường hợp cụ thể, thông tin của người dùng đưa vào mạng có thể vượt quá lượng thông tin giới hạn ở một mức độ nào đó cho phép.

Cơ chế kiểm soát băng thông của thông tin đi vào mạng chia làm hai loại:

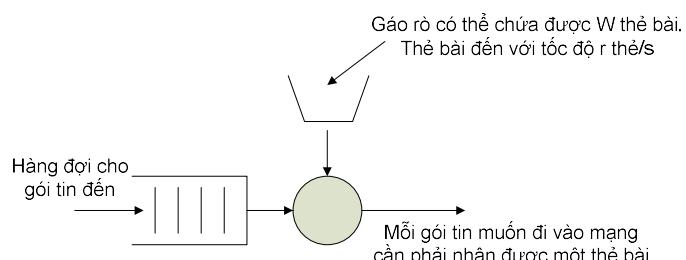
- **Kiểm soát chặt** (strict implementation) – với tốc độ thông tin vào mạng trung bình là r gói/s, thì hệ thống kiểm soát sẽ chỉ cho một gói vào cứ sau mỗi $1/r$ giây. Phương pháp này không phù hợp cho các thông tin có thay đổi với biên độ lớn (bursty traffic). Ví dụ điển hình của phương pháp này là cơ chế TDMA.
- **Kiểm soát lỏng** (less-strict implementation) – với tốc độ thông tin vào mạng trung bình là r gói/s thì hệ thống kiểm soát sẽ cho W gói vào mạng trong khoảng thời gian W/r giây. Trong phương pháp này, tốc độ dữ liệu trung bình là không đổi nhưng cho hệ thống cho phép nhận tối đa W gói tại một thời điểm (bursty traffic). Cơ chế này thường được triển khai với việc sử dụng gáo rò (leaky bucket)

Trong phần dưới đây, chúng tôi sẽ trình bày nguyên tắc hoạt động của gáo rò.

6.4.2. Điều khiển băng thông theo thuật toán gáo rò (leaky bucket)

Nguyên tắc hoạt động của leaky bucket

Hình 5-15 dưới đây minh họa mô hình gáo rò



Hình 5-15: Mô hình gáo rò

Trong mô hình này, nút mạng được trang bị một gáo rò dùng kiểm soát lưu lượng thông tin đi vào mạng. Gáo là một bộ đệm có khả năng lưu trữ tối đa là W thẻ bài. Các thẻ bài được điền vào gáo với tốc độ r thẻ bài/s. Khi gáo đã đầy thẻ bài thì thẻ bài sẽ không được điền thêm vào gáo.

Mỗi khi một gói tin đến và để có thể được vào được mạng thì gói tin đó phải nhận được một thẻ bài. Tốc độ trung bình của thông tin vào mạng là r gói tin/s và bằng tốc độ điền thẻ bài vào gáo.

Trong trường hợp gáo rò đầy thẻ bài, nút mạng có thể cho tối đa W gói tin vào mạng tại một thời điểm (burst size). Nếu W nhỏ thì khả năng kiểm soát tốc độ luồng thông tin vào là tốt, nếu W lớn thì khả năng hỗ trợ burst tốt.

Với việc sử dụng gáo rò, luồng thông tin vào mạng có tốc độ không vượt quá r gói/s. Nếu mạng có nhiều nút mạng để giao tiếp với bên ngoài (entry point), mỗi nút mạng được trang bị một gáo rò để kiểm soát lưu lượng thông tin vào mạng thì cho dù tốc độ thông tin của đến các nút có thể thay đổi, nhưng tốc độ thông tin trong mạng khá ổn định. Với đặc điểm này, người ta nói gáo rò thực hiện chức năng định dạng lưu lượng.

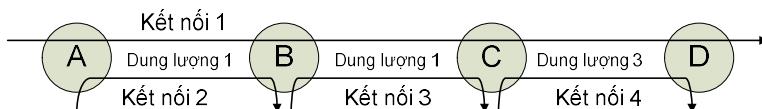
Tính toán hiệu năng của leaky bucket (pending)

- Trễ trung bình của gói khi đi qua leaky bucket
- Độ dài hàng đợi gói trung bình

Chọn các tham số của leaky bucket (pending)

Mô hình công bằng cực đại – cực tiểu (max-min fairness)

Một trong những vấn đề khó khăn nhất của thực hiện điều khiển luồng và kiểm soát tắc nghẽn là đảm bảo tính công bằng cho các kết nối hoặc người dùng khi xảy ra tắc nghẽn. Khái niệm tính công bằng thể hiện ở chỗ các kết nối, người dùng được sử dụng tài nguyên mạng với cơ hội như nhau. Để có thể hiểu rõ hơn về tính công bằng, xét mô hình mạng trên hình vẽ 5-16 dưới đây



Hình 5-16: Tính công bằng

Trên hình 1-16, đường nối $A - B$ và $B - C$ có dung lượng 1 và đường nối $C - D$ có dung lượng 3. Kết nối 1 đi qua tất cả các nút A, B, C, D ; kết nối 2 đi qua A, B ; kết nối 3 đi qua B, C ; kết nối 4 đi qua C, D .

Ta thấy, có tốc độ của các kết nối 1, 2 và 3 đều là $1/2$ để đảm bảo các kết nối này sử dụng băng thông trên các đường $A - B$ và $B - C$ là công bằng. Tuy nhiên, trên đường liên kết $C - D$, mặc dù nó được chia sẻ bởi kết nối 1 và kết nối 4, tuy nhiên băng thông của kết nối 4 có thể đạt đến $5/2$ vì kết nối 1 chỉ sử dụng hết $1/2$ mà thôi.

Như vậy, tính công bằng không chỉ đơn thuần là chia sẻ băng thông bình đẳng cho các kết nối/người dùng trên tất cả các phân vùng trong mạng mà nó được hiểu và sử dụng mềm dẻo trong từng trường hợp cụ thể.

Việc sử dụng tài nguyên mạng hiệu quả nhất có thể trong khi vẫn có thể đảm bảo được tính công bằng cho các kết nối được thực hiện bởi cơ chế điều khiển luồng cực đại – cực tiểu (max-min flow control). Cơ chế này được xây dựng trên mô hình công bằng cực đại – cực tiểu (max-min fairness).

Nguyên tắc hoạt động cơ bản của cơ chế điều khiển luồng cực đại – cực tiểu như sau:

Nguyên tắc – Sau khi người dùng với yêu cầu ít nhất về tài nguyên đã được đáp ứng công bằng, các tài nguyên còn lại được tiếp tục phân chia (một cách công bằng) cho những người dùng còn lại. Trong nhóm người dùng này, tài nguyên lại được phân chia sao cho người dùng có yêu cầu ít nhất được đáp ứng, và quá trình cứ tiếp tục đến hết. Nói một cách khác, việc cấp phát tài nguyên mạng cho một người dùng i không được làm ảnh hưởng đến tài nguyên đã cấp các người dùng khác với yêu cầu ít hơn i.

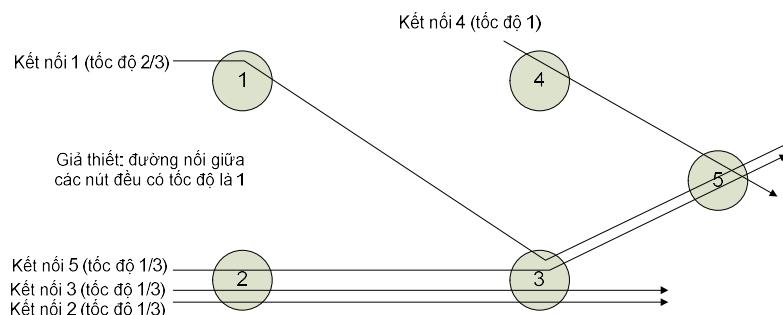
Một số quy ước và định nghĩa:

- Giả thiết mạng là một đồ thị có hướng $G = (N, A)$ trong đó N là tập hợp các nút và A là tập hợp các đường liên kết giữa các nút
- P là tập hợp các kết nối hiện sử dụng trong mạng, một kết nối bất kỳ trong tập hợp các kết nối được ký hiệu là p .
- r_p là tốc độ (hay băng thông) dùng cho kết nối p .

Với một đường liên kết a bất kỳ ($a \in A$) thì lưu lượng thông tin trên liên kết a là $F_a = \sum_{p \in P} \delta_p(a) \cdot r_p$ trong đó $\delta_p(a) = 1$ nếu kết nối p đi qua liên kết a và bằng 0 trong trường hợp ngược lại. Gọi C_a là dung lượng của liên kết a , khi ấy ta có: $r_p \geq 0$ với $\forall p \in P$ và $F_a \leq C_a$ với $\forall a \in A$ (*)

Mục đích của cơ chế công bằng cực đại – cực tiểu là tìm được tập hợp các giá trị r_p (với $\forall p \in P$) thỏa mãn (*) đồng thời thỏa mãn nguyên tắc của quy chế công bằng cực đại – cực tiểu. Tập hợp các giá trị r_p tạo thành vector công bằng cực đại – cực tiểu, ký hiệu là r .

Một đặc điểm quan trọng của vector công bằng cực đại – cực tiểu là với mỗi một kết nối p bất kỳ thuộc P , có ít nhất một liên kết a mà p đi qua sao cho $F_a = C_a$ và r_p không nhỏ hơn tốc độ của bất kỳ kết nối nào trên liên kết đó. Liên kết đó gọi là điểm nghẽn của p (bottleneck arc). Hình 1-17 minh họa khái niệm vector công bằng cực đại – cực tiểu và khái niệm điểm nghẽn.



Hình 5-17: Ví dụ về tính công bằng cực đại – cực tiểu

Trên hình 5-17, điểm nghẽn của các kết nối 1, 2, 3, 4 và 5 lần lượt là $(3,5)$, $(2,3)$, $(2,3)$, $(4,5)$ và $(2,3)$. Liên kết $(3,5)$ không phải điểm nghẽn cho kết nối 5 vì liên kết này được chia sẻ bởi hai kết nối 1 và 5 trong đó kết nối 1 có tốc độ cao hơn kết nối 5 trên liên kết này. Liên kết $(1,3)$ không phải là điểm tắc nghẽn của tất cả các kết nối vì tài nguyên trên kết nối này chưa được sử dụng hết (còn dư thừa $1/3$ tốc độ)

Thuật toán tìm giá trị băng thông tối ưu (max-min fair algorithm)

Phần này sẽ trình bày thuật toán tìm giá trị băng thông tối ưu.

1) Khởi tạo tất cả các kết nối với tốc độ = 0

Tăng tốc độ của tất cả các kết nối với một lượng nhỏ bằng nhau δ , lặp lại quá trình này cho đến khi tồn tại các liên kết có tổng băng thông đạt đến giá trị băng thông cực đại ($F_a = C_a$). Lúc này:

- Tất cả các kết nối chia sẻ liên kết này đều sử dụng băng thông bằng nhau
- Liên kết này là điểm tắc nghẽn đối với tất cả các kết nối sử dụng liên kết này
- Ngừng việc tăng băng thông cho các kết nối này vì các kết nối này đã đạt đến trạng thái cân bằng cực đại – cực tiểu

2) Lặp lại quá trình tăng tốc độ cho các kết nối khác chưa đạt đến điểm tắc nghẽn cho đến khi lại tìm thấy các điểm tắc nghẽn ứng với các kết nối khác (lặp lại bước 2)

3) Thuật toán kết thúc khi tất cả các kết nối đều đã tìm được điểm tắc nghẽn.

Có cần phải minh họa bằng công thức không???

Ví dụ: xét trường hợp tìm băng thông tối ưu trong phương pháp công bằng cực đại – cực tiểu như trên hình 1-17. Giả thiết tất cả các liên kết đều có tốc độ là 1.

- Bước 1: tất cả các kết nối đều có tốc độ $1/3$, liên kết $(2,3)$ bão hòa (đạt giá trị cực đại) và tốc độ của ba kết nối $(2, 3$ và $5)$ đặt trên liên kết này được đặt ở giá trị $1/3$.

- Bước 2: hai kết nối 1 và 4 được tăng thêm một lượng bằng thông là 1/3 và đạt giá trị 2/3. Lúc này liên kết (3,5) bão hòa và tốc độ của kết nối 1 đặt ở giá trị 2/3
- Bước 3: kết nối 4 được tăng thêm một lượng là 1/3 và đạt đến giá trị 1. Liên kết (4,5) lúc này trở nên bão hòa và tốc độ của kết nối 4 đạt được là 1.
- Bước 4: cúc này tất cả các kết nối đều đã đi qua các liên kết bão hòa (điểm nghẽn) nên giải thuật dừng lại đây và kết quả của giải thuật tìm giá trị băng thông tối ưu chính là băng thông của các kết nối cho ở phần trên.

Dưới đây là thuật toán tìm giá trị băng thông tối ưu. Quy ước:

- A^k là tập hợp các liên kết chưa bão hòa (chưa hoạt động với tốc độ cực đại của liên kết) tại lúc bắt đầu bước k .
- P^k là tập hợp các kết nối không đi qua liên kết bão hòa nào, tính tại lúc bắt đầu của bước k
- n_a^k là số lượng kết nối trong P^k sử dụng liên kết a . Đây là số kết nối sẽ chia sẻ phần dung lượng đường truyền còn chưa dùng hết của liên kết a .
- \tilde{r}^k là phần băng thông tăng lên cho mỗi kết nối trong P^k tại bước k
- Tại điều kiện ban đầu: $k = 1, F_a^0 = 0, r_p^0 = 0, P^1 = P$ và $A^1 = A$

Thuật toán hoạt động như sau:

$$\begin{aligned}
 n_a^k &:= \text{số lượng đường } p \in P^k \text{ với } \delta_p(a) = 1 \\
 \tilde{r}^k &:= \min_{a \in A^k} (C_a - F_a^{k-1}) / n_a^k \\
 r_p^k &= \begin{cases} r_p^{k-1} + \tilde{r}^k & (p \in P^k) \\ r_p^{k-1} & (p \notin P^k) \end{cases} \\
 F_a^k &:= \sum_{a \in A} \delta_p(a) \cdot r_p^k \\
 A^{k+1} &:= \{a \mid C_a - F_a^k > 0\} \\
 P^{k+1} &:= \{p \mid \delta_p(a) = 0, \text{for all } a \notin A^{k+1}\} \\
 k &:= k + 1 \\
 \text{Nếu } P^k &\text{ là tập hợp rỗng thì dừng lại, nếu không} \\
 &\text{thì quay lại bước 1.}
 \end{aligned}$$

6.4.3. Thuật toán GPS (pending)

6.5. Bài tập (Pending)

Chương 7 Kỹ thuật mô phỏng

7.1. Giới thiệu

Công cụ NS2 (network simulator version 2) [5] được phát triển bởi trường Đại học Berkeley (Mỹ) là một công cụ cho phép mô phỏng và đánh giá đặc tính của mạng máy tính và viễn thông thay thế cho việc tiến hành thực nghiệm trên thiết bị thực tế. Do có một số ưu điểm như mã nguồn mở, có các module ứng dụng phong phú, NS2 hiện là một trong những công cụ mô phỏng được phổ biến rộng rãi nhất hiện nay trên thế giới, đặc biệt là trong các viện nghiên cứu và trường đại học.

Trong chương này, trước tiên chúng tôi sẽ trình bày khái niệm chung về phương pháp mô phỏng dựa trên các sự kiện rời rạc (discrete event simulation). Tiếp theo, nhằm cung cấp cho người đọc một cái nhìn tổng quan về các công cụ mô phỏng cho mạng, chúng tôi sẽ giới thiệu một số công cụ mô phỏng mạng thông dụng hiện nay và phân tích các ưu nhược điểm của chúng. Cấu trúc của NS2, các module có sẵn cũng như ứng dụng của chúng sẽ được trình bày trong phần tiếp theo. Sau cùng là một số kết luận chung về phạm vi ứng dụng cũng như ưu nhược điểm của NS2.

7.2. Mô phỏng dựa trên các sự kiện rời rạc và các công cụ

7.2.1. Phương pháp mô phỏng dựa trên sự kiện rời rạc

Trước khi đi vào trình bày khái niệm mô phỏng dựa trên sự kiện rời rạc, chúng tôi định nghĩa một số khái niệm sau:

Định nghĩa 6.1 - Mô hình mô phỏng (Simulation Model): là sự biểu diễn một hệ thống cần mô phỏng bằng cách mô tả các mối quan hệ toán học, logic hoặc cấu trúc của nó về mặt trạng thái, các thực thể làm nên hệ thống, sự kiện làm thay đổi trạng thái hệ thống, các tiến trình hoặc các hoạt động của hệ thống đó.

Định nghĩa 6.2 - Trạng thái hệ thống (System State): là tập hợp các biến cần thiết chứa đựng đầy đủ thông tin để mô tả một hệ thống tại một thời điểm bất kỳ.

Định nghĩa 6.3 - Thực thể (Entity): Một mô hình của hệ thống cần mô phỏng được chia nhỏ thành các thực thể với các chức năng khác nhau (thí dụ hàng đợi, server, gói dữ liệu .v.v.)

Định nghĩa 6.4 - Thuộc tính (Attributes): Mỗi thực thể trong một hệ thống sẽ có các thuộc tính khác nhau đặc trưng cho thực thể đó, thí dụ như luật phục vụ các gói trong một hàng đợi .v.v..

Định nghĩa 6.5 - Sự kiện (Event): Sự xuất hiện của một sự kiện sẽ làm thay đổi trạng thái của một hệ thống (thí dụ sự kiện xuất hiện của gói mới sẽ làm tăng số gói đang chờ trong một hàng đợi).

Định nghĩa 6.6 - Bản ghi sự kiện (Event Notice): Là một bản ghi có gắn thời gian sẽ xảy ra một sự kiện trong tương lai, cùng với nó là những dữ liệu cần thiết để thực hiện sự kiện đó, thí dụ như kiểu sự kiện và thời gian xảy ra sự kiện.

Định nghĩa 6.7 - Danh sách sự kiện (Event List): Là một danh sách chứa nhiều bản ghi sự kiện được sắp xếp theo trình tự thời gian xảy ra các sự kiện đó.

Định nghĩa 6.8 - Hoạt động (Activity): là một quãng thời gian với độ dài **được xác định** (khoảng thời gian truyền một gói tin, thời gian đến giữa hai gói tin liên tiếp) và thời điểm bắt đầu của hoạt động đó cũng đã được xác định.

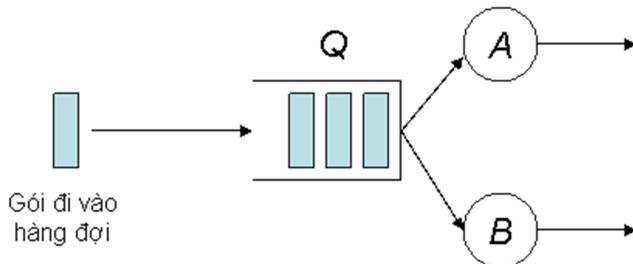
Định nghĩa 6.9 - Trễ (Delay): là một quãng thời gian với độ dài **không xác định** (như khoảng thời gian đợi của một gói tin trong một hàng đợi khi đằng trước nó còn n gói đang đợi).

Định nghĩa 6.10 - Đồng hồ (Clock): Là một biến số thể hiện thời gian mô phỏng của một hệ thống.

Từ những khái niệm cơ bản trên, phương pháp mô phỏng dựa trên sự kiện rời rạc được xây dựng bằng cách mô hình hóa một hệ thống mà trạng thái của nó thay đổi tại các thời điểm rời rạc, tức là thời điểm xảy ra một sự kiện nào đó. Như vậy quá trình chạy một mô phỏng thực chất là quá trình khảo sát một hệ thống khi trạng thái của nó thay đổi từ thời điểm này sang thời điểm khác, tương ứng với thời điểm xảy ra các sự kiện theo trình tự thời gian tăng dần.

Thí dụ 6.1:

Để dễ hiểu có thể lấy một thí dụ về một hệ thống bao gồm một hàng đợi Q và hai thực thể phục vụ (server) A và B cùng phục vụ các gói đang đợi ở Q. Đầu tiên các gói sẽ đi vào hàng đợi Q và đợi cho đến lượt mình được phục vụ. Thực thể A và B có thời gian phục vụ gói trung bình là t_{sa} và t_{sb} (đây chính là hai thuộc tính tương ứng với A và B). Khi có một gói đến, nếu A đang rỗi thì A sẽ phục vụ gói đó, nếu A bận B rỗi thì B phục vụ, nếu không gói sẽ đợi tại hàng đợi Q (Hình 6.1).



Hình 6.1. Hệ thống gồm 1 hàng đợi và 2 thực thể phục vụ

Có thể mô hình hóa hệ thống này bằng 3 trạng thái hiện bằng 3 tham số:

- L_Q : độ dài hàng đợi (số gói hiện tại có trong Q)
- S_A : 0 – A bận; 1 – A rỗi
- S_B : 0 – B bận; 1 – B rỗi

Ngoài ra cũng có thể định nghĩa 3 kiểu sự kiện làm thay đổi trạng thái của hệ thống như sau:

- 1) Sự kiện E_1 : một gói P_i nào đó đi vào hàng đợi;
- 2) Sự kiện E_2 : gói P_i bắt đầu được phục vụ bởi A hoặc B ;
- 3) Sự kiện E_3 : gói P_i được phục vụ xong.

Giả sử tại thời điểm t_1 gói P_n được A phục vụ xong, P_{n+1} bắt đầu được phục vụ, tại thời điểm t_2 gói P_i đi vào hàng đợi Q .

Đồng hồ	Trạng thái hệ thống	Danh sách sự kiện	Thực thể liên quan đến sự kiện	Các số liệu thống kê
t ($t < t_1$)	$L_Q(t)$ $S_A(t)=1, S_B(t)=1$	$(E3, t_1) - P_n$ được phục vụ xong $(E1, t_2) - P_i$ đi vào hàng đợi Q	A Q	...

- Bước 1: Xoá $(E3, t_1)$ ra khỏi danh sách sự kiện
- Bước 2: Đưa đồng hồ từ thời điểm t lên t_1
- Bước 3: Thực hiện sự kiện $(E3, t_1)$: bắt đầu phục vụ P_{n+1} , thay đổi các biến trạng thái hệ thống
- Bước 4: Tạo ra một bản ghi sự kiện mới: $(E3, t^*)$ là sự kiện gói P_{n+1} được phục vụ xong; Chèn bản ghi vào danh sách sự kiện
- Bước 5: Tính toán các số liệu thống kê của hệ thống tại t , (TD: khoảng thời gian chờ trung bình của một gói)

Đồng hồ	Trạng thái hệ thống	Danh sách sự kiện	Thực thể liên quan đến sự kiện	Các số liệu thống kê
t_1	$L_Q(t_1)=L_Q(t)+1$ $S_A(t)=1, S_B(t)=1$	$(E3, t^* < t_2) - P_{n+1}$ được phục vụ xong $(E1, t_2) - P_i$ đi vào hàng đợi Q	A Q	...

Hình 6.2. Mô phỏng hệ thống với trình tự thời gian tăng dần

Hình 6.2 thể hiện quá trình mô phỏng một hệ thống theo trình tự thời gian của đồng hồ và quá trình thay đổi, bổ sung các bản ghi sự kiện trong bản danh sách sự kiện. Việc xử lý danh sách sự kiện là một trong những nhiệm vụ chính của bất kỳ một chương trình mô phỏng nào. Do các bản ghi sự kiện là một chuỗi được sắp xếp theo trình tự thời gian, một danh sách sự kiện bao giờ cũng có hai con trỏ: một con trỏ trả vào đầu bản danh sách và con trỏ thứ hai trả vào bản ghi cuối cùng trong danh sách. Mỗi bản ghi cũng phải có các con trỏ trả đến bản ghi tiếp theo nằm trong bản danh sách. Các thao tác liên quan đến danh sách sự kiện bao gồm:

- 1) Xoá bản ghi đầu danh sách;
- 2) Xoá bản ghi ở vị trí bất kỳ trong danh sách;
- 3) Thêm một bản ghi vào đầu hoặc cuối danh sách;
- 4) Thêm một bản ghi vào vị trí bất kỳ trong danh sách phụ thuộc vào thời gian xảy ra sự kiện. Các phương pháp mô hình hoá một hệ thống thông tin cũng như các chi tiết về kỹ thuật mô phỏng có thể tìm thấy trong [1][2][3].

7.2.2. Các công cụ mô phỏng thông dụng dựa trên sự kiện rời rạc

Trước khi đi vào trình bày cấu trúc của công cụ NS2, phần này sẽ điểm lại một số công cụ mô phỏng thông dụng hiện nay và nhận xét ưu nhược điểm của chúng.

OPNET [8] là một sản phẩm thương mại tương đối nổi tiếng của công ty OPNET, bao gồm hai phần chính là OPNET Modeler và phần mở rộng cho mạng không dây OPNET Wireless Module. OPNET chạy dưới môi trường Windows cũng như Unix/Linux. OPNET rất thích hợp cho các tổ chức công nghiệp trong việc quy hoạch và đánh giá chất lượng dịch vụ của mạng thực tế bởi nó có sẵn một thư viện rất phong phú với các module mô phỏng thiết bị của nhiều nhà sản xuất khác nhau như Cisco, Lucent, Juniper. Tuy nhiên đối với các cơ sở nghiên cứu và trường đại học, có lẽ OPNET không phù hợp do giá tương đối đắt, mặt khác khi mô hình hoá một hệ thống, OPNET yêu cầu phải sử dụng thư viện với các thiết bị cụ thể nên việc xây dựng các mô hình tổng quát sẽ gặp khó khăn.

Ptolemy II [9] là một bộ công cụ mô phỏng trên nền Java được phát triển bởi trường Berkeley (Mỹ). Ptolemy II có thể được tải xuống miễn phí, tuy nhiên Ptolemy II chỉ cung cấp môi trường mô phỏng dựa trên sự kiện rời rạc nói chung, các module hỗ trợ cho mô phỏng hệ thống mạng không có nhiều nên người lập trình phải tự phát triển các ứng dụng của riêng mình.

OMNET++ [10] là chương trình mô phỏng cho hệ thống mạng được phát triển bởi Andras Varga, trường Đại học Bách khoa Budapest. OMNET++ được viết bằng ngôn ngữ C++ và hỗ trợ cả Windows lẫn Unix/Linux. OMNET++ có thể tải xuống miễn phí. Ngoài ra OMNET++ sử dụng giao diện đồ họa thân thiện với người sử dụng (như trong môi trường phát triển của OPNET), do đó khối lượng công việc và độ phức

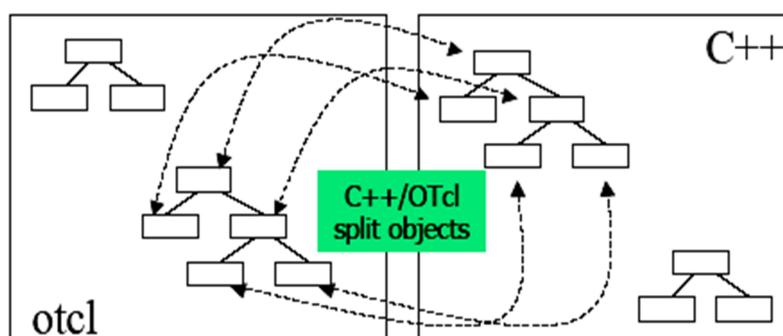
tập khi phát triển một module mới được giảm nhẹ khá nhiều. Tuy nhiên OMNET++ vẫn còn khá mới trong cộng đồng nghiên cứu nên các module có sẵn vẫn chưa nhiều.

NS2 [5] là công cụ mô phỏng mạng được sử dụng khá rộng rãi hiện nay trong các trường đại học và viện nghiên cứu. NS2 được phát triển trong khuôn khổ của dự án VINT, kết hợp giữa trường Berkeley, Viện Khoa học thông tin ISI, Xerox PARC và phòng thí nghiệm quốc gia Lawrence Berkeley. NS2 là công cụ mô phỏng hướng đối tượng, được phát triển dựa trên hai ngôn ngữ là C++ và OTcl (Object-oriented Tcl), chủ yếu chạy trong môi trường Unix/Linux. Ưu điểm của NS2 là mã nguồn mở, có cộng đồng sử dụng và phát triển khá đông đảo nên các module hỗ trợ cho mô phỏng mạng (như các giao thức, các cơ chế đảm bảo chất lượng dịch vụ, các công nghệ mạng lớp 2, 3) rất phong phú. Tuy nhiên nó cũng có một số nhược điểm:

- Do không có giao diện đồ họa với người sử dụng nên việc tạo các kịch bản mô phỏng cũng như phát triển các module mới phức tạp hơn các công cụ khác như OPNET hoặc OMNET++;
- Khả năng hỗ trợ các hệ điều hành khác như Windows kém;
- Do được phát triển bởi nhiều cá nhân và tổ chức khác nhau nên cấu trúc NS2 tương đối phức tạp, sau một thời gian làm quen và dùng thử nhất định người sử dụng mới có khả năng làm chủ chương trình, đặc biệt khi phải tạo ra các module chức năng mới. Sau đây chúng tôi sẽ tập trung giới thiệu công cụ NS2. Việc so sánh và liệt kê công cụ mô phỏng và đánh giá hoạt động của mạng có thể tìm thấy trong [2][11][12].

7.3. Công cụ mô phỏng mạng NS2

7.3.1. Cấu trúc



Hình 3. Cấu trúc của công cụ mô phỏng NS

Mô phỏng NS được xây dựng trên cơ sở hai ngôn ngữ:

- **C++:** NS có một thư viện phong phú về đối tượng mạng và giao thức được mô tả bằng C++ (thí dụ như các nút mạng, đường nối, nguồn, hàng đợi .v.v.).

- **OTcl:** Ngoài ra chương trình thông dịch OTcl (OTcl là ngôn ngữ mở rộng các chức năng hướng đối tượng của Tcl) cho phép người sử dụng xây dựng các kịch bản mô phỏng cụ thể và truyền tham số cho các thực thể C++. Mỗi đối tượng (tương ứng với từng thực thể) trong C++ sẽ có một đối tượng tương ứng ở lớp OTcl như thể hiện ở Hình 3.

Như vậy C++ là phần cho dữ liệu và là lõi của NS còn OTcl là phần đặt cấu hình cho chương trình mô phỏng. NS phải sử dụng 2 ngôn ngữ là do có hai nhiệm vụ khác nhau khi tiến hành mô phỏng. Một mặt, mô tả chi tiết các giao thức, các khối hoặc cơ chế của mạng yêu cầu phải sử dụng các ngôn ngữ bậc cao để xử lý số liệu, thực hiện các thuật toán. Đối với nhiệm vụ này do yêu cầu về tính hiệu quả của chương trình mô phỏng (như khoảng thời gian chạy chương trình, quản lý bộ nhớ .v.v.), các thực thể bắt buộc phải được viết dưới C++. Mặt khác, các quá trình xây dựng một kịch bản mô phỏng như đặt cấu hình cho các phần tử mạng, truyền các tham số cụ thể, thiết lập topo cho mạng thì chỉ sử dụng các phần tử đã có sẵn nên yêu cầu ở khâu này là thời gian thiết lập một cấu hình phải thấp (vì các kịch bản mô phỏng có thể lắp đi lắp lại). Vì vậy, một chương trình thông dịch như OTcl là thích hợp.

Trong một kịch bản mô phỏng dưới dạng OTcl do người dùng đưa ra, chúng ta có thể thiết lập một topo mạng, những giao thức và ứng dụng cụ thể mà chúng ta muốn mô phỏng và mẫu của đầu ra mà chúng ta mong nhận được từ mô phỏng, OTcl có thể sử dụng những đối tượng được biên dịch trong C++ qua một liên kết OTcl (sử dụng `tclCL` là thư viện gắn kết để dễ dàng chia sẻ chức năng và biến) để tạo ra một ánh xạ 1-1 của đối tượng OTcl cho mỗi đối tượng C++ cũng như định nghĩa sự liên hệ giữa các đối tượng đó.

Như đã trình bày ở trên, một trong những phần cơ bản của NS cũng là bản danh sách sự kiện mà ở đây người ta gọi là *bộ phân hoạch sự kiện* (event scheduler). NS sử dụng 4 phương pháp phân hoạch sự kiện khác nhau, được trình bày cụ thể trong [4]. Một sự kiện là một đối tượng trong C++ bao gồm một số hiệu nhận dạng (ID) duy nhất, thời gian được phân hoạch và con trỏ trả đến một đối tượng thực thi sự kiện đó.

Cấu trúc của một sự kiện và bộ phân hoạch sự kiện được định nghĩa như sau:

```
class Event {
public:
    Event* next_; /* event list */
    Handler* handler_; /* handler to call when
event ready */
    double time_; /* time at which event is ready
*/
    int uid_; /* unique ID */
    Event() : time_(0), uid_(0) {}
```

```

};

/*
 * The base class for all event handlers. When
an event's scheduled
 * time arrives, it is passed to handle which
must consume it.
 * i.e., if it needs to be freed it, it must be
freed by the handler.
*/
class Handler {
public:
virtual void handle(Event* event);
};

```

Các gói tin trong NS được định nghĩa từ lớp *Event* như sau:

```

class Packet : public Event {
private:
friend class PacketQueue;
u_char* bits_;
...
protected:
static Packet* free_;
public:
Packet* next_; /* for queues and the free list
*/
static int hdrlen_;
Packet() : bits_(0), datalen_(0), next_(0) {}
u_char* const bits() { return (bits_); }
static void free(Packet* );
...
};

```

7.3.2. Các tiện ích trong NS hỗ trợ cho mô phỏng mạng [Pending]

Các module phục vụ cho mô phỏng mạng máy tính và viễn thông: Mobile networks, mobile IP, DiffServ, IntServ, MPLS, UDP/TCP/IP, SCTP, routing protocols (mobile ad-hoc, unicast, multicast), RED, RIO, WFQ, CSMA/CD, ON/OFF source, Pareto .v.v.

Các chương trình trợ giúp trong việc khai thác số liệu mô phỏng: Nam, XGraph .v.v.

7.3.3. Thí dụ (Pending)

7.4. Kết luận (Pending)

7.5. Bài tập (Pending)

Tài liệu tham khảo

- [1] John S. Carson II, Barry L. Nelson, *Discrete-Event System Simulation*, Jerry Banks, Prentice Hall 1996
- [2] Richard Blum, *Network Performance Open Source Toolkit Using Netperf, tcptrace, NIST Net, and SSFNet*, Wiley Publishing 2003
- [3] Raj Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*, John Wiley and Sons 1991
- [4] Kannan Varadhan, Kevin Fall, *NS Manual*, <http://www.isi.edu/nsnam/ns/documentation.html>
- [5] <http://www.isi.edu/nsnam/ns/>
- [6] Marc Greis, *NS Tutorial*, <http://www.isi.edu/nsnam/ns/tutorial/index.html>
- [7] Eintan Altman, Tania Jiménez, *NS for Beginners*, <http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf>
- [8] <http://www.opnet.com>
- [9] <http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm>
- [10] <http://www.omnetpp.org/>
- [11] <http://www.inrialpes.fr/planete/people/ernst/Documents/simulator.html>
- [12] <http://www.topology.org/soft/sim.html>
- [13] Kishor Shridharbhai Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, Wiley-Interscience, 2001
- [14] Donald Gross, Carl M. Harris, *Fundamentals of Queueing Theory*, Wiley-Interscience, 1998
- [15] Dimitri Bertsekas, Robert Gallager, *Data Networks*, Prentice-Hall International Editions, 1987
- [16] Andrew S. Tanenbaum, *Computer Networks*, Prentice-Hall, 2003
- [17] Joseph L. Hammond, Peter J.P.O' Reilly, *Performance Analysis of Local Computer Networks*, Addison-Wesley, 1988
- [18] Jeremiah F. Hayes, Thimma V. J. Ganesh Babu, *Modeling and Analysis of Telecommunications Networks*, Wiley-Interscience, 2004
- [19] Gunter Bolch, Stefan Greiner, Hermann de Meer, Kishor S. Trivedi, *Queueing Networks and Markov Chains, Modeling and Performance Evaluation with Computer Science Evaluation*, John Wiley and Sons, 1998
- [20] Rudolf Avenhaus, *Quantitative Modelle für Rechen- und Kommunikationssysteme*, Universität der Bundeswehr München, 2000

