BÀI 3: LIÊN KẾT DỮ LIỆU VÀ CÁC GIAO THỨC TRUYỀN THÔNG TÌM ĐƯỜNG

Nội dung

- Thủ tục liên kết dữ liệu cơ bản.
- Các giao thức (hướng bit, hướng ký tự).
- Mã phát hiện sai và sửa sai.
- Tìm đường đi trong mạng.
- Các giải thuật chống tắc nghẽn.

Hướng dẫn học

- Đọc kỹ giáo trình bài giảng.
- Thực hành các mã phát hiện và sửa sai bằng cách tự lấy ví dụ áp dụng để phát hiện sai và sửa sai.
- Tự lấy ví dụ cho các giải pháp tìm đường và thực hiện chúng.
- Làm bài tập, trả lời các câu trắc nghiệm.

Thời lượng học

- 15 tiết.

Mục tiêu

Sau khi học bài này, các bạn có thể:

- Biết được các thủ tục liên kết cơ bản.
- Hiểu được tổng quan về hai loại giao thức hướng bit và hướng ký tự.
- Nắm rõ được các cơ chế phát hiện sai và sửa sai như: chẵn lẻ, checksum bằng đa thức chuẩn, mã Hamming tự sửa một sai.
- Thành thạo việc tính toán để phát hiện sai và sửa sai.
- Biết được ý nghĩa, mục tiêu của việc tìm đường trong mạng. Hiểu được cơ chế của các giải thuật tìm đường trong mạng.
- Giải thuật tìm đường đi tối ưu.
- Giải pháp tìm đường Vector khoảng cách.
- Giải pháp chọn đường "Trạng thái kết nối".
- Tìm đường phân cấp.
- Tìm đường trong mạng di động.
- Hiểu được nguyên nhân, bản chất của tắc nghẽn trong mạng.
- Nắm rõ các nguyên tắc, biện pháp phòng ngừa và điều khiển tắc nghẽn trong mạng.

TÌNH HUỐNG KHỞI ĐỘNG BÀI

Tình huống dẫn nhập

Chắc các bạn đã biết muốn 2 máy tính giao tiếp được với nhau ta cần có cáp mạng, nhưng không phải chỉ đơn giản thế thôi chứ?

Thực ra nó không đơn giản vậy đâu. Để 2 máy tính có thể giao tiếp, truyền dữ liệu qua lại với nhau, nhân loại đã tốn không ít công sức và trí lực để xây dựng những quy tắc (giao thức). Nếu không có những giao thức này thì hai máy tính cũng chẳng khác gì người Việt Nam không biết tiếng Lào và người Lào không biết tiếng Việt Nam nói chuyện với nhau.

Tuy nhiên với mạng máy tính (một mạng có rất nhiều máy tính) thì chỉ giao tiếp, truyền dữ liệu thôi cũng có rất nhiều vấn đề nan giải như: truyền phải trúng đích, tin truyền phải chính xác,… rồi vấn đề tắc nghẽn đường truyền,…

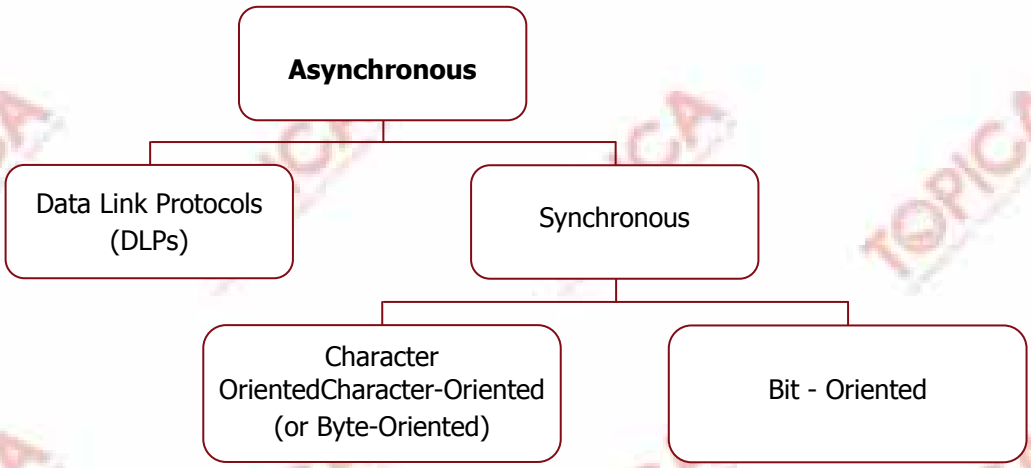
Tất cả những điều này sẽ được giải đáp sau khi bạn học bài "Liên kết dữ liệu và các giao thức truyền thông tìm đường" này.

Câu hỏi

1. Thủ tục liên kết dữ liệu gồm những giao thức nào ?
2. Giải quyết vấn đề sai lệch thông tin trong quá trình truyền dữ liệu như thế nào?
3. Các giải thuật tìm đường trên mạng là gì?
4. Vấn đề tắc nghẽn đường truyền: phát hiện và giải quyết nó như thế nào?

3.1. Thủ tục liên kết dữ liệu cơ bản

Các giao thức được xây dựng cho tầng Liên kết dữ liệu (gọi chung là – Data Link Protocol). Các DLP được phân chia thành loại: dị bộ (Asynchoronous DLP) và đồng bộ (Synchronous DLP), trong đó loại "đồng bộ" lại chia thành hai nhóm là hướng ký tự (Character-Oriented) và hướng bit (Bit-Oriented).



Hình 3.1: ***Phân loại các giao thức liên kết dữ liệu***

- DLP dị bộ:
  - Các DLP dị bộ sử dụng phương thức truyền dị bộ, trong đó các bit đặc biệt START và STOP được sử dụng để tách các xâu bit biểu diễn các ký tự trong dòng dữ liệu cần truyền đi. Phương thức này được gọi là "dị bộ" là vì không cần có sự đồng bộ liên tục giữa người gửi và người nhận tin. Nó cho phép một ký tự dữ liệu được truyền đi bất kỳ lúc nào mà không cần quan tâm đến các tín hiệu đồng bộ trước đó.
  - Các giao thức loại này thường được dùng trong các máy điện báo hoặc các máy tính trạm cuối tốc độ thấp. Phần lớn các máy PC sử dụng phương thức truyền dị bộ do tính đơn giản của nó.
- DLP đồng bộ:
  - Phương thức truyền đồng bộ không dùng các bit đặc biệt START, STOP để "đóng khung" mỗi ký tự mà chèn các ký tự đặc biệt như SYN (Synchronization), EOT (End of Transmission) hay đơn giản hơn, một cái "cờ" (Flag) giữa các dữ liệu của người sử dụng để báo hiệu cho người nhận biết được dữ liệu "đang đến" hoặc "đã đến".
  - Cần lưu ý rằng các hệ thống truyền thông đòi hỏi hai mức đồng bộ hóa:
    - Ở mức vật lý: để giữ đồng bộ giữa các đồng hồ của người gửi và người nhận.
    - Ở mức liên kết dữ liệu: để phân biệt dữ liệu của người sử dụng với các "cờ" và các vùng thông tin điều khiển khác.
  - Các DLP hướng ký tự được xây dựng dựa trên các ký tự đặc biệt của một mã chuẩn nào đó (như ASCII hay EBCDIC), trong khi các DLP hướng bit lại dùng các cấu trúc nhị phân (xâu bit) để xây dựng các phần tử của giao thức (đơn vị dữ liệu, các thủ tục, …) và khi nhận, dữ liệu sẽ được tiếp nhận lần lượt từng bit một.

Dưới đây chúng ta sẽ xem xét kỹ hơn hai loại giao thức đồng bộ đó thông qua các ví dụ minh họa điển hình.

3.2. Các giao thức (hướng bit, hướng ký tự)

3.2.1. Các giao thức hướng ký tự

Các giao thức loại này xuất hiện từ những năm 1960 và giờ đây vẫn còn được sử dụng. Chúng được dùng cho các ứng dụng điểm-điểm (Point to Point) lẫn nhiều điểm (Multipoint). Giao thức loại này có thể đáp ứng cho các phương thức khai thác đường truyền khác nhau: một chiều (Simplex), hai chiều luân phiên (Half-Duplex) hoặc hai chiều đồng thời (Full-Duplex).

Đối với phương thức một chiều, giao thức hướng ký tự được dùng rộng rãi nhất là giao thức truyền tệp Kermit do Đại học Columbia (Mỹ) chế tác. Kermit có nhiều phiên bản cho phép truyền tệp giữa 2 máy PC hoặc giữa một PC và một máy chủ (File server) hoặc một máy lớn (Mainframe).

Đối với phương thức hai chiều luân phiên, giao thức hướng ký tự nổi tiếng nhất chính là BSC (Binary Synchronous Control) hay còn gọi là Bisync – một sản phẩm của IBM. Giao thức này đã được ISO lấy làm cơ sở để xây dựng giao thức hướng ký tự chuẩn quốc tế với tên gọi Basic Mode. Bởi thế nó sẽ được trình bày minh họa một cách chi tiết trong phần dưới đây.

Có rất ít giao thức hướng ký tự được phát triển cho phương thức hai chiều đồng thời. Ví dụ điển hình trong số này là giao thức giữa các nút chuyển mạch (còn gọi là các IMP – Interface Message Protocols) trong mạng ARPANET nổi tiếng của Bộ quốc phòng Mỹ.

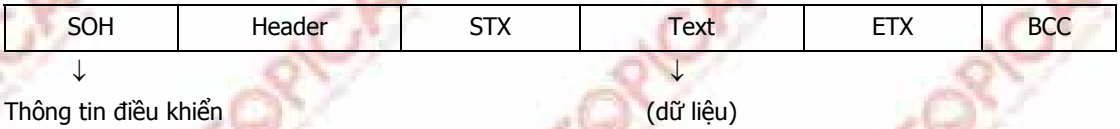
Giao thức BSC/Basic Mode:

Như trên đã nói, họ giao thức này áp dụng cho trường hợp điểm-điểm hoặc nhiều điểm, hai chiều luân phiên. Các ký tự đặc biệt của bộ mã chuẩn EBCDIC (đối với BSC) hoặc của bộ mã chuẩn ASCII (đối với Basic Mode của ISO) được sử dụng để xây dựng giao thức.

Các ký tự đặc biệt đó gồm có:

- SOH (Start Of Header): bắt đầu của phần header của một đơn vị thông tin chuẩn.
- STX (Start of Text): kết thúc của header và bắt đầu của phần dữ liệu (văn bản).
- ETX (End of Text): kết thúc của phần dữ liệu.
- EOT (End Of Transmission): kết thúc việc truyền của một hoặc nhiều đơn vị dữ liệu và để giải phóng liên kết.
- ETB (End of Transmission Block): kết thúc của một khối dữ liệu, trong trường hợp dữ liệu được chia thành nhiều khối.
- ENQ (Enquiry): yêu cầu phúc đáp từ một trạm xa.
- DLE (Data Link Escape): thay đổi ý nghĩa của các ký tự điều khiển truyền tin khác.
- NAK (Negative Acknowledge): báo cho người gửi biết là tiếp nhận không tốt thông tin.
- SYN (Synchronous Idle): ký tự đồng bộ, dùng để duy trì sự đồng bộ giữa người gửi và người nhận.

Một đơn vị dữ liệu (frame) dùng trong giao thức này có khuôn dạng tổng quát như sau:



Hình 3.2: ***Đơn vị dữ liệu trong giao thức BSC***

- Phần header (có thể vắng mặt) chứ thông tin điều khiển, thường là số thứ tự của frame và địa chỉ của trạm đích, …
- BCC (Block Check Character) là 8 bit kiểm tra lỗi theo kiểu bit chẵn lẻ (theo chiều dọc) cho các ký tự thuộc vùng Text (trường hợp Basic Mode), hoặc 16 bit kiểm tra lỗi theo phương pháp CRC-16 cho vùng Text (trường hợp BSC). Kích thước vùng Text trong cả 2 trường hợp đều được giới hạn để đảm bảo được kiểm soát lỗi khi truyền. Trường hợp dữ liệu quá dài có thể tách thành nhiều khối (block).

Ví dụ: ta có thể có 3 khối của một message như sau: *(Giả thiết phần Text được tách thành 3 phần con là Text1, Text2, Text3).*

Khối 1:	SOH	id	…	STX	Text1	ETX	BCC
		←——header——→					
Khối 2:	SOH	id	…	STX	Text2	ETX	BCC
Khối 3:	SOH	id	…	STX	Text2	ETX	BCC

Hình 3.3: ***Các khối của một Message***

Trong đó id (Identifier) là số thứ tự gửi của khối.

Các thủ tục chính của BSC/ Basic Mode gồm có:

Mời truyền tin

Giả sử trạm A muốn mời trạm B truyền tin, A sẽ gửi lệnh sau đây tới B

EOT	**B**	**ENQ**

Trong đó:

- B là địa chỉ của trạm được mời truyền tin.
- EOT để chuyển liên kết sang trạng thái điều khiển.

Khi B nhận được lệnh này, có thể xảy ra 2 trường hợp:

- Nếu có tin để truyền thì trạm B cấu trúc tin theo khuôn dạng chuẩn (đã trình bày ở trên) và gửi đi.
- Nếu không có tin để truyền thì B gửi lệnh EOT để trả lời.
- Ở phía A, sau khi gửi lệnh đi nếu quá một thời đoạn xác định trước mà không nhận được trả lời của B, hoặc nhận được trả lời sai, thì A sẽ chuyển sang trạng thái "phục hồi" (Recovery State). Hành động của một trạm khi ở trong trạng thái "phục hồi" sẽ được nói đến sau.

Mời nhận tin

Giả sử trạm A muốn mời trạm B nhận tin, lúc đó A sẽ gửi tới B một lệnh tương tự trường hợp trên:

EOT	**B**	**ENQ**

Ở đây EOT có thể vắng mặt.

Khi nhận được lệnh này, nếu B đã sẵn sàng nhận tin thì nó gửi ACK để trả lời, ngược lại nó gửi NAK để trả lời.

Về phí A, sau khi gửi lệnh đi nếu quá một thời đoạn xác định trước mà không nhận được trả lời của B hoặc nhận được trả lời sau, thì A sẽ chuyển sang trạng thái "phục hồi".

Yêu cầu trả lời: khi một trạm cần trạm kia trả lời một yêu cầu nào đó đã gửi đi trước đó thì nó chỉ cần gửi lệnh ENQ đến trạm kia.

Ngừng truyền tin (tạm thời): gửi lệnh EOT.

Giải phóng liên kết: gửi lệnh DLE EOT.

Trạng thái "phục hồi": khi một trạm nào đó đi vào trạng thái "phục hồi" thì nó sẽ thực hiện một trong các hành động sau:

- Lặp lại lệnh đã gửi n lần (n là một số nguyên cho trước).
- Gửi "yêu cầu trả lời" n lần.
- Kết thúc truyền bằng cách gửi lệnh EOT.

(Nếu sau n lần vẫn thất bại thì có thể thông báo cho tầng trên, hoặc cho người sử dụng).

Để thấy rõ hơn phương thức trao đổi thông tin của giao thức BSC/Basic Mode, ta sẽ dùng sơ đồ để minh họa, trong đó có sơ đồ cho trường hợp thông thường và cho cả trường hợp trao đổi ở dạng "hội thoại".

Lưu ý: dạng hội thoại là một cải tiến của dạng thông thường, trong đó một trạm có thể dùng ngay khối tin cần truyền của mình để trả lời (báo nhận tốt – thay cho ACK) cho một khối tin của trạm kia gửi đến. Như vậy sẽ giảm được khá nhiều thời gian

3.2.2. Các giao thức hướng bit

Giao thức HDLC (High-Level Data Link Control)

Giao thức điều khiển liên kết dữ liệu quan trong nhất là HDLC. Không phải vì nó được sử dụng rộng rãi mà nó còn là cơ sở cho nhiều giao thức điều khiển liên kết dữ liệu khác.

- Các đặc tính của giao thức HDLC.

  Giao thức HDLC định nghĩa 3 loại máy trạm, hai cấu hình đường kết nối và 3 chế độ điều khiển truyền tải.
- Ba loại trạm trong HDLC:
  - Trạm chính (Primary Station): có trách nhiệm điều khiển các thao thác về đường truyền. Các khung được gửi từ trạm chính gọi là lệnh (Command).
  - Trạm phụ (Secondary Station): hoạt động dưới sự kiểm soát của trạm chính. Khung gửi từ trạm phụ gọi là các trả lời. Trạm chính duy trì nhiều đường kết nối luận lý đến các trạm phụ trên đường truyền.
  - Trạm hỗn hợp (Combined Station): bao gồm đặc điểm của trạm chính và trạm phụ. Một trạm hỗn hợp có thể gửi đi các lệnh và các trả lời.
- Hai cấu hình đường kết nối:
  - Cấu hình không cân bằng (Unbalanced Configuration): gồm một máy trạm chính (Primary Station) và nhiều máy trạm phụ (Secondary Station) và hỗ trợ cả 2 chế độ truyền song công và bán song công.
  - Cấu hình cân bằng (Balanced Configuration): bao gồm 2 máy trạm hỗn hợp, và hỗ trợ cả 2 chế độ truyền song công và bán song công.
- Có 3 chế độ truyền tải là:

- Chế độ trả lời bình thường (NRM - Normal Response Mode): được sử dụng với cấu hình đường kết nối không cân bằng. Máy chính có thể khởi động một cuộc truyền tải dữ liệu về cho máy phụ. Nhưng máy phụ chỉ có thể thực hiện việc truyền dữ liệu cho máy chính như là những trả lời cho các yêu cầu của máy chính.
- Chế độ cân bằng bất đồng bộ (ABM - Asynchronous Balance Mode): được sử dụng với cấu hình kết nối cân bằng. Cả hai máy đều có quyền khởi động các cuộc truyền tải dữ liệu mà không cần sự cho phép của máy kia.
- Chế độ trả lời bất đồng bộ (ARM-Asynchronous Response Mode): sử dụng cấu hình không cân bằng. Một máy phụ có thể khởi động một cuộc truyền tải và không cần sự cho phép tường minh của máy chính. Máy chính vẫn đảm trách vai trò bảo trì đường truyền bao gồm việc khởi động, phục hồi lỗi và xóa kết nối.

Chế độ NRM đòi hỏi phải có nhiều đường dây để nối một máy chính với nhiều thiết bị đầu cuối. Chế độ ABM được sử dụng nhiều nhất trong 3 chế độ, nó cho phép sử dụng hiệu quả đường truyền. Chế độ ARM thì ít được dùng đến.

Cấu trúc khung

Bảng mô tả cấu trúc khung của HDLC

Bits	8	8	8	≥ 0	16	8
	01111110	Address	Control	Data	Checksum	011111110

HDLC sử dụng chế độ truyền tải đồng bộ, các bits dữ liệu truyền đi được gói vào trong các khung và sử dụng một cấu trúc khung cho tất cả các loại dữ liệu cũng như thông tin điều khiển. Khung trong giao thức HDLC có cấu trúc như sau:

Bảng mô tả ý nghĩa các bit

Flag (8 bit)	Là cờ dùng để xác định điểm bắt đầu và kết thúc của khung, giá trị nó là 01111110. HDLC sử dụng kỹ thuật bit độn để loại trừ sự xuất hiện của cờ trong dữ liệu.
Address (8 bit)	Vùng ghi địa chỉ để xác định máy phụ được phép truyền hay nhận khung.
Control (8 bit)	Được dùng để xác định loại khung. Mỗi loại có thông tin điều khiển khác nhau. Có 3 loại khung: Thông tin (I), điều khiển (S) và không đánh số (U).
Information (128-1024 bytes)	Vùng chứa dữ liệu cần truyền.
FCS (Frame Check Sequence – 8 bit)	Vùng chứa mã kiểm soát lỗi, dùng phương pháp đa thức CRC-CCITT = $X^{16} + X^{12} + X^5 + 1$

Giá trị 8 bit của trường control hình thành 3 loại khung như sau:

Bảng cấu trúc trường điều khiển trong khung HDLC

Bit	1	3	1	3
Khung I	0	Seq	P/F	Next
Khung S	1	0	Type	P/F	Next
Khung U	1	1	Type	P/F	Modifier

Giao thức HDLC sử dụng một cửa sổ trượt với số thứ tự khung 3 bít. Trường seq trong khung I để chỉ số thứ tự của khung thông tin hiện tại. Trường Next để chỉ số thứ tự của khung thông tin mà bên gửi đang chờ nhận (thay vì là khung đã nhận tốt như giao thứ cửa sổ trượt đã giới thiệu ở phần trước).

Bit P/F có ý nghĩa là Poll/Final, tức chọn hoặc kết thúc. Khi máy tính chính mời một máy phụ truyền tin, thì bit này được đặt lên 1 có ý nghĩa là P (Poll, chọn). Ngược lại khi thông tin được truyền từ máy phụ lên máy chính thì nó được đặt xuống 0, để báo với máy chính rằng máy phụ hiện tại vẫn còn dữ liệu để gửi đi. Khi máy phụ gửi khung cuối cùng, bit này được đặt lên 1, có ý nghĩa là F (Final, kết thúc), để báo cho máy chính biết rằng nó đã hoàn thành việc truyền tải thông tin.

Khung S (Supervisory Frame) là khung điều khiển, dùng để kiểm soát lỗi và luồng dữ liệu trong quá trình truyền tin. Khung S có 4 kiểu được xác định bởi tổ hợp giá trị của 2 bit trong trường Type.

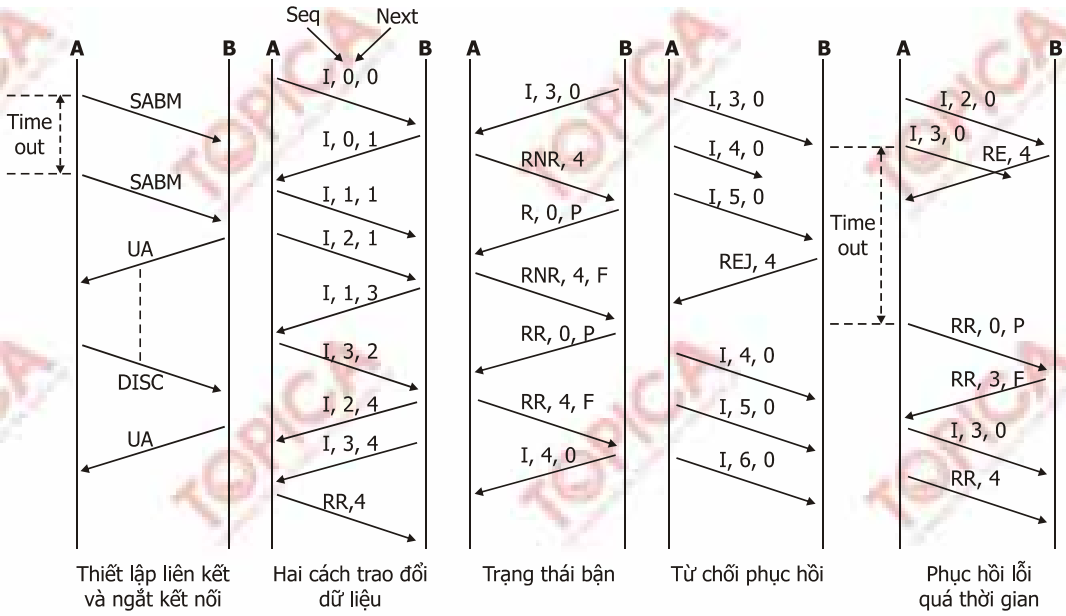
SS = 0	RR (Receive Ready): là khung báo nhận, thông báo sẵn sàng nhận dữ liệu, đã nhận tốt đến khung Next-1 và đang đợi nhận khung Next. Được dùng đến khi không còn dữ liệu gửi từ chiều ngược lại để vừa làm báo nhận (Figgyback).
SS = 01	REJ (Reject): đây là một khung báo không nhận (Negative acknowledge) yêu cầu gửi lại các khung, từ khung Next.
SS = 10	RNR (Receive Not Ready): thông báo không sẵn sàng nhận tin, đã nhận đến khung thứ Next-1, chưa sẵn sàng nhận khung Next.
SS = 11	SREJ (Selective Reject): yêu cầu gửi lại một khung có số thứ tự là Next.

Khung U (Unnumbered Frame) thường được sử dụng cho mục đích điều khiển đường truyền, nhưng đôi khi cũng được dùng để gửi dữ liệu trong dịch vụ không kết nối. Các lệnh của khung U được mô tả như sau:

1111P100	Lệnh này dùng để thiết lập chế độ truyền tải SABM (Set Asynchronous Balanced Mode).
1100P001	Lệnh này dùng để thiết lập chế độ truyền tải SNRM (Set Normal Response Mode).
1111P000	Lệnh này dùng để thiết lập chế độ truyền tải SARM (Set Asynchronous Response Mode).
1100P010	Lệnh này để yêu cầu xóa kết nối DISC (Disconnect).
1100F110	UA (Unumbered Acknowledgment). Được dùng bởi các trạm phụ để báo với trạm chính rằng nó đã nhận và chấp nhận các lệnh loại U ở trên.
1100F001	CMDR/FRMR (Command Reject/Frame Reject): được dùng bởi trạm phụ để báo rằng nó không chấp nhận một lệnh mà nó đã nhận chính xác.

Một vài kịch bản về giao thức HDLC

Kịch bản (a) mô tả các khung liên quan trong quá trình thiết lập và xóa kết nối. Đầu tiên một trong hai bên giao tiếp sẽ gửi khung SABM sang bên kia và thiết lập một bộ đếm thời gian. Bên phía còn lại khi nhận được khung SABM sẽ trả lời bằng khung UA. Bên yêu cầu kết nối khi nhận được khung UA sẽ xóa bỏ bộ đếm thời gian. Kết nối đã được hình thành và hai bên có thể truyền khung qua lại cho nhau. Kết nối sẽ xóa đi nếu một trong hai bên giao tiếp gửi khung DISC. Trong một trường hợp khác, nếu sau một khoảng thời gian trôi qua, bên yêu cầu kết nối không nhận được khung UA, nó sẽ cố gắng gửi lại khung SABM một số lần quy định. Nếu vẫn không nhận được khung UA, bên yêu cầu kết nối sẽ thông báo lỗi lên tầng cao hơn.



Hình 3.4: ***Một vài kịch bản của HDLC***

Kịch bản (b) mô tả tiến trình trao đổi khung I giữa hai bên. Ta thấy rằng bên A gửi liên tiếp các khung (I,1,1 và I,2,1) mà không nhận được khung báo nhận thì số thứ tự của khung chờ nhận vẫn không thay đổi, trong trường hợp này là 1. Ngược lại khi bên B nhận liên tiếp các khung (I,1,1 và I,2,1) mà không gửi khung nào đi, thì khung chờ nhận kế tiếp của khung thông tin truyền đi phải là số kế tiếp của khung vừa nhận, là 3.

Trong kịch bản (c) máy A không thể xử lý kịp các khung do B gửi đến vì thế nó gửi khung RNR để yêu cầu B tạm dừng việc việc truyền tải. Bên B định kỳ gửi thăm dò bên A bằng cách gửi khung RR với bit P được đặt lên 1. Nếu bên A vẫn chưa thể nhận thông tin từ bên B nó sẽ trả lời bằng khung RNR, ngược lại nếu A đã sẵn sàng thì nó sẽ trả lời bằng khung RR.

Trong kịch bản (d), bên A gửi sang B ba khung thông tin 3, 4 và 5. Khung 4 bị mất hoàn toàn trên đường truyền. Khi bên B nhận được khung 5, nó sẽ bỏ qua khung này vì sai thứ tự khung. B gửi REJ với trường Next là 4 để yêu cầu A gửi lại tất cả các khung từ khung số 4.

Kịch bản (e) minh họa cách thức phục hồi lỗi dựa vào thời gian (Time out). Khung số 3 bị lỗi và do đó B bỏ nó. B không thể gửi khung REJ vì nó không thể xác định được đó có phải là khung I hay không. Bên A sau một khoảng thời gian trôi qua không thấy khung trả lời từ B, nó sẽ gửi khung RR với bit P = 1 để kiểm tra trạng thái của bên kia. Bên B sẽ đáp lại bằng khung RR với trường Next là 3 để báo hiệu khung số 3 đã mất. Sau đó A sẽ truyền lại khung số 3.

3.3. Mã phát hiện sai và sửa sai

3.3.1. Chẵn lẻ (Parity)

Những bộ mã phát hiện lỗi (Error-Detecting Codes)

Có nhiều sơ đồ phát hiện lỗi, trong đó có các sơ đồ thông dụng là:

- Kiểm tra tính chẵn lẻ (Parity Checks).
- Kiểm tra thêm theo chiều dọc (Longitudinal Redundancy Check).
- Kiểm tra phần dư tuần hoàn (Cyclic Redundancy Check).

Kiểm tra chẵn lẻ (Parity Check)

Sơ đồ phát hiện bit lỗi đơn giản nhất là nối một bít chẵn-lẻ vào cuối mỗi từ trong khung. Một ví dụ tiêu biểu là việc truyền ký tự ASCII, mà trong đó một bit chẵn-lẻ được nối vào mỗi ký tự ASCII 7 bit. Giá trị của bit này được lựa chọn sao cho có một số chẵn của bit 1, với kiểm tra chẵn (Even Parity) hoặc một số lẻ của bit 1, với kiểm tra lẻ (Odd Parity).

Ví dụ: Nếu bên gửi truyền một ký tự ASCII G (mã ASCII là 110001) và đang dùng phương pháp kiểm tra lẻ, nó sẽ nối một bit 1 và truyền đi 1100011. Bên nhận sẽ kiểm tra ký tự nhận được và nếu tổng của các bit 1 là lẻ, nó xem như không có lỗi. Nếu một bit hoặc một số lẻ bất kỳ các bit bị lỗi đảo ngược thì rõ ràng bên nhận sẽ phát hiện được lỗi. Tuy nhiên, nếu hai hay một số bit chẵn bất kỳ các bit bị lỗi đảo ngược thì nó sẽ không phát hiện được lỗi. Trên thực tế những xung nhiễu lại thường đủ dài để có thể phá hủy hơn một bit, đặc biệt là với tốc độ đọc dữ liệu cao. Do đó, cần phải có một phương pháp cải thiện trường hợp này.

3.3.2. Check sum bằng đa thức chuẩn

Kiểm tra thêm theo chiều dọc (Longitudinal Redundancy Check or Checksum)

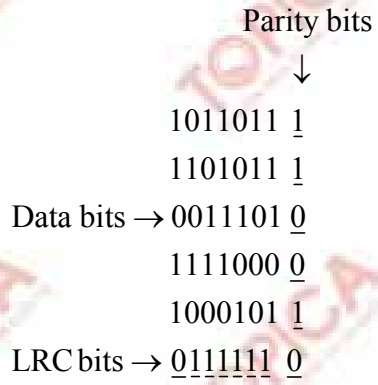
Có thể cải thiện sơ đồ trên bằng cách dùng phương pháp LRC. Trong phương pháp này, khung được xem như một khối nhiều ký tự được sắp xếp theo dạng hai chiều, và việc kiểm tra được thực hiện cả chiều ngang lẫn chiều dọc.

Theo chiều ngang, mỗi ký tự được thêm vào một bit kiểm tra chẵn lẻ như trường hợp trên, và được gọi là bit kiểm tra chiều ngang VRC (Vertical Redundancy Check).

Theo chiều dọc, cung cấp thêm một ký tự kiểm tra, được gọi là ký tự Kiểm tra chiều dọc LRC (Longitudinal Redundancy Check) hay Checksum. Trong đó, bit thứ I của ký tự này chính là bit kiểm tra cho tất cả các bit thứ I của tất cả các ký tự trong khối.

Các phép đo chỉ ra rằng việc dùng cả hai VRC và LRC giảm đi tỷ lệ lỗi không phát hiện được hai đến bốn bậc so với dùng chỉ VRC. Hãy xem trường hợp bit 1 và 3 trong ký tự 1 đang bị lỗi. Khi bên nhận tính toán được bit VRC cho ký tự 1, nó sẽ kiểm tra với bit VRC đã nhận và sẽ không phát hiện được lỗi. Tuy nhiên, khi nó tính toán được ký tự LRC, bit 1 và 3 của ký tự này sẽ khác với bit đó trong ký tự LRC nhận được, và sẽ phát hiện được lỗi.

Tuy nhiên, ngay sơ đồ này cũng không phải là thật sự tốt. Bây giờ, nếu giả sử bit 1 và 3 của ký tự 5 cũng bị lỗi, phương pháp này sẽ không phát hiện được điểm sai.



Hình 3.5: ***Kiểm tra chiều dọc tuần hoàn***

Kiểm tra phần dư tuần hoàn (Cyclic Redundancy Check)

Để cải tiến hơn nữa các nhà thiết kế đã dùng kỹ thuật mới dễ dàng và hiệu quả gọi là kiểm tra phần dư tuần hoàn, trong đó có thể sử dụng một số phương pháp cài đặt khác nhau như: modulo 2, đa thức, thanh ghi và các cổng Exclusive-or.

Các thủ tục với modulo 2 diễn ra như sau. Với một thông điệp M có k bit cần gửi đi, bên gửi sẽ nối vào cuối thông điệp một chuỗi F có r bit, được gọi là Chuỗi kiểm tra khung (FCS: Frame Check Sequence). Chuỗi kiểm tra khung sẽ tính toán sao cho khung kết quả T được hình thành từ việc nối M với F (gồm k + r bit) có thể chia hết bởi số P nào đó được định trước. Nếu phép chia không hết, tức có số dư, bên nhận xác định rằng khung T đã bị lỗi, ngược lại là không có lỗi. Nếu khung không có lỗi, bên nhận sẽ tách thông điệp M từ T, là k bits trọng số cao của T.

Phương pháp này dùng phép chia modulo 2 trong việc chia T cho P, phép toán modulo 2 dùng một phép cộng nhị phân không nhớ và đó cũng chính là phép toán Exclusive-or. Ví dụ sau mô tả phép toán cộng và nhân modulo 2:

$$\begin{array}{r} 1111 \\ +\ 1010 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 11001 \\ \times\ \ \ 11 \\ \hline 11001 \\ 11001 \\ \hline 101011 \end{array}$$

Giả sử ta có:

- M: Thông điệp k bit cần được gửi sang bên nhận.
- F: Chuỗi kiểm tra khung FCS gồm r bit là thông tin điều khiển được gửi theo M để giúp bên nhận có thể phát hiện được lỗi.
- T = MF là khung (k + r) bit, được hình thành bằng cách nối M và F lại với nhau, T sẽ được truyền sang bên nhận, với r < k.

Với M (k bit), P (r + 1 bit), F (r bit), T (k + r bit), thủ tục tiến hành để xác định checksum F và tạo khung truyền như sau:

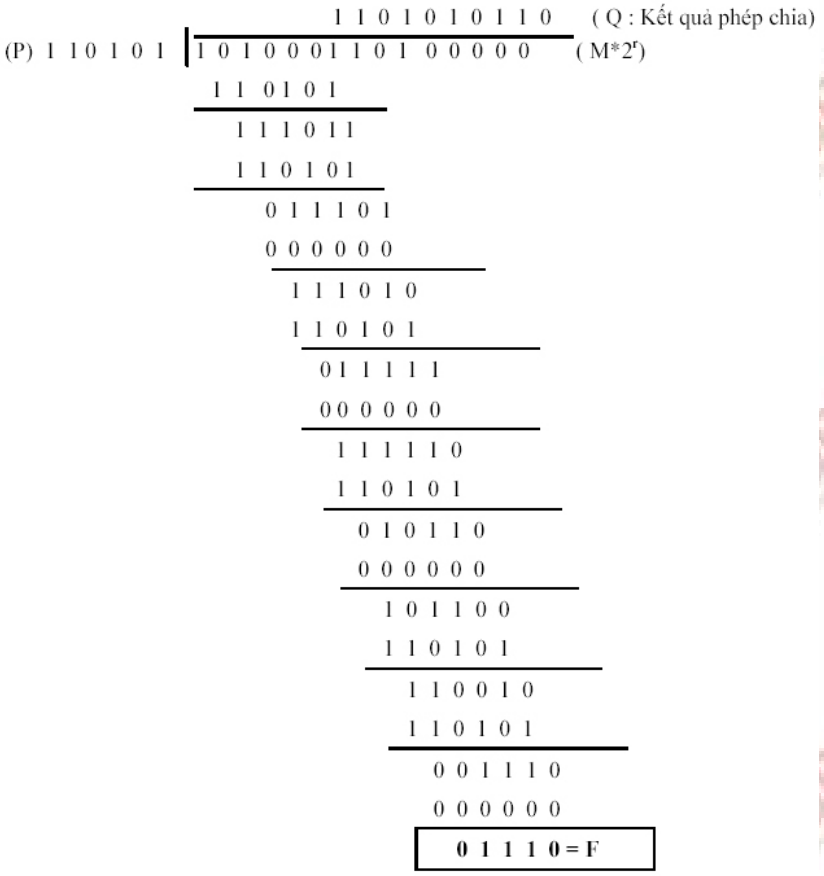
- Nối r bit 0 vào cuối M, hay thực hiện phép nhân M với 2^r.
- Dùng phép chia modulo 2 chia chuỗi $M \times 2^r$ cho P.
- Phần dư của phép chia sẽ được cộng với $M \times 2^r$ tạo thành khung T truyền đi.
- Trong đó P được chọn dài hơn F một bit. Cả hai bit cao nhất và thấp nhất phải là 1.

Ví dụ:

Giả sử có: M = 1010001101 (10 bit); P = 110101 (6 bit); FCS cần phải tính toán (5 bit).

Ta lần lượt thực hiện các bước sau:

- Tính $M \times 2^5$ = 101000110100000.
- Thực hiện phép chia modulo $M \times 2^5$ cho P như hình dưới.
- Được phần dư F = 01110.
- Tạo khung gửi đi là $T = M \times 2^r + F$ = 101000110101110.



Ngoài ra người ta còn có thể sử dụng phương pháp đa thức để biểu diễn phương pháp kiểm tra phần dư tuần hoàn. Trong phương pháp này người ta biểu diễn các chuỗi nhị phân dưới dạng những đa thức của biến x với các hệ số nhị phân. Các hệ số tương ứng với các bit trong chuỗi nhị phân cần biểu diễn.

Giả sử ta có M = 110011 và P = 11001, khi đó M và P sẽ được biểu diễn lại bằng 2 đa thức sau:

$$M(x) = x^5 + x^4 + x + 1$$

$$P(x) = x^4 + x^3 + 1$$

Những phép toán trên đa thức vẫn là modulo 2. Quá trình tính CRC được mô tả dưới dạng các biểu thức sau:

$$1.\ \frac{X^n M(X)}{P(X)} = Q(X) + \frac{R(X)}{P(X)}$$

$$2.\ T(X) = X^n M(X) + R(X)$$

Các version thường được sử dụng của P là:

$$CRC-12 = X^{12} + X^{11} + X^3 + X^2 + X + 1$$

$$CRC-16 = X^{16} + X^{15} + X^2 + 1$$

$$CRC-CCITT = X^{16} + X^{12} + X^5 + 1$$

$$CRC-32 = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

Ví dụ:

Cho: M = 1010001101, P = 110101

Ta có: r = 5

$$M(x) = x^9 + x^7 + x^3 + x^2 + 1$$

$$x^5 M(x) = x^{14} + x^{12} + x^8 + x^7 + x^5$$

$$P(x) = x^5 + x^4 + x^2 + 1$$

Thực hiện phép toán:

$$\frac{X^k M(X)}{P(X)} = Q(X) + \frac{F(X)}{P(X)}$$

$$\Rightarrow Q(x) = x^9 + x^8 + x^6 + x^2 + x^1$$

$$F(x) = x^3 + x^2 + x^1 <> 01110$$

→ Khung cần truyền đi là T = 101000110101110

Một số giao thức điều khiển lỗi (Error Control)

Phần kế tiếp chúng ta xem xét một số giao thức cơ bản được sử dụng nhiều trong việc điều khiển lỗi. Các giao thức này được xây dựng dựa trên các giả định sau:

- Chúng ta có máy tính A muốn gửi dữ liệu cho máy tính B.
- Luôn luôn có đủ dữ liệu cho máy tính A gửi đi.
- Các giao diện giao tiếp với tầng mạng và tầng vật lý đã được định nghĩa chuẩn.
- Bên nhận thông thường thực hiện việc chờ đợi một sự kiện nào đó phát sinh bằng cách gọi hàm wait_for_event().

Các giao thức được trình bày dưới dạng các chương trình viết bằng ngôn ngữ C. Chúng sử dụng các định nghĩa trong tập tin **protocol.h** có nội dung như sau:

```
#define MAX_PKT 1024 /*Kích thước tối đa của một gói tin*/
typedef enum{false, true} Boolean; /*Kiểu lôgic*/
typedef unsigned int seq_nr; /*Số thứ tự của khung gửi hoặc khung
                               báo nhận*/
typedef struct {unsigned char data[MAX_PKT]} pascket;
                                        /*Định nghĩa kiểu của gói tin*/
typedef enum{data, ack, nak} frame_kind; /*Các loại khung*/
typedef struct { /*Kiểu dữ liệu của khung;*/
frame_kind kind; //Loại khung
seq_nr seq; //Số thứ tự của khung gửi đi
seq_nr ack; // Số thứ tự của khung muốn báo nhận
packer info; //Thông tin gửi nhận,
} frame; //là gói tin nhận của tầng mạng
/*Chờ một sự kiện xuất hiện; trả vè kiểu của sự kiện*/
Void wait_for_event(event_type*event);
/*Nạp gói tin nhận được từ tầng mạng vào khung để gửi đi*/
Void from_network_layer(packet *p);
/*Chuyển dữ liệu từ khung nhận được cho tầng mạng*/
Void to_network_layer(packet *p);
```

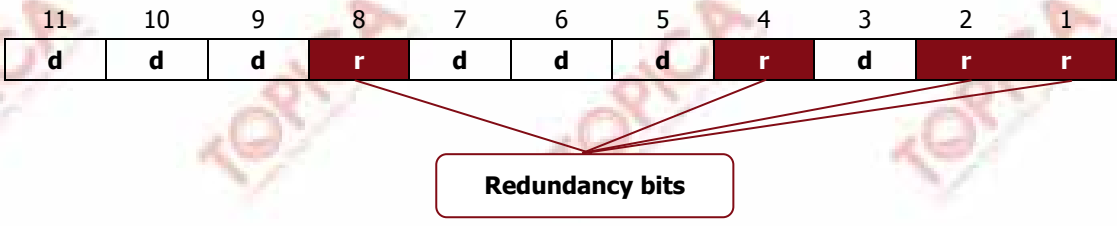
```
/*Nhận khung đến từ tầng vật lý và lưu nó vào khung r*/
Void from_physical_layer(frame *r);
/*Chuyển một khung xuống tầng vật lý để truyền đi*/
Void to_physical_layer(frame *s);
/*Khởi động đồng hồ và bật sự kiện quá thời hạn cho khung thứ k đang
gửi đi*/
Void start_timer(seq_nr k);
/*Dừng đồng hồ và tắt sự kiện quá thời hạn cho khung thứ k đang gửi đi*/
Void stop_timer(seq_nr k);
/*Khởi động đồng hồ phụ và bật sự kiện quá thời hạn cho khung thứ k
đang gửi đi*/
Void star_ack_timer(void);
/*Dừng đồng hồ phụ và tắt sự kiện quá thời gian cho khung phản hồi*/
Void stop_ack_timer(void);
/*Cho phép tầng mạng tạo sự kiện tầng mạng đã sẵn sàng*/
Void enable_network_layer(void);
/*Cấm tầng mạng tạo sự kiện tầng mạng đã sẵn sàng*/
Void disable_network_layer(void);
/*Macro để tăng giá trị K theo kiểu quay vòng*/
#define inc(k) if (k<MAX_SEQ) k=k+1; else k=0
```

3.3.3. Mã Hamming tự sửa một sai

Từ trước tới nay chúng ta đã kiểm tra số các bit cần thiết để có thể kiểm soát toàn bộ các trạng thái lỗi bit đơn có thể xảy ra trong khi truyền. Nhưng làm cách nào chúng ta có thể xử lý các bit này để khám phá ra trạng thái nào đã xảy ra? Một kỹ thuật được phát triển bởi R.W Hamming cung cấp một giải pháp thực tế.

Định vị trí các bit dư thừa – Positioning Redundancy Bits

Mã Hamming có thể được áp dụng cho các đơn vị dữ liệu có chiều dài bất kỳ và sử dụng mỗi quan hệ giữa dữ liệu và bit dư thừa được bàn luận ở phần trên. Ví dụ, một mã ASCII 7 bit cần phải có 4 bit dư thừa mà có thể được thêm vào cuối đơn vị dữ liệu hoặc đặt rải rác với các bit dữ liệu gốc. Trong hình 3.9, những bit này được đặt ở các vị trí 1, 2, 4 và 8 (các vị trí trong chuỗi tuần tự 11 bit là bình phương của 2). Để thấy rõ trong các ví dụ dưới đây, chúng ta cần xem các bit r1, r2, r4 và r8.



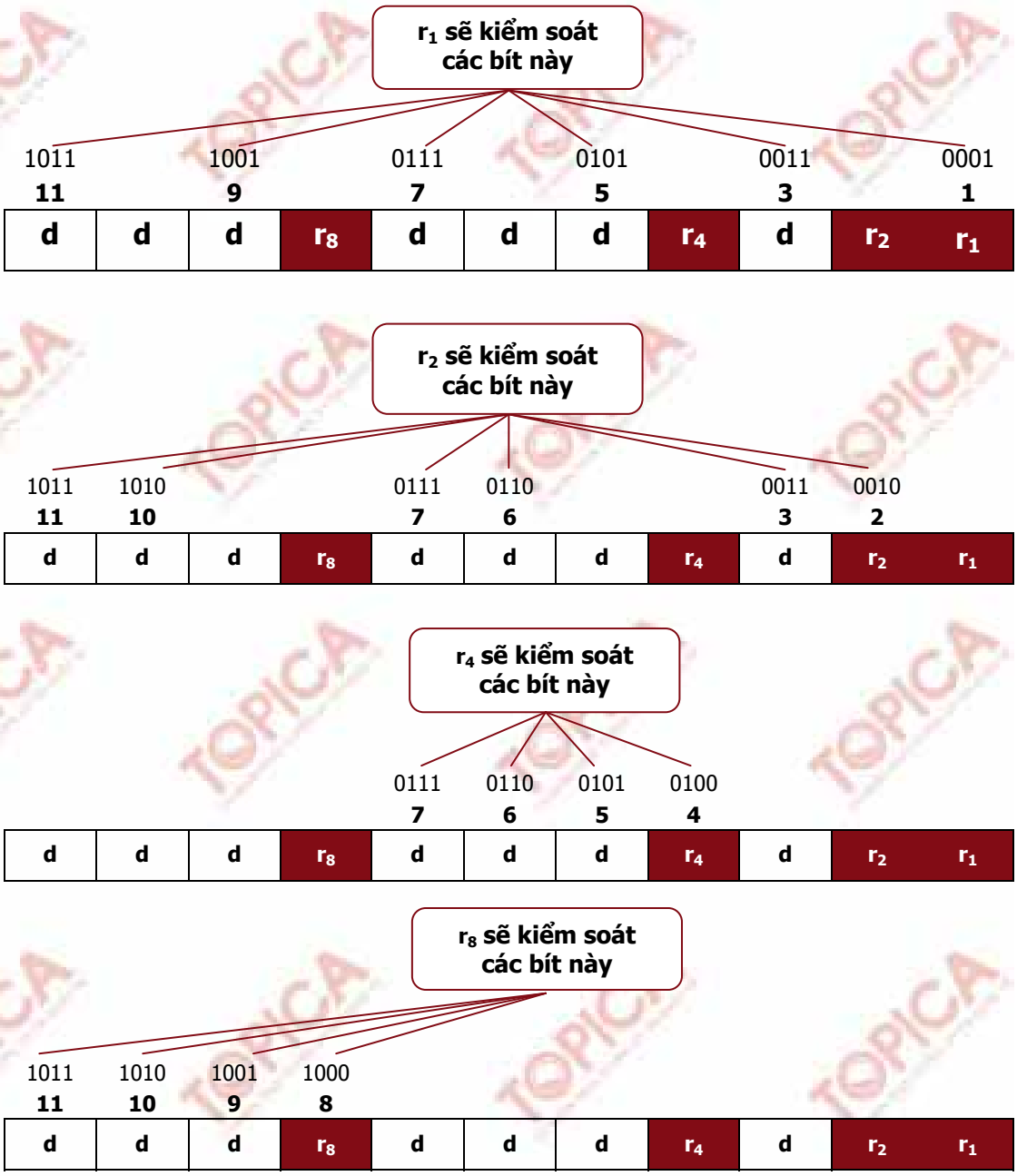
Hình 3.6: ***Các vị trí của các bit dư thừa trong mã Hamming***

Trong mã Hamming, mỗi r bit là bit VRC cho một tổ hợp các bit dữ liệu; r1 là bit VRC cho một tổ hợp các bit dữ liệu, r là bit VRC cho tổ hợp các bit dữ liệu khác, …. Các tổ hợp được sử dụng để tính toán từng giá trị r cho chuỗi 7 bit dữ liệu như sau:

- r1: các bit 1, 3, 5, 7, 9, 11
- r2: các bit 2, 3, 6, 7, 10, 11
- r4: các bit 4, 5, 6, 7
- r8: các bit 8, 9, 10, 11

Mỗi bit dữ liệu này có thể bao gồm một hoặc nhiều tính toán VRC. Trong các chuỗi trên, từng bit dữ liệu gốc được bao gồm trong ít nhất 2 tập hợp, trong khi r bit chỉ được bao gộp một lần.

Để xem mẫu theo giải pháp này, hãy xem biểu diễn nhị phân của từng vị trí bit. Bit r_1 được tính toán bằng cách sử dụng tất cả các vị trí mà biểu diễn nhị phân của nó bao gồm một bit 1 ở vị trí phải nhất. Bit r_2 được tính toán bằng cách sử dụng tất cả các vị trí bit với một bit 1 ở vị trí thứ 2…(xem hình 3.7).



Hình 3.7: ***Tính toán các bit dư thừa***

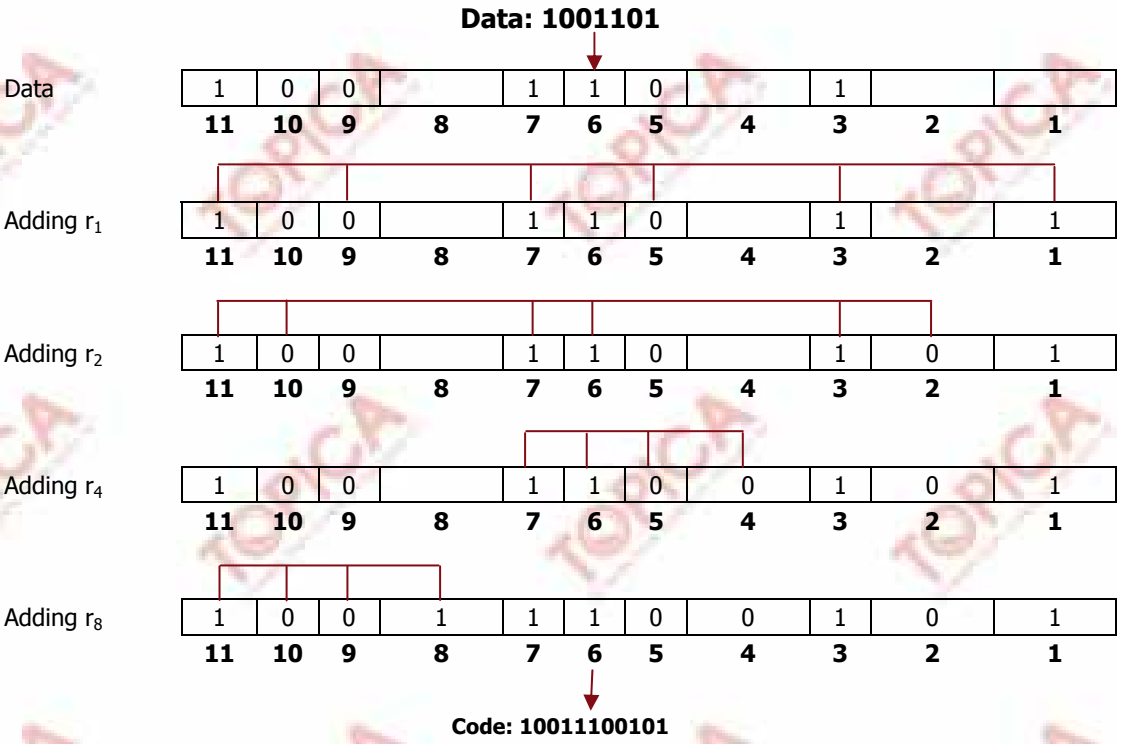
Tính toán các giá trị r

Hình 3.7 thể hiện mã Hamming áp dụng đối với một ký tự ASCII. Trong bước đầu tiên, chúng ta thay thế từng bit của ký tự gốc trong các vị trí thích hợp của nó bằng đơn vị dữ liệu có chiều dài 11 bit. Trong các bước tiếp sau, chúng ta tính toán các giá trị chẵn lẻ đối với nhiều tổ hợp bit. Giá trị chẵn lẻ cho từng tổ hợp đó là giá trị tương ứng với r bit.

Ví dụ, giá trị của r1 được tính toán để cung cấp tính chẵn lẻ đối với tổ hợp các bit 3, 5, 7, 9 và 11. Giá trị của r2 được tính toán để cung cấp tính chẵn lẻ với các bit 3, 6, 7, 10 và 11 … Mã 11 bit cuối cùng được gửi qua đường truyền dẫn.

Dò tìm và sửa lỗi

Giờ chúng ta hãy hình dung rằng theo thời gian khi dữ liệu được truyền tới bên nhận, số 7 bit đã bị thay đổi từ 1 thành 0.



Hình 3.8: ***Ví dụ về tính toán các bit dư thừa***

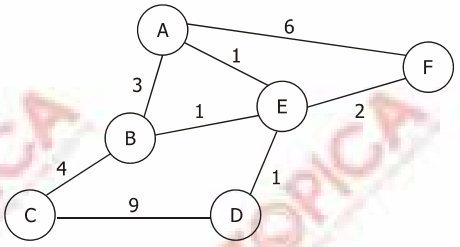
Bên nhận lấy dữ liệu từ đường truyền và tính toán lại 4 VRC mới sử dụng cùng tập bit đã được sử dụng bởi bên gửi cộng với bit chẵn lẻ tương ứng (r) đối với từng tập hợp (xem hình 3.8). Sau đó nó lắp ghép các giá trị chẵn lẻ đó vào một số nhị phân theo thứ tự của r (r8, r4, r2, r1). Trong ví dụ của chúng ta, bước này cho chúng ta số nhị phân 0111 (bằng 7 trong hệ mười), là vị trí chính xác của bit lỗi.

Một khi bit lỗi đã được xác định, bên nhận có thể đảo ngược giá trị của nó và thực hiện sửa lỗi đó.

3.4. Tìm đường đi trong mạng

3.4.1. Giới thiệu

Tìm đường về bản chất là một bài toán trong lý thuyết đồ thị (đồ hình: Graph). Hình 3.9 thể hiện một đồ thị biểu diễn cho một mạng.



Hình 3.9: ***Mạng được biểu diễn như một đồ thị***

Các nút trong đồ thị (được đánh dấu từ A đến F) có thể là các host, switch, router hoặc là các mạng con. Ở đây chúng ta tập trung vào một trường hợp các nút là các router. Các cạnh của đồ thị tương ứng với các đường kết nối mạng. Mỗi cạnh có một chi phí đính kèm, là thông số chỉ ra cái giá phải trả khi lưu thông trên kết nối mạng đó. Vấn đề cơ bản của việc tìm đường là tìm ra đường đi có chi phí thấp nhất giữa hai nút mạng bất kỳ, trong đó chi phí của đường đi được tính bằng tổng chi phí khi đi qua tất cả các cạnh làm thành đường đi đó. Nếu không có một đường đi giữa hai nút, thì độ dài đường đi giữa chúng được xem như bằng vô cùng.

3.4.2. Mục tiêu của giải thuật chọn đường

- Xác định đường đi nhanh chóng, chính xác.
- Khả năng thích nghi được với những thay đổi về hình trạng mạng.
- Khả năng thích nghi được với những thay đổi về tải đường truyền.
- Khả năng tránh được các kết nối bị tắt nghẽn tạm thời.
- Chi phí tính toán để tìm ra được đường đi là thấp nhất.

3.4.3. Phân loại giải thuật chọn đường

Giải thuật chọn đường có thể được phân thành những loại sau:

- Chọn đường tập trung (Centralized Routing): Trong mạng có một Trung tâm điều khiển mạng (Network Control Center) chịu trách nhiệm tính toán và cập nhật thông tin về đường đi đến tất cả các điểm khác nhau trên toàn mạng cho tất cả các router.
- Chọn đường phân tán (Distributed Routing): Trong hệ thống này, mỗi router phải tự tính toán tìm kiếm thông tin về các đường đi đến những điểm khác nhau trên mạng. Để làm được điều này, các router cần phải trao đổi thông tin qua lại với nhau.
- Chọn đường tĩnh (Static Routing): Trong giải thuật này, các router không thể tự cập nhật thông tin về đường đi khi hình trạng mạng thay đổi. Thông thường nhà quản trị mạng sẽ là người cập nhật thông tin về đường đi cho router.
- Chọn đường động (Dynamic Routing): Trong giải thuật này, các router sẽ tự động cập nhật lại thông tin về đường đi khi hình trạng mạng bị thay đổi.

3.4.4. Các giải thuật tìm đường đi tối ưu

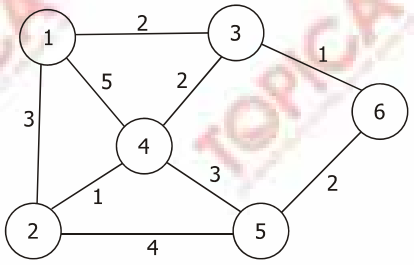
Đường đi tối ưu từ A đến B là đường đi "ngắn" nhất trong số các đường đi có thể. Tuy nhiên khái niệm "ngắn" nhất có thể được hiểu theo nhiều ý nghĩa khác nhau tùy thuộc vào đơn vị dùng để đo chiều dài đường đi. Đối với các router, các đại lượng sau có thể được sử dụng để đo độ dài đường đi:

- Số lượng các router trung gian phải đi qua (HOP).
- Độ trì hoãn trung bình của các gói tin.
- Cước phí truyền tin.

Mỗi giải thuật chọn đường trước tiên phải chọn cho mình đơn vị đo chiều dài đường đi.

Để xác định được đường đi tối ưu, các giải thuật chọn đường sử dụng phương pháp đồ thị để tính toán. Trước tiên, nó mô hình hóa hình trạng mạng thành một đồ thị có các đặc điểm như sau:

- Nút là các router.
- Cạnh nối liền 2 nút là đường truyền nối hai router.
- Trên mỗi cạnh có giá đó là chiều dài đường đi giữa 2 router thông qua đường truyền nối hai router.
- Chiều dài đường đi từ nút A đến nút B là tổng tất cả các giá của các cạnh nằm trên đường đi. Nếu không có đường đi giữa 2 router thì xem như giá là vô cùng.



Hình 3.10: ***Mô hình hóa mạng thành đồ thị***

Trên đồ thị này sẽ thực hiện việc tính toán tìm đường đi ngắn nhất.

3.4.4.1. Giải thuật tìm đường đi ngắn nhất Dijkstra

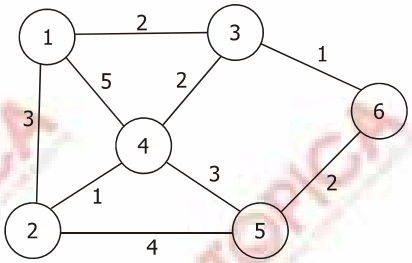
Mục đích là để tìm đường đi ngắn nhất từ một nút cho trước trên đồ thị đến các nút còn lại trên mạng.

Giải thuật được mô tả như sau:

- Gọi:
  - S: là nút nguồn cho trước.
  - N: là tập hợp tất cả các nút đã xác định được đường đi ngắn nhất từ S.
  - Di: là độ dài đường đi ngắn nhất từ nút nguồn S đến nút i.
  - aij: là giá của cạnh nối trực tiếp nút i với nút j, sẽ là ∞ nếu không có cạnh nối trực tiếp giữa i và j.
  - Pj: là nút cha của nút j.
- Bước 1: Khởi tạo
  - N = {S}; Ds = 0.
  - Với i ≠ S: Di = aij, Pj = S.
- Bước 2: Tìm nút gần nhất kế tiếp
  - Tìm nút i ∉ N thoả Di = min (Dj) với j ∉ N.
  - Thêm nút i vào N.
  - Nếu N chứa tất cả các nút của đồ thị thì dừng. Ngược lại sang Bước 3.
- Bước 3: Tính lại giá đường đi nhỏ nhất
  - Với mỗi nút j ∉ N: Tính lại Dj = min { Dj, Di + aij }; Pj = i.
  - Trở lại Bước 2.

Ví dụ:

Cho mạng có hình trạng như đồ thị hình 3.11:



Hình 3.11: ***Đồ thị mạng***

Tìm đường đi ngắn nhất từ nút 1 đến các nút còn lại.

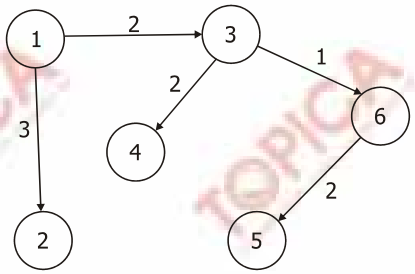
Hướng dẫn:

Áp dụng giải thuật ta có: S = 1.

Các bước thực hiện được mô tả như sau:

Lần lặp	N	D 2	D 3	D 4	D 5	D 6	P 2	P 3	P 4	P 5	P 6
Khởi tạo	{1}	3	2	5	∞	∞	1	1	1	1	1
1	{1, 3}	3	2	4	∞	3	1	1	3	1	3
2	{1, 3, 2}	3		4	7	3	1		3	2	3
3	{1, 3, 2, 6}			4	5	3			3	6	3
4	{1, 3, 2, 6, 4}			4	5				3	6	
5	{1, 3, 2, 6, 4, 5}				5					6	

Từ kết quả trên ta vẽ được cây có đường đi ngắn nhất từ nút số 1 đến các nút còn lại như hình 3.12. Từ cây đường đi ngắn nhất này, ta xác định được rằng: để đi đến các router 4, 5, 6, bước kế tiếp router 1 cần gửi gói tin đến là router số 3 (Next Hop). Chú ý, đường ngắn nhất này chỉ đúng theo hướng từ nút số 1 về các nút còn lại và chỉ đúng cho nút số 1 mà thôi.



Hình 3.12: ***Đường đi ngắn nhất từ nút số 1***

Thông thường giải thuật Dijkstra được sử dụng theo mô hình chọn đường tập trung. Trong đó, Trung tâm điều khiển mạng sẽ tìm cây đường đi ngắn nhất cho từng router trên mạng và từ đó xây dựng bảng chọn đường tối ưu cho tất cả các router.

3.4.4.2. Giải thuật chọn đường tối ưu Ford-Fulkerson

Mục đích của giải thuật này là để tìm đường đi ngắn nhất từ tất cả các nút đến một nút đích cho trước trên mạng.

Giải thuật được mô tả như sau:

- Gọi:
  - d: là nút đích cho trước.
  - Di: là chiều dài đường đi ngắn nhất từ nút i đến nút d.
  - Ci: là nút con của nút i.
- Bước 1: Khởi tạo
  - Gán Dd = 0.
  - Với $\forall i \neq d$: gán Di = ∞; Ci = -1.
- Bước 2: Cập nhật giá đường đi ngắn nhất từ nút i đến nút d
  - Di= min{ aij + Dj } với $\forall i \neq j$ => Ci = j.
  - Lặp lại cho đến khi không còn Di nào bị thay đổi giá trị.

Ví dụ:

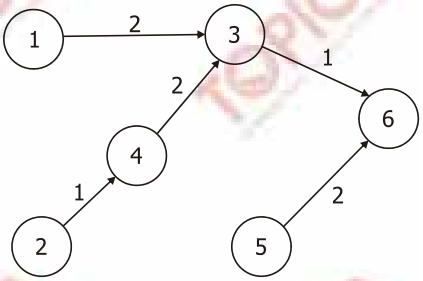
Trong ví dụ trên áp dụng tìm đường đi ngắn nhất từ nút khác trên đồ thị đến nút 6.

Áp dụng giải thuật ta có: d = 6.

Các bước thực hiện được mô tả như sau:

Lần lặp	D 1	D 2	D 3	D 4	D 5	C 1	C 2	C 3	C 4	C 5
Khởi tạo	∞	∞	∞	∞	∞	-1	-1	-1	-1	-1
1	∞	∞	1	3	2	-1	-1	6	3	6
2	3	4	1	3	2	3	4	6	3	6
3	3	4	1	3	2	3	4	6	3	6

Từ kết quả trên ta vẽ lại được cây đường đi ngắn nhất từ các nút khác nhau về nút đích số 6 như hình 3.13. Cây này cho phép xác định đường đi tối ưu từ những nút khác nhau về nút số 6. Chẳng hạn tại nút 1, để đi về nút số 6 thì bước kế tiếp phải đi là nút số 3. Tương tự, tại nút 2, để đi về nút số 6 thì bước kế tiếp phải đi là nút số 4.

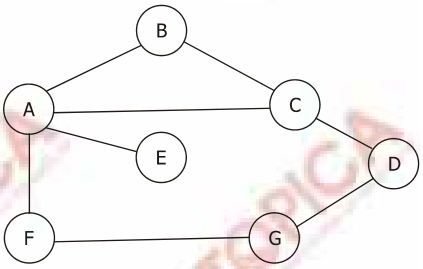


Hình 3.13: ***Đường đi ngắn nhất từ nút số 1***

Giải thuật này được sử dụng theo mô hình phân tán. Ở đó mỗi router sẽ tự tính toán, tìm cây có đường đi ngắn nhất từ các nút khác về nó. Từ đó suy ra đường đi tối ưu cho các nút khác đến nó và gửi các đường đi này đến từng nút trên mạng.

3.4.5. Giải pháp tìm đường Vector Khoảng cách (Distance Vector)

Ý tưởng của Distance - Vector như sau: Mỗi nút thiết lập một mảng một chiều (Vector) chứa khoảng cách (chi phí) từ nó đến tất cả các nút còn lại và sau đó phát vector này đến tất cả các nút láng giềng của nó. Giả thiết đầu tiên trong Distance-Vector là: mỗi nút phải biết được chi phí của các đường nối từ nó đến tất cả các nút láng giềng có đường kết nối trực tiếp với nó. Một kết nối bị đứt (down) sẽ được gán cho chi phí có giá trị vô cùng.



Hình 3.14: ***Một mạng làm ví dụ trong giải thuật Distance-Vector***

Để xem xét giải thuật tìm đường Distance-Vector hoạt động như thế nào, cách dễ nhất là xem xét đồ thị được cho như trong hình vẽ 3.14.

Trong ví dụ này, chi phí trên mỗi đường nối đều được đặt là 1. Chúng ta có thể biểu diễn sự hiểu biết của các nút về khoảng cách từ chúng đến các nút khác như trong bảng sau:

Thông tin được lưu tại các nút	Khoảng cách đến nút						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Chúng ta có thể xem mỗi một hàng trong bảng trên như là một danh sách các khoảng cách từ một nút đến tất cả các nút khác. Khởi đầu, mỗi nút đặt giá trị 1 cho đường kết nối đến các nút láng giềng kề nó, ∞ cho các đường nối đến tất cả các nút còn lại. Do đó, lúc đầu A tin rằng nó có thể tìm đến B qua một bước nhảy (hop) và rằng nó không thể đi đến D được. Bảng tìm đường lưu tại A thể hiện những niềm tin mà A có được, ngoài ra còn lưu thêm nút kế tiếp mà A cần phải đi ra để đến một nút nào đó. Khởi đầu, bảng tìm đường của nút A trông giống như trong bảng sau:

Đích (Destination)	Chi phí (Cost)	Nút kế tiếp (Next Hop)
B	1	B
C	1	C
D	∞	–
E	1	E
F	1	F
G	∞	–

Bước kế tiếp trong giải thuật tìm đường Distance-Vector là: mỗi nút sẽ gửi một thông điệp đến các láng giềng liền kề nó, trong thông điệp đó chứa danh sách các khoảng cách mà cá nhân nút tính được. Ví dụ, nút F bảo nút A rằng F có thể đi đến nút G với chi phí là 1; A cũng biết được rằng nó có thể đến F với chi phí là 1, vì thế A cộng các chi phí lại thành chi phí đi đến G là 2 thông qua F. Tổng chi phí là 2 này nhỏ hơn chi phí vô cùng lúc đầu, do đó A ghi lại nó có thể đi đến G thông qua F với chi phí là 2. Tương tự, A học được từ C rằng, nó có thể đi đến D thông qua C với chi phí là 2, và chi phí này nhỏ hơn chi phí cũ là vô cùng. Cùng lúc A cũng học từ C rằng, nó có thể đi đến B thông qua C với chi phí là 2, nhưng chi phí này lại lớn hơn chi phí cũ là 1, vì thế thông tin mới này bị bỏ qua.

Tại thời điểm này, A có thể cập nhật lại bảng chọn đường của nó với chi phí và nút ra kế tiếp để có thể đi đến tất cả các nút khác trong mạng. Kết quả được cho trong bảng sau:

Đích (Destination)	Chi phí (Cost)	Nút kế tiếp (Next Hop)
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Nếu không có sự thay đổi về hình trạng mạng nào, chỉ cần vài cuộc trao đổi thông tin tìm đường giữa các nút trong mạng thì mọi nút đều có được thông tin tìm đường hoàn hảo. Quá trình đem thông tin tìm đường nhất quán đến mọi nút trong mạng được gọi là sự hội tụ (Convergence). Bảng dưới chỉ ra thông tin về chi phí cuối cùng từ một nút đến các nút khác trong mạng khi quá trình tìm đường đã hội tụ.

Thông tin được lưu tại các nút	Khoảng cách đến nút						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Nét đẹp của loại giải thuật phân tán trên nằm ở chỗ nó cho phép tất cả các nút đạt được thông tin tìm đường mà không cần có sự hiện diện của bộ điều khiển trung tâm nào.

Còn có vài chi tiết làm cho giải thuật Distance-Vector hoàn hảo hơn. Thứ nhất, chú ý rằng có hai tình huống khác nhau mà tại đó một nút quyết định gửi thông tin tìm đường của mình cho các nút láng giềng kề bên. Tình huống đầu tiên là sự cập nhật theo chu kỳ (Periodic Update). Trong tình huống này, mỗi nút tự động gửi bản cập nhật thường xuyên, ngay cả khi không có thay đổi gì trong đó cả. Điều này giúp cho các nút khác biết được nút hiện tại đang còn sống. Vả lại việc cập nhật thường xuyên còn giúp cho các nút láng giềng có thể có được thông tin tìm đường nhanh chóng trong trường hợp thông tin của chúng bị mất. Tần số phát thông tin tìm đường đi có thể khác nhau tùy vào giải thuật, chúng thường có giá trị từ vài giây đến vài phút. Tình huống thứ hai gọi là sự cập nhật do bị kích hoạt (Triggered Update). Tình huống này xảy ra mỗi khi có sự thay đổi thông tin trong bảng tìm đường của nút. Nghĩa là mỗi khi bảng tìm đường có sự thay đổi, nút sẽ gửi bản cập nhật này cho các láng giềng của mình.

Bây giờ ta xem xét điều gì xảy ra khi một đường kết nối hay một nút bị hỏng. Những nút đầu tiên phát hiện ra điều này sẽ gửi thông tin tìm đường mới cho láng giềng của chúng ngay, và thông thường hệ thống sẽ ổn định với tình trạng mới một cách nhanh chóng. Còn đối với câu hỏi làm sao nút phát hiện ra sự cố, có nhiều câu trả lời khác nhau. Cách tiếp cận thứ nhất là: một nút liên tục kiểm tra đường nối tới các nút láng giềng khác bằng cách gửi các gói tin điều khiển tới chúng và kiểm tra xem nó có nhận được các gói tin trả lời hay không. Cách tiếp cận khác là: một nút phát hiện ra một đường kết nối (hay nút ở đầu kia của đường nối) gặp sự cố khi nó không nhận được thông tin tìm đường một cách định kỳ từ láng giềng của nó.

Ví dụ: Xem xét điều gì sẽ xảy ra khi F phát hiện ra đường nối từ nó đến G bị hỏng. Đầu tiên, F đặt chi phí của đường nối từ nó đến A thành vô cùng và gửi thông tin này đến A. Do A đã biết là cần 2 bước để đi từ nó đến G thông qua F, A sẽ đặt lại chi phí từ nó đến G là vô cùng. Tuy nhiên, với bản cập nhật kế tiếp từ C, A phát hiện ra rằng có một đường đi dài 2 hops từ C đến G, do đó nó sẽ cập nhật lại đường đi từ nó đến G dài 3 hops thông qua C. Và khi A quảng cáo thông tin này cho F, F lại cập nhật lại đường đi dài 4 hops đến G thông qua A.

Không may là một số tình huống phát sinh lỗi khác lại làm cho mạng mất ổn định nghiêm trọng. Giả sử kết nối từ A đến E bị đứt. Trong những chu kỳ cập nhật sau, A thông báo đường đi từ nó đến E dài vô cùng, nhưng B và C lại quảng cáo đường đi từ chúng đến E dài 2 hops. Nếu các bản cập nhật được định thời gian để phát cùng lúc, B sẽ sửa lại độ dài đường đi từ nó đến E là 3 thông qua C, C sửa lại độ dài đường đi từ nó đến E là 3 thông qua B. Sau đó A lại nghe B và C quảng cáo độ dài đường đi từ

chúng đến E là 3 và giả sử A chọn B là nút kế tiếp để đi đến E, nó sẽ cập nhật lại độ dài đoạn đường là 4. Đến chu kỳ kế tiếp, B nghe C nói độ dài từ C đến E là 3 nên cập nhật lại độ dài đường đi từ B đến E là 4 thông qua C, C thì làm ngược lại: sửa lại con đường từ nó đến E là 4 thông qua B. Rồi lại đến lượt A nghe B sửa lại độ dài từ A đến E là 5 thông qua B. Sự thể sẽ tiếp diễn cho đến khi các độ dài tăng đến một số có thể coi là vô cùng. Nhưng tại thời điểm này, không nút nào biết là E không thể đến được, và các bảng tìm đường trong mạng luôn không ổn định. Tình huống này được gọi là vấn đề "đếm tới vô cùng" (count-to-infinity problem).

Có vài giải pháp giải quyết một phần vấn đề "đếm tới vô cùng". Giải pháp thứ nhất là dùng một số khá nhỏ để coi như gần bằng vô cùng. Ví dụ như chúng ta có thể quyết định số lượng bước nhảy (hop) tối đa để đi qua một mạng là không quá 16, và do đó ta chọn 16 là số gần bằng vô cùng. Con số này ít ra cũng giới hạn được thời gian mà các nút có thể phải bỏ ra để đếm tới vô cùng. Tuy nhiên giải pháp này có thể gặp vấn đề nếu một số nút mạng được chia tách và mạng có thể cần nhiều hơn 16 bước nhảy để duyệt hết nó.

Một kỹ thuật khác dùng để cải thiện thời gian và ổn định hóa mạng được gọi là kỹ thật "chia tầm nhìn" (Split Horizon). Ý tưởng là: khi một nút gửi một bảng cập nhật tìm đường cho các láng giềng của nó, nó sẽ không gửi những thông tin tìm đường mà nó đã nhận từ một nút láng giềng ngược lại chính nút láng giềng đó. Ví dụ như nếu B có một đường đi (E, 2, A) trong bảng vạch đường của nó, B chắc hẳn phải biết rằng nó học con đường này từ A, vì thế mỗi khi B gửi thông tin cập nhật cho A nó sẽ không gửi con đường (E, 2) trong đó. Tuy nhiên giải pháp này chỉ tốt khi nó xoay quanh 2 nút mà thôi.

3.4.6. Giải pháp chọn đường "Trạng thái kết nối" (Link State)

Tìm đường kiểu Link-state là một ví dụ thứ hai trong họ giao thức tìm đường bên trong một miền. Giả thiết đầu tiên trong Link-state cũng khá giống trong Distance-vector: Mỗi nút được giả định có khả năng tìm ra trạng thái của đường nối nó đến các nút láng giềng và chi phí trên mỗi đường nối đó. Nhắc lại lần nữa: chúng ta muốn cung cấp cho mỗi nút đủ thông tin để cho phép nó tìm ra đường đi có chi phí thấp nhất đến bất kỳ đích nào. Ý tưởng nền tảng đằng sau các giao thức kiểu Link-state là rất đơn giản: Mọi nút đều biết đường đi đến các nút láng giềng kề bên chúng và nếu chúng ta đảm bảo rằng tổng các kiến thức này được phân phối cho mọi nút thì mỗi nút sẽ có đủ hiểu biết về mạng để dựng lên một bản đồ hoàn chỉnh của mạng. Giải thuật Link - State dựa trên hai kỹ thuật: sự phân phối một cách tin cậy thông tin về trạng thái các đường kết nối; và sự tính toán các đường đi từ kiến thức tổng hợp về trạng thái các đường kết nối.

3.4.6.1. Làm ngập một cách tin cậy (Reliable Flooding)

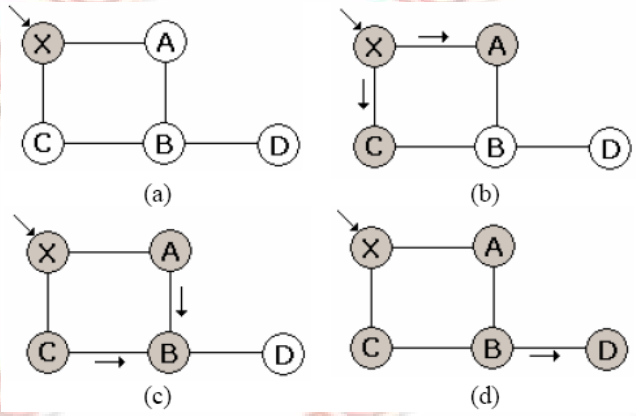
"Làm ngập" là quá trình thực hiện cam kết: "tất cả các nút tham gia vào giao thức tìm đường đều nhận được thông tin về trạng thái kết nối từ tất cả các nút khác". Như khái niệm "làm ngập" ám chỉ, ý tưởng cơ sở của Link-state là cho một nút phát thông tin về trạng thái kết nối của nó với mọi nút láng giềng liền kề, đến lượt mỗi nút nhận được thông tin trên lại chuyển phát thông tin đó ra các nút láng giềng của nó. Tiến trình này cứ tiếp diễn cho đến khi thông tin đến được mọi nút trong mạng. Cụ thể hơn, mỗi nút tạo ra gói tin cập nhật, còn được gọi là gói tin trạng thái kết nối (Link – State Packet – LSP), chứa những thông tin sau:

- ID của nút đã tạo ra LSP.
- Một danh sách các nút láng giềng có đường nối trực tiếp tới nút đó, cùng với chi phí của đường nối đến mỗi nút.
- Một số thứ tự.
- Thời gian sống (Time To Live) của gói tin này.

Hai mục đầu là cần thiết cho việc tính toán chọn đường; hai mục sau cùng được sử dụng để giúp cho quá trình làm ngập thật chắc. Tính tin cậy ở đây đòi hỏi việc đảm bảo các nút trong mạng có được thông tin có phiên bản mới nhất, do có nhiều LSP trái ngược nhau từ một nút được phát lên mạng. Đảm bảo việc làm ngập có thể tin cậy được là một việc khó (Ví dụ, một phiên bản cũ của giao thức tìm đường link-state trong ARPANET đã làm cho mạng này bị tê liệt vào năm 1981). Việc làm ngập được thực hiện như sau: Đầu tiên, việc truyền các LSP giữa các nút kề nhau được bảo đảm tính tin cậy bằng cách sử dụng cơ chế báo nhận (Acknowledgement) và làm lại khi bị lỗi (Retransmission) giống như ở tầng liên kết dữ liệu. Tuy nhiên, cần thực hiện thêm một số bước để đảm bảo việc phát một LSP từ một nút đến tất cả các nút khác trong mạng là đáng tin cậy.

Giả sử nút X nhận được một phiên bản LSP có nguồn gốc từ nút Y nào đó. Chú ý rằng nút Y có thể là bất kỳ router nào ở trong cùng một miền với X. X kiểm tra xem nó đã có bất kỳ phiên bản LSP nào từ Y không. Nếu không, nó sẽ lưu LSP này. Nếu có, X sẽ so sánh hai số thứ tự trong hai LSP. Nếu LSP mới đến có số thứ tự lớn hơn số thứ tự của LSP có sẵn, X cho rằng LSP mới đến là mới hơn, và do đó X sẽ thay LSP cũ bằng phiên bản mới này. Ngược lại, với một số thứ tự nhỏ hơn (hoặc bằng), LSP mới đến sẽ bị coi là cũ hơn cái đang có sẵn (hoặc ít ra là không mới hơn), và vì thế nó sẽ bị bỏ qua, không cần phải làm gì thêm. Nếu LSP mới đến là cái mới hơn, nút X sẽ gửi một phiên bản của LSP này ra tất cả các nút láng giềng liền kề nó ngoại trừ nút láng giềng vừa gửi cho nó phiên bản LSP mới vừa đề cập. Đến phiên các nút láng giềng của X lại xoay qua phát tán LSP mới này sang các nút láng giềng khác. Việc "LSP không được gửi ngược lại nút vừa phát ra nó" sẽ giúp dẫn đến điểm dừng của quá trình phát tán LSP này. Sự phát tán dây chuyền có điểm dừng này sẽ đảm bảo cho việc đem phiên bản LSP mới nhất đến tất cả các nút trong mạng.

Hình sau thể hiện một LSP được dùng làm ngập một mạng nhỏ. Hình (a) thể hiện X nhận được một LSP mới; (b) X đẩy LSP mới ra A và C; (c) A và C đẩy LSP qua B; (d) B đẩy LSP qua D và quá trình làm ngập kết thúc.



Hình 3.18: ***Việc làm ngập mạng với các gói tin LSP***

Cũng giống như trong giải thuật Distance-Vector, sẽ có hai tình huống mà một nút quyết định gửi LSP đến các nút láng giềng: gửi một cách định kỳ hoặc gửi do bị kích hoạt.

Một trong những ưu tiên hàng đầu của cơ chế làm ngập (Flooding) là phải đảm bảo đem thông tin mới nhất đến mọi nút trong mạng càng nhanh càng tốt và các thông tin cũ phải được rút ra không cho lưu thông trên mạng nữa. Thêm nữa, rất là lý tưởng nếu ta có thể giảm thiểu lượng thông tin tìm đường lưu chuyển trên mạng – một kiểu phí tổn theo cách nhìn của nhiều người.

Một phương pháp cần thiết để giảm thiểu phí tổn dành cho việc tìm đường là tránh gửi các LSP trừ trường hợp hết sức cần thiết. Điều này có thể thực hiện được bằng cách sử dụng các bộ định thời (Timer) có giá trị rất lớn – thường là kéo dài hàng giờ - dùng để định kỳ phát các LSP.

Còn để đảm bảo rằng thông tin cũ phải được thay thế bởi thông tin mới, các LSP sẽ mang số thứ tự. Mỗi khi một nút phát LSP mới, nó sẽ tăng số thứ tự lên 1. Không giống như hầu hết các giao thức khác, số thứ tự trong LSP sẽ không được đếm xoay vòng (Modulo), vì thế trường chứa số này phải đủ lớn (ví dụ như 64 bit). Nếu một nút bị chết (Down) và sau đó được khởi động lại, nó sẽ khởi động trường số thứ tự lại bằng 0. Nếu một nút bị chết quá lâu, tất cả các LSP của nút đó sẽ bị mãn kỳ (timed out); ngoài ra, nếu cuối cùng nút này lại nhận được LSP của chính nó với số thứ tự lớn hơn bản gốc, nút có thể lấy số lớn hơn làm số thứ tự mới.

Các LSP cũng mang theo thời gian sống của nó (Time to live - TTL). Điều này dùng để đảm bảo các LSP cũ rút cuộc cũng bị xóa khỏi mạng. Một nút luôn luôn giảm trường TTL của một LSP mới đến nó đi 1 trước khi chuyển LSP này ra các nút láng giềng. Khi trường TTL còn 0, nút phát lại LSP với TTL = 0, điều đó sẽ được thông dịch bởi tất cả các nút trong mạng như là tín hiệu cho phép xóa LSP đó.

3.4.6.2. Tính toán chọn đường trong Link State

Khi một nút có một phiên bản LSP từ tất cả các nút khác trong mạng, nó đã có thể tính toán ra bản đồ hoàn chỉnh cho hình thái của mạng, và từ bản đồ này nút có thể quyết định con đường tốt nhất đến tất cả các nút còn lại trong mạng. Giải pháp chọn đường chính là giải thuật tìm đường đi ngắn nhất Dijkstra.

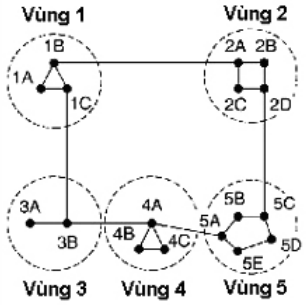
3.4.7. Tìm đường phân cấp (Hierarchical Routing)

Khi mạng tăng kích thước, kích thước bảng tìm đường của các router tăng theo. Không chỉ bộ nhớ của router bị tiêu hao quá nhiều cho việc trữ các bảng tìm đường, mà CPU còn phải tốn nhiều thời gian để quét bộ nhớ và cũng cần nhiều băng thông hơn để truyền những thông tin chọn đường này. Rồi cũng sẽ đến lúc mạng máy tính phát triển đến mức không một router nào có đủ khả năng trữ một đầu mục thông tin về một router khác, vì thế việc tìm đường phải phát triển theo đường hướng khác: tìm đường phân cấp.

Khi việc tìm đường phân cấp được áp dụng, các router được chia thành những vùng (domain). Trong mỗi vùng, mỗi router biết cách tìm đường cho các gói tin đi đến được mọi đích trong nội vùng của nó, nhưng không biết gì về cấu trúc bên trong của các vùng khác. Khi nhiều vùng được kết nối với nhau, đương nhiên mỗi vùng được công nhận tính độc lập để giải phóng các router trong các vùng đó khỏi việc phải tìm hiểu hình trạng của các vùng khác.

Với những mạng thật lớn, kiến trúc phân cấp hai mức có thể sẽ không đủ; có thể cần phải nhóm các vùng lại thành liên vùng, nhóm các liên vùng thành khu vực...

Hình 3.19 thể hiện một mạng được tìm đường phân cấp gồm hai mức có năm vùng. Bảng tìm đường đầy đủ của router A gồm có 17 mục từ như trong hình 3.19(b). Khi tìm đường được thực hiện theo kiểu phân cấp, bảng tìm đường của A giống như bảng hình 3.19(c), có mọi mục từ chỉ đến các router cục bộ giống như trước, tuy nhiên các mục từ chỉ đến các vùng khác lại được cô đặc lại thành một router. Do tỉ lệ các router trong các vùng tăng, vì thế cách làm này giúp rút ngắn bảng tìm đường.



Bảng vạch đường đầy đủ của nút 1A

Đích	Lối ra	Chi phí
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Bảng vạch đường phân cấp của host 1A

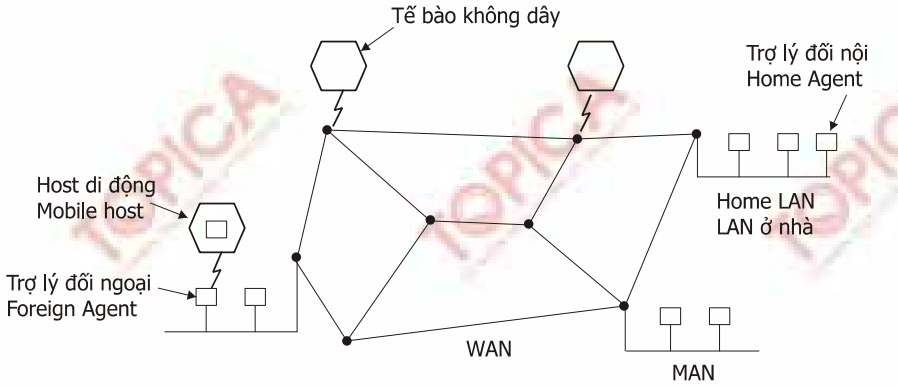
Đích	Lối ra	Chi phí
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Hình 3.16: ***Mạng và bảng tìm đường của Rounter***

3.4.8. Tìm đường trong mạng di động

Ngày nay, hàng triệu người đang sở hữu máy tính xách tay, và thông thường họ muốn đọc email cũng như truy xuất các hệ thống tập tin cho dù họ đang ở bất kỳ nơi nào trên thế giới. Việc sử dụng các host di động này dẫn đến một vấn đề phức tạp mới: để tìm đường cho gói tin đến host di động, trước tiên phải tìm ra nó đã. Chủ đề về tích hợp các host di động lại thành một mạng là tương đối mới, tuy vậy trong phần này chúng ta sẽ phác thảo ra một số vấn đề phát sinh và chỉ ra các giải pháp khả thi.



Hình 3.17: ***Mô hình mạng***

Mô hình mạng mà các nhà thiết kế thường sử dụng được chỉ ra trong hình 3.17. Ở đây chúng ta có một mạng WAN bao gồm vài router và host. Mạng WAN được dùng để kết nối các mạng LAN, MAN, các tế bào mạng không dây (Wireless Cell). Các host không bao giờ di chuyển được gọi là cố định, chúng được nối vào mạng bởi đường cáp đồng

trục hoặc cáp quang. Ngược lại, chúng ta sẽ phân biệt hai loại host khác: loại di cư (Migratory Host) và loại chuyển vùng (Roaming Host). Loại host di cư về bản chất là host cố định, nhưng chúng thỉnh thoảng lại chuyển từ địa bàn (Site) này đến địa bàn mạng kia, và chúng chỉ có thể sử dụng mạng mới khi được kết nối vật lý vào đấy. Loại host chuyển vùng thực chất vừa chạy vừa tính toán, nó muốn duy trì các kết nối mạng ngay cả khi đang di chuyển. Chúng ta sẽ sử dụng thuật ngữ "host di động" để ám chỉ hai loại di động vừa nói đến, tức là chúng đã đi khỏi nhưng lại muốn duy trì liên lạc về nhà. Tất cả các host được giả sử đều có vị trí mạng nhà (Home Location) và vị trí này không bao giờ thay đổi. Các host cũng có địa chỉ lâu dài tại nhà (Home Address) và địa chỉ này có thể được dùng để xác định vị trí mạng nhà của nó, cũng giống như số điện thoại (0) 84-4-38692137 chỉ ra số đó ở Việt Nam (mã 084), thành phố Hà Nội (mã 04). Mục tiêu của việc tìm đường trong hệ thống có các host di động là phải đảm bảo có thể gửi được gói tin đến host di động sử dụng địa chỉ tại nhà của nó và làm cho các gói tin đến được host di động một cách hiệu quả cho dù host này có ở đâu đi nữa. Trong mô hình ở hình trên, WAN được chia thành các đơn vị nhỏ, ở đây chúng ta gọi là khu vực (Area), thường là LAN hoặc tế bào mạng không dây. Mỗi khu vực có một hoặc nhiều trợ lý đối ngoại (Foreign Agent - FA) – đó là những tiến trình làm nhiệm vụ theo dõi các host khách đang viếng thăm khu vực của mình. Thêm vào đó, mỗi khu vực còn có một đại lý nội bộ (Home Agent - HA), làm nhiệm vụ theo dõi những host có nhà nằm trong khu vực nhưng hiện đang viếng thăm khu vực khác. Khi một host đi vào một khu vực mới (có thể là host này muốn thường trú trong mạng LAN mới hoặc chỉ đi ngang cell này thôi), nó phải đăng ký với trợ lý đối ngoại ở đó. Thủ tục đăng ký diễn ra như sau:

1) Theo chu kỳ, mỗi trợ lý đối ngoại sẽ phát ra những thông điệp thông báo sự hiện diện của nó cùng với địa chỉ. Một host mới tới sẽ chờ lắng nghe thông báo này. Nếu host cảm thấy nó đã chờ lâu nhưng không nhận được thông báo, host có thể tự phát câu hỏi: Có bất kỳ trợ lý đối ngoại nào ở đây không?

2) Host di động đăng ký với trợ lý đối ngoại, cung cấp thông tin về địa chỉ ở nhà, địa chỉ MAC và một số thông tin về an ninh khác.

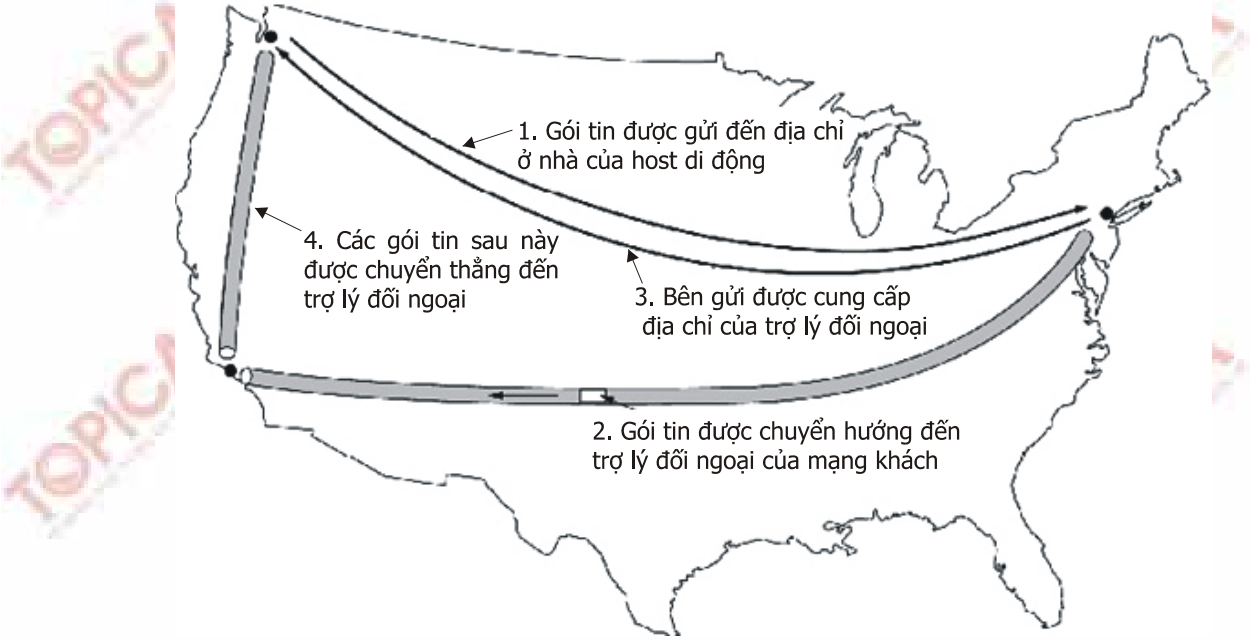
3) Trợ lý đối ngoại liên hệ với trợ lý đối nội ở nhà của host đó và nói: Một host của ông đang ở đây. Thông điệp mà trợ lý đối ngoại gửi cho trợ lý đối nội bên kia chứa địa chỉ mạng của trợ lý đối ngoại đó. Thông điệp này còn chứa thông tin an ninh dùng để thuyết phục trợ lý đối nội bên kia rằng host di động của nó thực sự đang ở đó.

4) Trợ lý đối nội bên kia kiểm tra thông tin an ninh, trong đó có một tem thời gian, để chứng tỏ được rằng tem này vừa được tạo ra trong vòng vài giây. Và nếu kết quả kiểm tra là tốt đẹp, nó sẽ bảo trợ lý đối ngoại bên kia tiến hành làm việc.

5) Khi trợ lý đối ngoại nhận được sự chấp thuận của trợ lý đối nội bên kia, nó tạo ra một đầu mục trong các bảng quản lý và thông báo cho host di động rằng: Bạn đã đăng ký thành công.

Lý tưởng nhất là khi một host di động rời khỏi một cell, nó phải thông báo với trợ lý đối ngoại ở đó để xóa đăng ký. Nhưng đa phần người sử dụng thường tắt máy ngay khi sử dụng xong.

Khi một gói tin được gửi đến một host di động, đầu tiên gói tin đó được gửi đến mạng LAN nhà của host đó (bước 1 trong hình 3.18).



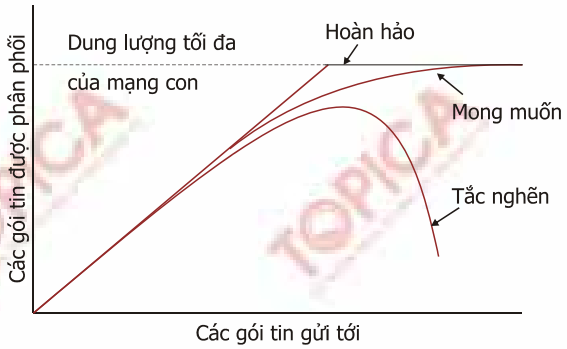
Hình 3.18: ***Mô hình truyền tin***

Bên gửi, ví dụ đang ở Bắc Giang, gửi gói tin đến mạng nhà của host di động ở Hà Nội. Giả sử host di động đang ở Nam Định. Trợ lý đối nội ở Hà Nội tìm ra được địa chỉ tạm thời của host di động, đóng gói gói tin đó và chuyển cho trợ lý đối ngoại của mạng ở Nam Định (bước 2). Đến phiên trợ lý đối ngoại ở Nam Định mở gói tin đó và phát cho host di động thông tin dưới dạng khung thông tin ở mức liên kết dữ liệu.

Sau đó trợ lý đối ngoại ở Nam Định sẽ bảo bên gửi ở Bắc Giang hãy đóng gói và gửi gói tin trực tiếp đến Nam Định (bước 3). Từ đó trở về sau, nhưng gói tin mà bên gửi muốn gửi cho host di động được gửi trực tiếp đến trợ lý đối ngoại tại Nam Định, rồi được trợ lý đối ngoại phát trực tiếp đến host (bước 4).

3.5. Các giải thuật chống tắc nghẽn

Khi có quá nhiều gói tin hiện diện trong một mạng con (hoặc một phần của nó), hiệu năng hoạt động của hệ thống bị giảm. Tình trạng này được gọi là "tắc nghẽn".



Hình 3.19: ***Mô tả tắc nghẽn***

Hình 3.19 mô tả lại hiện tượng tắc nghẽn. Khi số lượng gói tin chạy trong mạng con nằm dưới ngưỡng cho phép, chúng đều được phân phối đến đích (ngoại trừ những gói tin bị lỗi), và số lượng gói tin được phân phối tỷ lệ thuận với số lượng gói tin được phát ra lúc đầu. Tuy nhiên, khi mật độ giao thông tăng quá cao, các router không còn đáp ứng kịp nữa và chúng dần dần đánh mất một số gói tin. Điều này có xu hướng làm

cho vấn đề tắc nghẽn nghiêm trọng thêm. Khi mà giao thông cực cao, hiệu năng hệ thống sụp đổ hoàn toàn và hầu như không gói tin nào được phân phát đến đích.

Có vài yếu tố góp phần gây ra tắc nghẽn. Nếu đột nhiên nhiều luồng mang các gói tin đến một nút tại nhiều ngõ vào, và tất cả các gói tin này đều cần một ngõ ra, thì một hàng đợi sẽ xuất hiện. Nếu không đủ bộ nhớ để lưu các gói tin trên hàng đợi này, một số gói tin sẽ bị mất. Tăng thêm bộ nhớ chỉ giúp không mất gói tin trong hàng đợi, nhưng Nagle (1987) đã chỉ ra rằng: nếu một router có bộ nhớ vô hạn, sự tắc nghẽn lại càng tồi tệ hơn! Lý do là khi mà gói tin đến được đầu của hàng đợi thì nó đã bị mãn kỳ (Timed - Out), và do đó sẽ có nhiều phiên bản trùng với gói tin đó được bên gửi gửi đến router, làm tăng thêm tải của mọi hướng đi đến đích của gói tin. Các bộ xử lý chậm cũng có thể gây ra tắc nghẽn. Nếu CPU của router xử lý các gói tin trung chuyển qua nó chậm, hàng đợi cũng sẽ phát sinh, cho dù dung lượng các đường nối vào và ra đều vượt yêu cầu.

Tóm lại, đường truyền băng thông thấp có thể gây ra tắc nghẽn. Nâng cấp đường truyền nhưng năng lực xử lý của bộ xử lý tại router yếu cũng gây ra tắc nghẽn. Thành thử, nâng cấp một phần mà không phải là toàn bộ hệ thống chỉ đẩy sự tắc nghẽn từ nơi này đến nơi khác mà thôi. Vấn đề phát sinh từ sự bất cân đối giữa các bộ phận của hệ thống, và nó chỉ qua đi khi mà các bộ phận này được giữ cân bằng với nhau.

3.5.1. Các nguyên tắc chung để điều khiển tắc nghẽn

Nhiều bài toán trong các hệ thống phức tạp, ví dụ như trong mạng máy tính, có thể được xem xét theo quan điểm của lý thuyết điều khiển (control theory). Cách tiếp cận này dẫn đến việc chia các giải pháp thành hai loại: vòng đóng và vòng mở (closed loop and open loop). Các giải pháp dạng vòng đóng cố gắng giải quyết vấn đề tắc nghẽn bằng cách đưa ra thiết kế tốt cho mạng, thực chất là để đảm bảo tắc nghẽn sẽ không xảy ra. Một khi mạng được khởi động và chạy, sẽ không có việc sửa chữa giữa kỳ. Các công cụ thực hiện việc điều khiển kiểu vòng mở bao gồm việc quyết định khi nào nên chấp nhận luồng giao thông mới, quyết định khi nào thì bỏ qua các gói tin và bỏ qua gói nào. Tất cả các công cụ trên đều có đặc điểm chung là chúng đưa ra các quyết định mà không quan tâm đến trạng thái hiện hành của mạng.

Ngược lại, các giải pháp kiểu vòng đóng dựa trên quan niệm về chu trình phản hồi thông tin. Cách tiếp cận này bao gồm 3 phần:

1) Giám sát hệ thống để phát hiện nơi nào và khi nào xảy ra tắc nghẽn.

2) Chuyển thông tin đến những nơi cần có những hành động ứng phó.

3) Điều chỉnh lại hoạt động của hệ thống để khắc phục sự cố.

Nhiều kiểu đo lường có thể được sử dụng để giám sát một mạng con để phát hiện ra tắc nghẽn ở đó. Các kiểu đo lường thường dùng nhất là tỉ lệ các gói tin bị bỏ qua do thiếu không gian trữ đệm, chiều dài trung bình của các hàng đợi, số lượng các gói tin bị mãn kỳ và được tái truyền, thời gian trì hoãn gói tin trung bình. Trong mọi tình huống, các số đo tăng đồng nghĩa với việc tăng tắc nghẽn.

Bước thứ hai trong chu trình phản hồi là chuyển thông tin về tắc nghẽn từ điểm được phát hiện bị tắc nghẽn đến điểm có trách nhiệm xử lý tình huống đó. Cách dễ nhất là để cho router phát hiện ra tắc nghẽn phát thông báo đến nút nguồn vừa gửi thông tin đến làm tắc hệ thống. Dĩ nhiên, thông báo này làm cho tắc nghẽn tăng thêm tạm thời.

Một cách thông báo tắc nghẽn khác là: Người ta dành riêng một bit hoặc một trường trong gói tin để trong trường hợp có tắc nghẽn, router có thể bật bit hoặc trường này lên và gửi nó đến mọi ngõ ra nhằm thông báo cho các láng giềng của nó biết. Hoặc cũng có thể dùng cách phản hồi sau: Cho các host hoặc router thường xuyên gửi các gói tin thăm dò ra ngoài để hỏi thẳng về tình hình tắc nghẽn. Thông tin này có thể được sử dụng để chuyển hướng tìm đường vòng qua khu vực bị tắc nghẽn. Ví dụ thực tế: Một số đài phát thanh thường phái một số máy bay trực thăng bay vòng quanh thành phố để báo cáo lại những trục đường bị tắc, từ đó thông báo đến thính giả giúp họ chuyển hướng lái xe tránh những điểm nóng.

Sự hiện diện của tắc nghẽn đồng nghĩa với việc: tài nguyên của hệ thống không đủ để tải gánh nặng thông tin truyền qua. Vì thế ta nghĩ ra hai giải pháp: tăng tài nguyên hoặc giảm tải. Ví dụ, một mạng con có thể bắt đầu sử dụng các đường điện thoại quay số để tạm thời tăng băng thông giữa một số điểm nào đó. Trong các hệ thống vệ tinh, việc tăng công suất truyền đồng nghĩa với việc cung cấp băng thông lớn hơn. Chia tách lưu lượng thông tin cho chúng chạy trên nhiều đường đi khác nhau cũng có thể giúp tăng băng thông. Cuối cùng, các router dự phòng (thường để dự phòng tình huống các router chính bị sự cố) có thể được mang ra chạy trực tuyến để tăng dung lượng truyền tải của hệ thống khi tắc nghẽn nghiêm trọng xảy ra.

Tuy nhiên, đôi khi ta không thể tăng tài nguyên của hệ thống lên nữa, hoặc tài nguyên đã tăng tối đa. Cách thức duy nhất để chống lại tắc nghẽn là giảm tải. Có nhiều cách giảm tải, ví dụ: từ chối phục vụ một số người dùng, giảm cấp dịch vụ đối với vài hoặc tất cả người dùng, và buộc người dùng cung cấp lịch trình phát ra yêu cầu của họ.

3.5.2. Các biện pháp phòng ngừa tắc nghẽn

Tại tầng mạng, việc chọn sử dụng mạch ảo hay datagram sẽ tác động đến tắc nghẽn do nhiều giải thuật điều khiển tắc nghẽn chỉ chạy trên mạch ảo. Giải pháp "lập hàng đợi cho các gói tin và phục vụ chúng" liên quan đến việc một router có một hàng đợi cho mỗi ngõ vào, một hàng đợi cho mỗi ngõ ra hay cả hai. Nó cũng liên quan đến trình tự xử lý các gói tin trong hàng đợi (Round-Robin hay dựa trên sự ưu tiên). Chính sách hủy bỏ gói tin sẽ chỉ ra gói tin nào cần bị hủy bỏ khi không còn không gian chứa. Một chính sách tốt có thể giúp làm giảm tắc nghẽn, ngược lại có thể làm tắc nghẽn trầm trọng thêm.

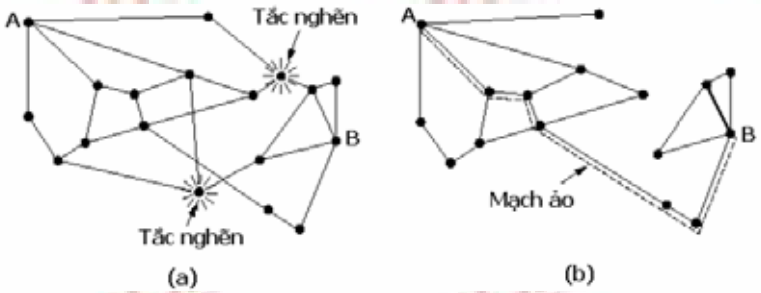
Một giải thuật tìm đường tốt có thể giúp tránh được tắc nghẽn bằng cách trải đều giao thông trên tất cả đường nối, trong khi một giải thuật tồi chỉ đơn giản gửi quá nhiều thông tin lên một đường tải đã quá tải rồi. Cuối cùng, việc quản lý thời gian sống của gói tin sẽ phải đưa ra quyết định là một gói tin có thể sống bao lâu trong hàng đợi trước khi bị hủy bỏ. Thời gian sống quá dài sẽ làm trì trệ công việc rất lâu. Nhưng nếu thời gian sống quá ngắn, các gói tin thỉnh thoảng sẽ bị mãn kỳ (Timed-Out) trước khi chúng đến được đích, vì thế dẫn đến việc tái truyền.

3.5.3. Điều khiển tắc nghẽn trong các mạng con dạng mạch ảo

Một giải pháp đơn giản là điều khiển cấp phép (Admission Control).

Ý tưởng như sau: một khi có cảnh báo về tắc nghẽn, hệ thống sẽ không thiết lập thêm mạch ảo nào nữa đến khi sự cố qua đi. Vì thế, trong lúc tắc nghẽn xảy ra, những cố gắng thiết lập mạch ảo đều thất bại. Lý do: cho phép nhiều người vào đấy sẽ làm cho vấn đề trở nên trầm trọng hơn. Cách tiếp cận khác là cho phép tạo ra các mạch ảo mới nhưng

cẩn trọng tìm đường cho các mạch ảo mới này đi vòng qua khu vực bị vấn đề tắc nghẽn. Ví dụ, xem xét mạng con như trong hình 3.20, trong đó hai router bị tắc nghẽn.



Hình 3.20 (a): ***Một mạng con bị tắc nghẽn. (b) Mạng con được vẽ lại sau khi loại trừ các điểm gây tắc nghẽn.***

Giả sử một host được nối với router A muốn thiết lập kết nối tới một host của router B. Thường thì kết nối này sẽ chạy qua một trong hai nút bị tắc nghẽn. Để tránh chuyện này, chúng ta vẽ lại mạng con như trong hình 3.20 (b), bỏ qua các router bị tắc nghẽn cùng với các đường nối của chúng. Đường chấm chỉ ra một đường đi có thể tránh được tắc nghẽn.

Một chiến lược khác liên quan đến mạch ảo là: host và mạng con thỏa thuận với nhau về việc thiết lập mạch ảo. Thỏa thuận này thường bao gồm dung lượng và đường đi của thông tin, chất lượng dịch vụ được yêu cầu và các thông số khác. Để đảm bảo thực hiện được thỏa thuận, mạng con sẽ dành riêng tài nguyên trên suốt con đường mạch ảo đi qua. Các tài nguyên này bao gồm không gian bảng vạch đường và bộ nhớ đệm trên các router, cùng với băng thông trên các đường nối. Trong tình huống này, tắc nghẽn hầu như không xảy ra trên một mạch ảo mới bởi vì tất cả tài nguyên cần thiết đã được đảm bảo sẵn dùng.

Kiểu dành riêng tài nguyên này có thể được thực hiện toàn thời gian như là một phương thức hoạt động chuẩn, hoặc chỉ được thực hiện khi tắc nghẽn xảy ra. Nếu được thực hiện toàn thời gian sẽ có hạn chế là lãng phí tài nguyên. Nếu đường truyền 6 Mbps được dùng riêng cho 6 mạch ảo, mỗi mạch ảo tiêu tốn 1 Mbps, thì đường truyền này luôn được đánh dấu là đầy, cho dù hiếm có khi nào 6 mạch ảo con của nó truyền hết công suất tại cùng thời điểm.

3.5.4. Điều khiển tắc nghẽn trong mạng con dạng Datagram

Trong mạng dạng Datagram, mỗi router có thể dễ dàng kiểm soát hiệu năng của các đường ra và các tài nguyên khác. Ví dụ, nó có thể gán cho mỗi đường nối một biến thực u, với giá trị từ 0.0 đến 1.0, dùng phản ánh hiệu năng gần đây của đường nối đó. Để duy trì độ chính xác tốt cho u, một mẫu hiệu năng tức thời f của đường nối sẽ được lấy thường xuyên, và u sẽ được cập nhật như sau: umới = a.ucũ + (1 - a) f ; trong đó hằng số a quyết định router quên đi lịch sử gần đây nhanh như thế nào.

Khi u vượt qua ngưỡng, đường ra rơi vào trạng thái "cảnh báo". Mỗi gói tin mới tới sẽ được giữ lại và chờ kiểm tra xem đường ra có ở trạng thái cảnh báo không. Nếu có, một số hành động sẽ được thực hiện, và chúng ta sẽ thảo luận ngay sau đây.

3.5.4.1. Các gói tin chặn (Choke Packets)

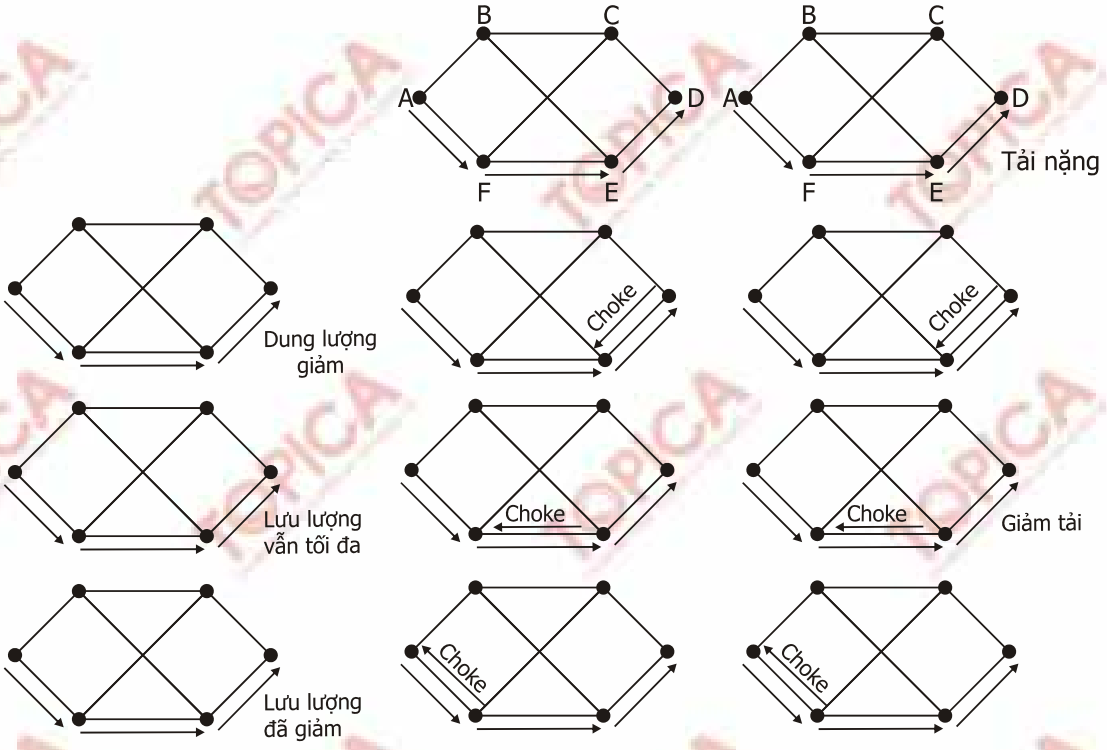
Khi một gói tin đến router và ngõ ra của nó đang ở trong trạng thái báo động, router sẽ gửi một gói tin chặn ngược về nút nguồn đã gửi gói tin đó. Gói tin gặp tắc nghẽn như

đã nói sẽ được đánh dấu để nó không làm phát sinh các gói tin chặn khác nữa. Khi gói tin chặn đến được nút nguồn, nút nguồn sẽ giảm lưu lượng thông tin đến điểm bị nghẽn đi X phần trăm. Do có thể còn vài gói tin đang trên đường đi đến đích bị nghẽn, sau này nút nguồn nên bỏ qua các gói tin chặn phát ra tiếp từ đích đó. Sau giai đoạn trên, nút nguồn bỏ thêm một khoảng thời gian để lắng nghe thêm các gói tin chặn khác. Nếu chúng còn tới, đường nối vẫn bị nghẽn, nút nguồn tiếp tục giảm dung lượng truyền. Nếu không còn gói tin chặn nào chạy ngược về nút nguồn trong thời gian lắng nghe, nó có thể từng bước tăng lưu lượng truyền lên.

3.5.4.2. Gửi các gói chặn từng bước một (Hop-by-Hop Choke Packets)

Ở tốc độ cao hoặc qua khoảng cách xa, việc gửi gói tin chặn ngược về nút nguồn là không hiệu quả, bởi vì phản ứng của nút nguồn sẽ chậm.

Một cách tiếp cận khác là làm cho gói tin chặn có tác dụng tại mọi nút trung gian mà nó đi qua. Hãy xem hình ví dụ:



Hình 3.21 (a): ***Một gói tin chặn chỉ tác động lên nút nguồn***

Hình 3.21 (b): ***Một gói tin chặn tác động lên mọi nút mà nó đi qua***

Trong hình 3.21 (b), ngay khi gói tin chặn vừa đến F, F liền giảm lưu lượng truyền đến D. Tương tự, khi gói tin chặn đến E, E sẽ giảm lưu lượng truyền đến F. Cuối cùng gói tin chặn đến A và lưu lượng được giảm suốt tuyến đường từ A đến D.

Hiệu quả của sơ đồ chặn từng bước một là có thể giải phóng điểm bị nghẽn nhanh chóng. Tuy nhiên cái giá phải trả là nó tiêu tốn băng thông hướng lên cho gói tin chặn. Nhưng cái lợi cuối cùng là ở chỗ, giải pháp này bóp chết tắc nghẽn ngay trong trứng nước.

TÓM LƯỢC CUỐI BÀI

Như thế là các bạn đã được học về liên kết dữ liệu và các giao thức truyền thông, tìm đường.

Ta cần phải nắm rõ được những vấn đề sau:

Liên kết dữ liệu và các giao thức truyền thông các bạn cần phải nắm rõ:

- Có 2 loại giao thức được xây dựng cho tầng Liên kết dữ liệu là: DLP đồng bộ (Asynchronous) và DLP dị bộ (Synchronous).
- Các giao thức đồng bộ được chia thành: hướng ký tự và hướng bit.
  - Hướng ký tự: căn cứ vào phương thức khai thác đường truyền thì được chia làm 3 loại là:
    - Truyền một chiều (Simplex) : có giao thức truyền tệp Kermit.
    - Truyền hai chiều luân phiên (Half-Duplex): có giao thức BSC (Binary Synchronous Control) của IBM.
    - Truyền hai chiều đồng thời (Full-Duplex): có giao thức IMP (Interface Message Protocols) trong mạng ARPANET của Bộ quốc phòng Mỹ.
  - Hướng bit: có giao thức HDLC (High-Level Data Link Control) là quan trọng nhất.
    - Hiểu được các đặc tính của HDLC: 3 loại trạm, 2 cấu hình đường kết nối và 3 chế độ truyền tải.
    - Hiểu được cấu trúc một khung dữ liệu (Frame) và ý nghĩa các trường trong khung.
    - Một vài kịch bản đối với HDLC.
- Trong quá trình truyền với các giao thức hướng bit không tránh khỏi các sai bit dữ liệu và có những bộ mã phát hiện, sửa những lỗi ấy. Các bạn cần hiểu cơ chế hoạt động của:
  - Kiểm tra chẵn lẻ (Parity Check).
  - Check sum bằng đa thức chuẩn.
  - Mã Hamming tự sửa một sai.

Tìm đường đi trong mạng:

- Hiểu được bản chất, mục tiêu, phân loại của các giải thuật tìm đường trên mạng.
- Hiểu và có thể thực hiện các giải thuật: (thuộc loại giải thuật chọn đường tập trung).
  - Giải thuật tìm đường đi ngắn nhất Dijkstra.
  - Giải thuật chọn đường đi tối ưu Ford-Fulkerson.
- Hiểu được cơ chế của giải pháp tìm đường Vector khoảng cách (Distance Vector).
- Giải pháp chọn đường "trạng thái kết nối" (Link State).
- Tìm đường phân cấp (Hierarchical Routing).
- Tìm đường trong mạng di động.

Các giải thuật chống tắc nghẽn

- Các nguyên tắc chung để điều khiển tắc nghẽn.
- Các biện pháp phòng ngừa tắc nghẽn.
- Điều khiển tắc nghẽn trong các mạng con dạng mạng ảo.
- Điều khiển tắc nghẽn trong các mạng con dạng Datagram.

CÂU HỎI TỰ LUẬN

Câu 1. Tại sao các giao thức xây dựng cho tầng liên kết dữ liệu (DLP) lại chia thành đồng bộ và dị bộ? Trong DLP đồng bộ ta cần mấy mức đồng bộ hóa? Đó là gì?

Câu 2. Hãy trình bày về các ký tự đặc biệt trong giao thức BSC/Basic Mode.

Câu 3. HDLC (High-Level Data Link Control) là gì? Nêu các đặc tính của nó.

Câu 4. Nêu cấu trúc một khung trong giao thức HDLC.

Câu 5. Hãy nêu cơ chế của phát hiện sai dùng phương pháp kiểm tra phần dư tuần hoàn (Cyclic Redundancy Check)? Lấy một ví dụ minh họa.

Câu 6. Trình bày mã Hamming tự sửa một sai.

Câu 7. Hãy lấy một ví dụ về mã Hamming tự sửa một sai.

Câu 8. Hãy nêu mục tiêu của giải thuật chọn đường và phân loại giải thuật chọn đường.

Câu 9. Hãy mô tả về giải thuật tìm đường đi ngắn nhất Dijkstra.

Câu 10. Hãy mô tả về giải thuật chọn đường tối ưu Ford-Fulkerson.

Câu 11. Ý tưởng của giải pháp chọn đường "Trạng thái kết nối" (Link State).
Quá trình "làm ngập" cần đến một gói tin cập nhật. Nêu cấu trúc của gói tin ấy.

Câu 12. Trong giải pháp tìm đường Vector khoảng cách (Distance- Vector), có một số tình huống phát sinh lỗi làm cho mạng mất ổn định. Hãy trình bày các lỗi và các giải pháp khắc phục chúng.

Câu 13. Hãy nêu nguyên nhân gây tắc nghẽn trong mạng.

Câu 14. Nêu các nguyên tắc chung điều khiển tắc nghẽn.

Câu 15. Nêu các biện pháp phòng ngừa tắc nghẽn.

BÀI TẬP TRẮC NGHIỆM

1. Tầng DataLink là tầng thứ mấy trong mô hình OSI?

a) 2. b) 3. c) 4. d) 5.

2. Dữ liệu trong tầng DataLink nằm dưới dạng nào?

a) Box. b) Package. c) Frame. d) Message.

3. BSC là giao thức hướng ký tự được phát triển bởi?

a) Đại học Colombia (Mỹ). b) IBM.

c) Bộ quốc phòng Mỹ. d) Đại học Stanford.

4. Mã phát hiện sai có tác dụng:

a) Phát hiện lỗi sai trong quá trình truyền với giao thức hướng ký tự.

b) Phát hiện lỗi sai bit khi truyền, trong các giao thức hướng bit.

c) Phát hiện và sửa mọi lỗi sai trong quá trình truyền.

d) Ngăn chặn sự xâm nhập của các Frame nguy hiểm.

5. Ký tự đặc biệt nào sau đây không nằm trong giao thức BSC?

a) SOH. b) EOT. c) EFT. d) NAK.

6. Giao thức HDLC (High-Level Data Link Control) có những loại trạm nào?

a) Trạm chính, trạm phụ và trạm hỗn hợp. b) Trạm đơn, trạm đa và trạm phức hợp.

c) Trạm chính và trạm phụ. d) Trạm Client và trạm Sever.

7. Các loại khung trong giao thức HDLC là gì?

a) I, S và C. b) I, S và U. c) I, C và U. d) S, C và U.

8. CRC là :

a) Mã phát hiện sai bằng cách kiểm tra phần dư tuần hoàn.

b) Mã phát hiện sai bằng cách kiểm tra theo chiều dọc và chiều ngang.

c) Mã Hamming phát hiện và sửa lỗi sai.

d) MÃ phát hiện sai bằng kiểm tra chẵn lẻ.

9. Một máy nhận được dữ liệu mà sau khi kiểm tra chẵn lẻ ta được các bit là 1010. Như vậy bit bị lỗi ở đây là bit thứ mấy?

a) 10. b) 9. c) 8. d) 7.

10. Mục tiêu nào sau đây không phải là mục tiêu của việc chọn đường?

a) Xác định hướng đi nhanh chóng, chính xác.

b) Khả năng tránh được các kết nối bị tắc nghẽn.

c) Giúp khung dữ liệu tìm được những con đường mà nó được ưu tiên.

d) Khả năng thích nghi với những thay đổi về tải đường truyền.

11. Giải thuật chọn đường đi ngắn nhất Dijkstra là:

a) Giải thuật có mục đích tìm đường đi ngắn nhất từ một nút cho trước trên đồ thị đến các nút còn lại.

b) Giải thuật có mục đích tìm đường đi ngắn nhất từ tất cả các nút đến một nút đích cho trước trên mạng.

c) Giải thuật chỉ tìm đường đi ngắn nhất từ một nút đến một nút khác trên mạng.

d) Là giải thuật chọn đường phân tán.

12. Giải thuật chọn đường tối ưu Ford-Fulkerson thuộc loại:

a) Chọn đường tập trung. b) Chọn đường phân tán.

c) Chọn đường tĩnh. d) Chọn đường động.

13. Trong giải pháp chọn đường "Trạng thái kết nối" (Link State). Trường số thứ tự trong gói tin cập nhật (hay gói tin trạng thái kết nối) có tác dụng gì?

a) Để phân biệt gói tin đó với các gói tin khác trong mạng.

b) Để nút đích có thể xác định được phiên bản của gói tin khi nó được gửi từ nút nguồn.

c) Để đếm thời gian sống của gói tin (Time to Live).

d) Để giúp nút nguồn kiểm soát những gói tin mà nó đã truyền đi.

14. Vì sao cần phải có tìm đường phân cấp?

a) Vì mạng máy tính phát triển đến mức không một router nào có đủ khả năng trữ một đầu mục thông tin về một router khác, vì thế việc tìm đường phải phát triển theo đường hướng khác: tìm đường phân cấp.

b) Vì bản thân các Router cũng có cấp độ, nên việc tìm đường cũng phải phân cấp.

c) Vì tìm đường phân cấp giúp nút nguồn tìm thấy nút đích ngay tức thì.

d) a, b và c đều đúng.

15. Theo quan điểm lý thuyết điều khiển ta chia các giải pháp điều khiển tắc nghẽn thành mấy loại?

a) 2. b) 3. c) 4. d) 5.