Các yếu tố cấu thành giao thức vận chuyển

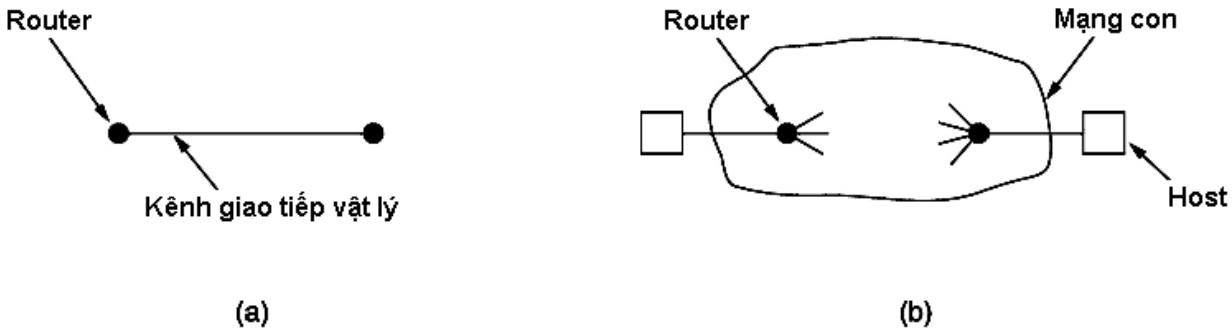
Bởi:

unknown

Các yếu tố cấu thành giao thức vận chuyển

Cũng giống như giao thức ở tầng liên kết dữ liệu, giao thức vận chuyển phải đối phó với các vấn đề về điều khiển lỗi, đánh số thứ tự gói tin và điều khiển luồng dữ liệu.

Tuy nhiên, giao thức trên hai tầng có nhiều điểm khác biệt quan trọng. Những khác biệt này xuất phát từ sự khác biệt của môi trường hoạt động của chúng (như được chỉ ra trong hình H7.2).



(a) Môi trường của lớp liên kết dữ liệu.(b) Môi trường của lớp vận chuyển (H7.2)

Tại lớp liên kết dữ liệu, hai router giao tiếp với nhau qua một kênh truyền vật lý, trong khi tại lớp vận chuyển, kênh truyền này được thay bằng cả một mạng con. Sự khác nhau này sẽ dẫn đến nhiều hệ lụy mà những người thiết kế giao thức vận chuyển phải đau đầu giải quyết: định địa chỉ các tiến trình trên các host khác nhau như thế nào, xử lý như thế nào đối với những trường hợp mất gói tin trong quá trình trao đổi hoặc gói tin đi chậm dẫn đến mãn kỳ và gởi thêm một gói tin bị trùng lắp, đồng bộ hóa hai tiến trình đang trao đổi dữ liệu như thế nào khi mà chúng đang ở rất xa nhau.

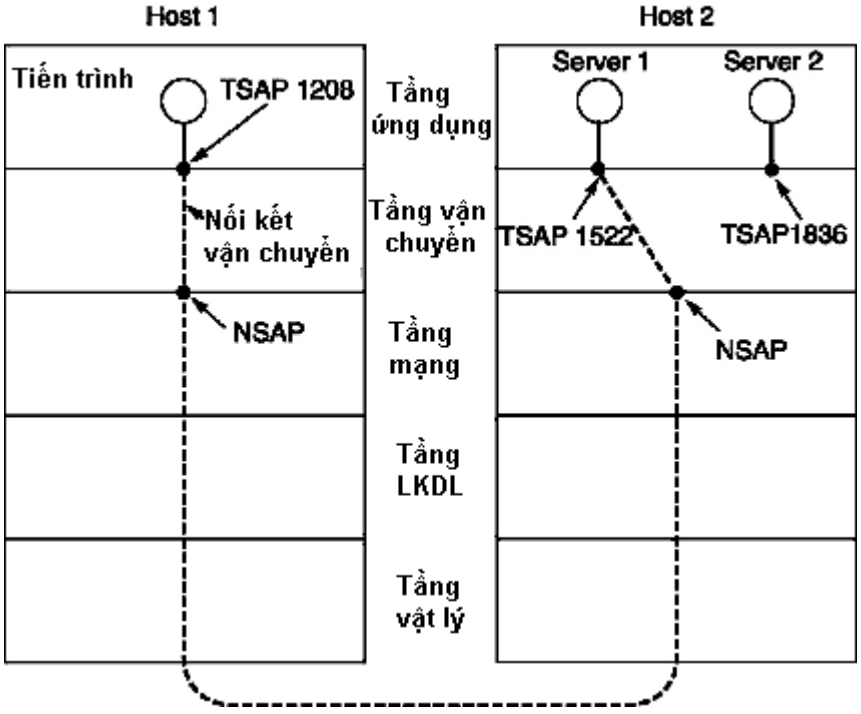
Định địa chỉ

Khi một tiến trình mong muốn thiết lập nối kết với một tiến trình khác từ xa, nó phải chỉ ra rằng nó muốn kết nối với tiến trình nào. (Vận chuyển hướng không nối kết cũng gặp vấn đề tương tự: thông điệp sẽ gởi đến ai?). Một phương pháp định địa chỉ ở tầng vận chuyển của Internet là dùng số hiệu cổng (port), còn ở trong mạng ATM là AAL-SAP. Chúng ta sẽ dùng từ chung nhất để định địa chỉ tiến trình là TSAP (Transport Service Access Point). Tương tự, địa chỉ trong tầng mạng được gọi là NSAP.

Hình H7.3 mô phỏng mối quan hệ giữa NSAP, TSAP và kết nối vận chuyển. Các tiến trình ứng dụng, cả client và server đều phải gắn vào một TSAP và thiết lập nối kết đến TSAP khác. Và kết nối này chạy qua cả hai TSAP. Mục tiêu của việc sử dụng các TSAP là vì trong một số mạng, mỗi máy tính chỉ có một NSAP, do đó cần phải có cách phân biệt nhiều điểm cuối mức vận chuyển khi chúng đang chia sẻ một NSAP.

Ví dụ, dàn cảnh một cuộc kết nối mức vận chuyển có thể diễn ra như sau:

1. Một server phục vụ thông tin về thời gian trên host 2 gắn nó vào TSAP 1522 để chờ một cuộc gọi đến.
2. Một tiến trình ứng dụng chạy trên host 1 muốn biết giờ hiện tại, vì thế nó đưa ra một yêu cầu nối kết chỉ ra TSAP 1208 là cổng nguồn và TSAP 1522 là cổng đích. Hành động này dẫn đến một kết nối vận chuyển được thiết lập giữa hai tiến trình client và server trên hai host 1 và 2.



TSAP, NSAP và kết nối vận chuyển (H 6.3.)

1. Tiến trình client gởi một yêu cầu đến server để hỏi về thời gian.
2. Server trả lời thời gian hiện tại cho client.

3. Kết nối vận chuyển cuối cùng được giải phóng.

Thiết lập nối kết

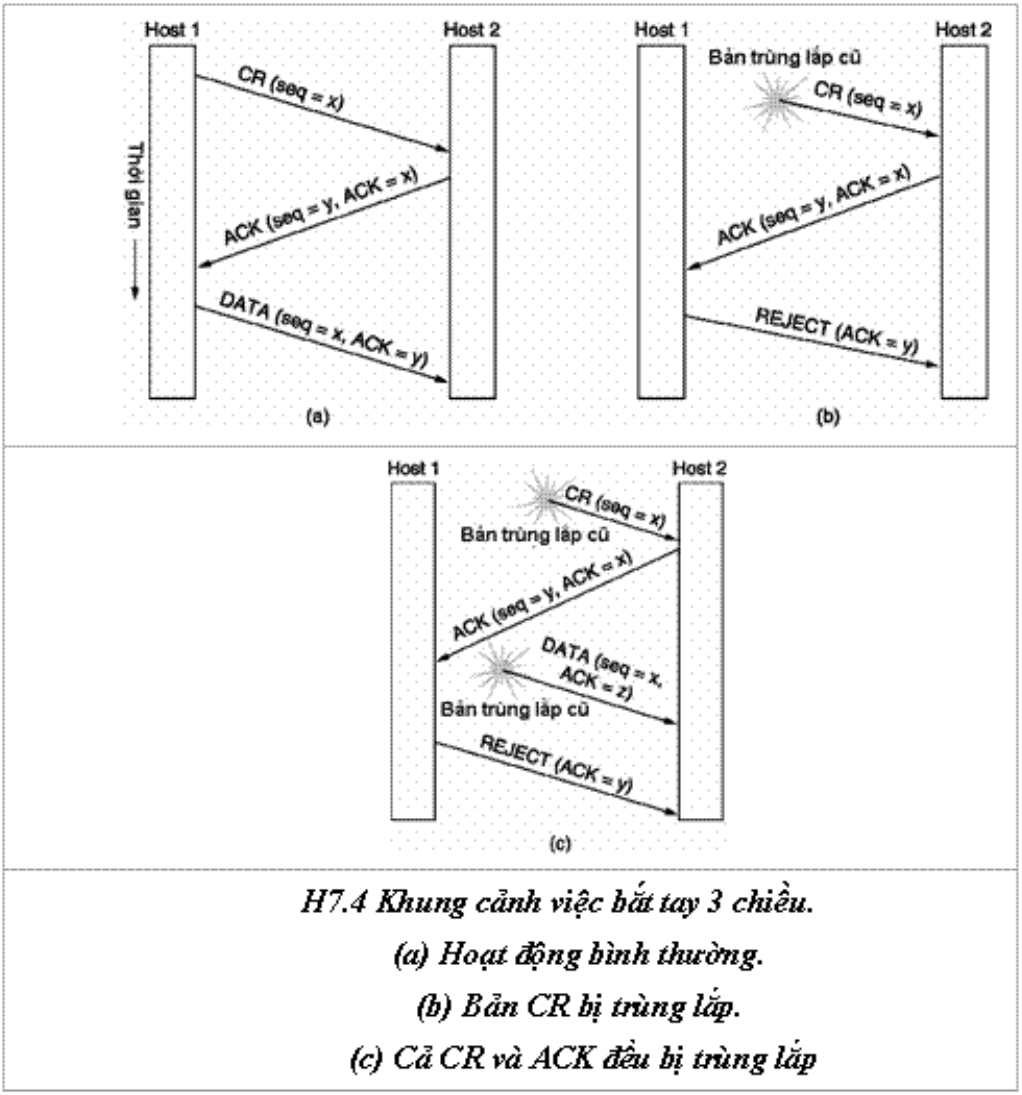
Việc thiết lập nối kết nghe có vẻ dễ dàng, nhưng khi thực hiện có thể sẽ gặp nhiều rắc rối. Thoạt nhìn, một phiên thiết lập nối kết sẽ diễn ra như sau: một bên sẽ gởi TPDU yêu cầu nối kết (Connection Request – CR) đến bên kia, bên kia sẽ gởi một TPDU trả lời chấp nhận nối kết (Connection Accepted – CA).

Vấn đề phát sinh khi mạng làm mất, tồn trữ quá lâu hay làm trùng lắp các gói tin do hai thực thể vận chuyển trao đổi qua lại với nhau. Ví dụ một tình huống như sau: tiến trình 1 gởi yêu cầu kết nối đến tiến trình 2, yêu cầu này bị các mạng con trung gian trì hoãn do tắc nghẽn. Mãn kỳ, tiến trình 1 gởi lại yêu cầu nối kết, vừa lúc đó yêu cầu nối kết bị trì hoãn cũng đến tiến trình 2.

Giải thuật thiết lập nối kết phổ biến nhất là giải thuật bắt tay 3 chiều (three-way hand-shake). Xin xem các tình huống được mô phỏng trong Hình H7.4. Giả sử yêu cầu nối kết phát sinh ở host 1. Host 1 chọn một số thứ tự là *x* và đính kèm số đó trong TPDU CR (*CR (seq=x)*) gởi đến host 2. Host 2 báo nhận ACK (*ACK (seq = y, ACK = x)*) và thông báo số thứ tự khởi đầu của nó là *y*. Cuối cùng host 1 báo nhận cho host 2 nó đã biết số thứ tự khởi đầu của host 2 là *y* bằng TPDU dữ liệu đầu tiên gởi đến host 2 (*DATA (seq=x, ACK=y)*).

Bây giờ xét đến tình huống TPDU CR bị trùng lắp. Khi TPDU CR thứ hai đến host 2, host 2 liền trả lời ACK vì tưởng rằng host 1 muốn thiết lập nối kết khác. Khi host 1 từ chối cố gắng thiết lập nối kết của host 2, host 2 hiểu rằng nó đã bị lừa bởi CR bị trùng lắp và sẽ từ bỏ nối kết đó.

Trường hợp xấu nhất là cả hai TPDU CR và ACK của host 1 đều bị trùng lắp. Như trong ví dụ (b), host 2 nhận được một CR trễ và trả lời cho yêu cầu đó với số thứ tự khởi đầu *y*. Giả sử, không may trong trả lời cho yêu cầu CR trước đó, host 2 thông báo số thứ tự khởi đầu của nó là *z*. Báo nhận ở chiều thứ ba của host 1 lại bị trễ. Khi host 1 nhận được báo nhận *ACK (seq=y, ACK=x)*, nó nhận ra rằng thông báo *DATA (seq=x, ACK=z)*bị trễ, do đó nó từ bỏ nối kết này.



H7.4 Khung cảnh việc bắt tay 3 chiều.

(a) Hoạt động bình thường.

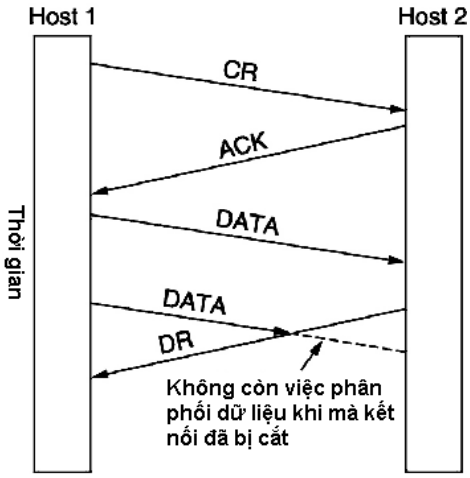
(b) Bản CR bị trùng lắp.

(c) Cả CR và ACK đều bị trùng lắp

Giải phóng nối kết

Việc giải phóng nối kết đơn giản hơn thiết lập nối kết. Tuy nhiên, người ta sẽ còn gặp nhiều khó khăn không ngờ tới. Bây giờ chúng ta sẽ đề nghị hai kiểu giải phóng nối kết: dị bộ và đồng bộ. Kiểu dị bộ hoạt động như sau: khi một bên cắt nối kết, kết nối sẽ bị hủy bỏ (giống như trong hệ thống điện thoại). Kiểu đồng bộ làm việc theo phương thức ngược lại: khi cả hai đồng ý hủy bỏ nối kết, nối kết mới thực sự được hủy.

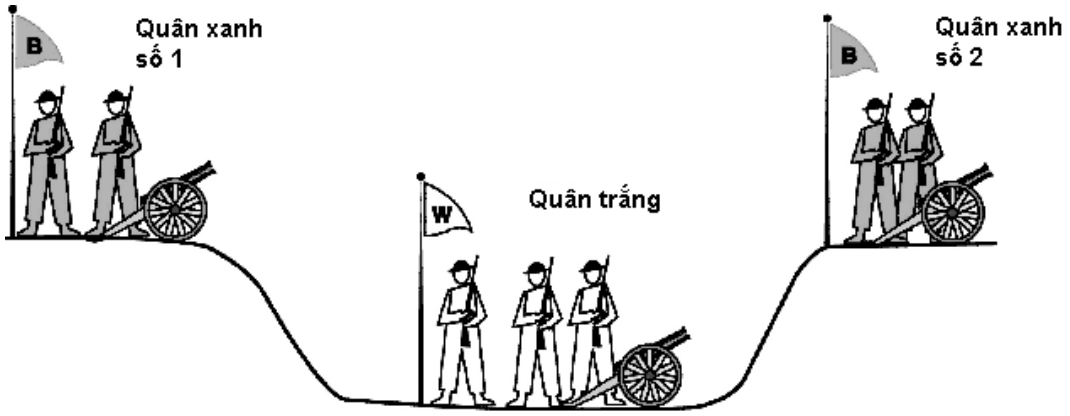
Giải phóng nối kết kiểu dị bộ là thô lỗ và có thể dẫn đến mất dữ liệu. Ví dụ tình huống trong Hình H7.5. Sau khi nối kết thành công, host 1 gởi một gói dữ liệu đến đúng host 2. Sau đó host 1 gởi tiếp một gói dữ liệu khác. Không may, host 2 gởi đi một yêu cầu cắt nối kết (DISCONNECT) trước khi gói dữ liệu thứ hai đến. Kết quả là kết nối được giải phóng và dữ liệu bị mất.



Sự cắt kết nối một cách thô lỗ sẽ dẫn đến mất dữ liệu (H7.5)

Rõ ràng, chúng ta cần một giải pháp hữu hiệu hơn để tránh mất dữ liệu. Một giải pháp là sử dụng việc giải phóng nối kết đồng bộ, trong đó, mỗi host đều có trách nhiệm trong việc giải phóng nối kết. Một nút phải tiếp tục nhận dữ liệu sau khi đã gởi đi yêu cầu giải phóng nối kết (DISCONNECT REQUEST – CR) đến bên đối tác, cho đến khi nhận được chấp thuận hủy bỏ nối kết của bên đối tác đó. Người ta có thể hình dung giao thức như sau: đầu tiên host 1 nói: "Tôi xong rồi, anh xong chưa?". Nếu host 2 trả lời: "Tôi cũng xong, tạm biệt" thì kết nối coi như được giải phóng an toàn.

Tuy nhiên, giải pháp trên không phải lúc nào cũng chạy đúng. Có một bài toán nổi tiếng dùng để mô tả vấn đề, được gọi là bài toán "**hai sứ quân**" (Two army problem).



Bài toán hai sứ quân (H7.6)

Có hai sứ quân đang dàn trận đánh nhau. Quân trắng dàn quân dưới thung lũng, quân xanh chia thành hai cánh quân chiếm lĩnh hai đỉnh đồi áng ngữ hai bên thung lũng đó. Chỉ huy của hai cánh quân xanh muốn thông báo và nhất trí với nhau về thời điểm cùng tấn công quân trắng. Do quân số hai cánh quân xanh cộng lại mới đủ sức thắng quân trắng, một cánh quân xanh tấn công riêng lẻ sẽ bị quân trắng tiêu diệt.

Hai cánh quân xanh muốn đồng bộ hóa cuộc tấn công của họ bằng cánh gởi các thông điệp qua lại. Nhưng những thông điệp đó phải chạy ngang qua thung lũng và có khả năng bị quân trắng phá hỏng. Câu hỏi ở đây là có giao thức nào đảm bảo sự thắng lợi của quân xanh hay không?

Giả sử chỉ huy cánh quân xanh số 1 gởi thông điệp đến chỉ huy cánh quân xanh số 2: "Tôi dự định tấn công vào lúc hoàng hôn ngày 14 tháng 12 năm 2004, có được không?". May mắn thay, chỉ huy cánh quân xanh số 2 nhận được thông điệp và trả lời "Đồng ý". Vậy cuộc tấn công có chắc xảy ra không? Không chắc, bởi vì chỉ huy cánh quân xanh số 2 không chắc câu trả lời của anh ta đến được chỉ huy của cánh quân số 1.

Bây giờ ta cải tiến giao thức thêm một bước: cho nó trở thành giao thức ba chiều: Bên cánh quân số 1 gởi bản hiệp đồng tấn công cho bên cánh quân số 2, bên cánh quân số 2 trả lời đồng ý, bên cánh quân 1 thông báo cho bên 2 nó đã biết được sự đồng ý của bên 2. Thế nhưng nếu thông báo cuối cùng của bên 1 bị mất thì sao? Bên 2 cũng sẽ không tấn công!

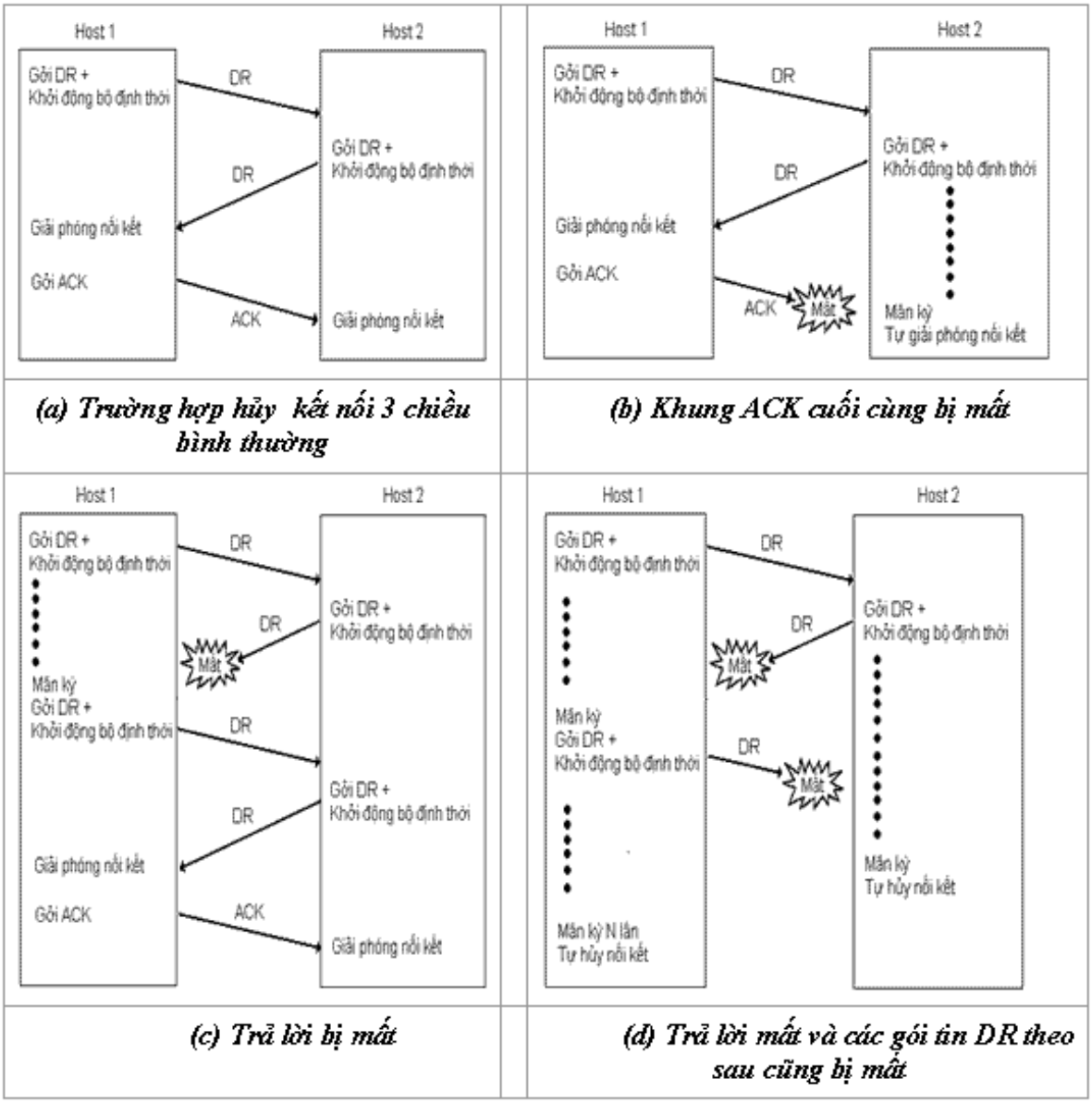
Nếu ta cố cải tiến thành giao thức n chiều đi nữa thì việc hiệp đồng vẫn thất bại nếu thông báo cuối cùng bị mất.

Ta có thể thấy mối tương đồng giữa bài toán hai sứ quân và giải pháp giải phóng nối kết. Thay vì hợp đồng tấn công, hai bên hợp đồng hủy nối kết!

Giải pháp cuối cùng là hai bên sử dụng phương pháp hủy nối kết ba chiều cùng với bộ định thời:

- Bên phát động việc hủy nối kết sẽ bật bộ định thời cho mỗi yêu cầu giải phóng nối kết của nó, nếu yêu cầu giải phóng nối kết bị mãn kỳ mà chưa nhận được trả lời của bên đối tác, nó sẽ gởi lại yêu cầu một lần nữa. Nếu yêu cầu hủy nối kết bị mãn kỳ liên tục N lần, bên phát động sẽ tự ý hủy bỏ nối kết đó.
- Bên đối tác khi nhận được yêu cầu hủy nối kết từ phía phát động, sẽ trả lời chấp thuận và cũng bật bộ định thời. Nếu mãn kỳ mà trả lời chấp thuận của nó không có báo trả từ phía phát động, bên đối tác sẽ tự hủy nối kết.

Hình H7.7 sẽ mô phỏng một số tình huống phát sinh trong quá trình hủy nối kết 3 chiều có sử dụng bộ định thời.



Một số tình huống hủy nối kết theo phương pháp 3 chiều (H7.7)

Điều khiển thông lượng

Điều khiển thông lượng trong tầng vận chuyển về cơ bản là giống giao thức cửa sổ trượt trong tầng liên kết dữ liệu, nhưng kích thước cửa sổ của bên gởi và bên nhận là khác nhau. Mỗi host có thể có quá nhiều kết nối tại một thời điểm, vì thế nó không chắc là có thể đảm bảo cung cấp đủ số lượng buffer cho mỗi kết nối nhằm thực hiện đúng giao thức cửa sổ trượt. Cần phải có sơ đồ cung cấp buffer động.

Trước tiên, bên gởi phải gởi đến bên nhận một yêu cầu dành riêng số lượng buffer để chứa các gói bên gởi gởi đến. Bên nhận cũng phải trả lời cho bên gởi số lượng buffer tối đa mà nó có thể cung cấp. Mỗi khi báo nhận ACK cho một gói tin có số thứ tự

SEQ_NUM, bên nhận cũng phải gởi kèm theo thông báo cho bên gởi biết là lượng buffer còn lại là bao nhiêu để bên gởi không làm ngập bên nhận.

Ví dụ sau sẽ mô phỏng một tình huống trao đổi thông tin giữa hai máy A và B.

	A	Thông điệp	B	Giải thích
1	→	<yêu cầu 8 buffers>	→	A muốn B cung cấp 8 buffers
2	←	<ack = 0, buf = 4>	←	B chỉ cấp cho A 4 buffers thôi
3	→	<seq = 0, data = m0>	→	A còn lại 3 buffers
4	→	<seq = 1, data = m1>		A còn lại 2 buffers
5	← →	<seq = 2, data = m2>	← → .	Thông điệp bị mất, nhưng A nghĩ nó còn 1 buffer
6	→ →	<ack = 1, buf = 3>	→ →	B báo nhận cho thông điệp 0 và 1, còn 3 buffers
7	←	<seq = 3, data = m3>	←	A còn lại 1 buffer
8		<seq = 4, data = m4>		A không còn buffer nào và phải dừng
9		<seq = 2, data = m2>		Thông điệp thứ 2 của A mãn kỳ và được truyền lại
10	←	<ack = 4, buf = 0>	←	Mọi thứ đã được báo nhận, nhưng A vẫn nghẽn
11	←	<ack = 4, buf = 1>	←	A có thể gởi 1 gói tin thứ 5
12	→ →	<ack = 4, buf = 2>	→ →	B có thêm 1 buffer nữa
13	←	<seq = 5, data = m5>	←	A còn lại 1 buffer
14		<seq = 6, data = m6>	←	A nghẽn một lần nữa
15		<ack = 6, buf = 0>		A vẫn còn nghẽn
16	...	<ack = 6, buf = 4>		Khả năng dẫn đến deadlock

Ví dụ một phiên giao dịch giữa hai thực thể tầng vận chuyển (H7.8)

Một vấn đề tiềm tàng trong sơ đồ dùng buffer động là cơ chế hoạt động của nó có thể dẫn đến deadlock. Ví dụ trong hàng 16, nếu báo nhận <ack = 6, buf = 4> của bên B bị mất, cả hai bên A và B đều rơi vào trạng thái deadlock. Để tránh tình trạng này, nên cho các host định kỳ gởi các báo nhận và trạng thái buffer lên mọi kết nối vận chuyển của chúng.