Trường Đại Học Bách Khoa Tp.HCM
Khoa Khoa Học và Kỹ Thuật Máy Tính

Bài giảng
Mạng máy tính

ThS. NGUYỄN CAO ĐẠT
E-mail:dat@cse.hcmut.edu.vn

Bài giảng 7: Tầng truyền tải (tt)

Tham khảo:

Chương 3: "Computer Networking – A top-down approach"
Kurose & Ross, 5th ed., Addison Wesley, 2010.

Chương 3: Mục lục

- 3.1 Các dịch vụ tầng-truyền tải
- 3.2 Sự dồn và tách
- 3.3 Sự truyền tải không kết nối: UDP
- 3.4 Sự truyền tải hướng kết nối : TCP
  - cấu trúc đoạn tin
  - truyền tải dự liệu tin cậy
  - kiểm soát lưu lượng
  - quản lý kết nối
- 3.5 Các nguyên lý của kiểm soát tắc nghẽn
- 3.6 Kiểm soát tắc nghẽn trong TCP

Truyền tải dữ liệu tin cậy TCP

- TCP tạo dịch vụ ttdltc trên nên dịch vụ không tin cậy IP
- Các khúc được tạo đường ống
- ACK cộng dồn
- TCP chỉ sử dụng một bộ đếm thời gian cho truyền tải lại
- Truyền tải lại được kích hoạt bởi:
  - sự kiện hết thời gian chờ
  - Trùng lặp ACK
- Đầu tiên xem xét ng/gửi TCP đơn giản:
  - bỏ qua các ack trùng lặp
  - bỏ qua kiểm tra lưu lượng, kiểm tra tắc nghẽn

Các sự kiện phía người gửi TCP:

nhận dữ liệu từ ứ/d:

- Tạo ra khúc với STT
- STT là stt trên luồng-byte của byte dữ liệu đầu tiên trong đoạn
- khởi động bộ đếm t/g nếu nó chưa chạy (bộ đếm t/g cho khúc dữ liệu chưa ACK lâu nhất)
- khoảng t/g hết hạn: `TimeOutInterval`

hết giờ:

- gửi lại khúc dữ liệu mà gây nên hết t/g chờ
- khởi động lại bđtg

Nhận được ACK:

- Nếu đó là ACK cho các khúc trước đó chưa được ACK
  - cập nhật danh sách các gói đã được ACK
  - chạy lại bđtg nếu như còn có các khúc chưa ACK

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
   switch(event)
   event: nhận được dữ liệu từ ứng dụng tầng trên
        tạo ra đoạn TCP với STT NextSeqNum
        if (bđtg không chạy)
             khởi chạy bđtg
        đẩy đoạn xuống IP
        NextSeqNum = NextSeqNum + length(data)

    event: bđtg hết giờ
        gửi lại đoạn chưa ACK với STT nhỏ nhất
        khởi chạy bđtg

    event: nhận được ACK, với giá trị trường ACK là y
        if (y > SendBase) {
             SendBase = y
            if (còn đoạn chưa ACK)
                  khởi chạy bđtg
            }
 }  /* end of loop forever */
```

ng/gửi TCP (đơn giản hóa)

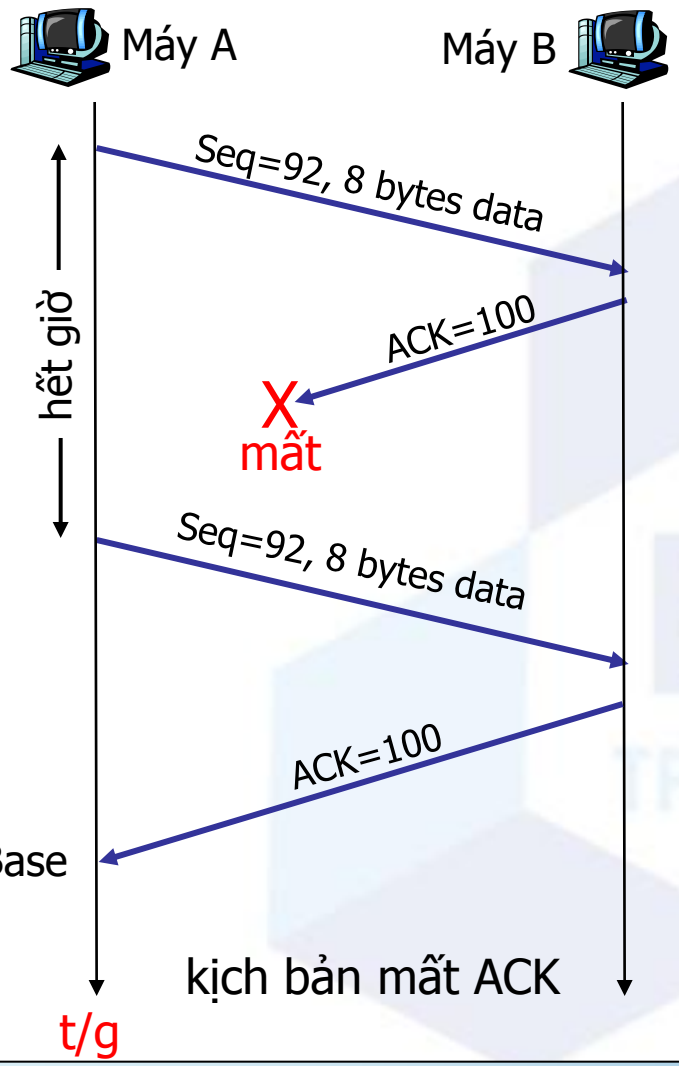
Chú thích:

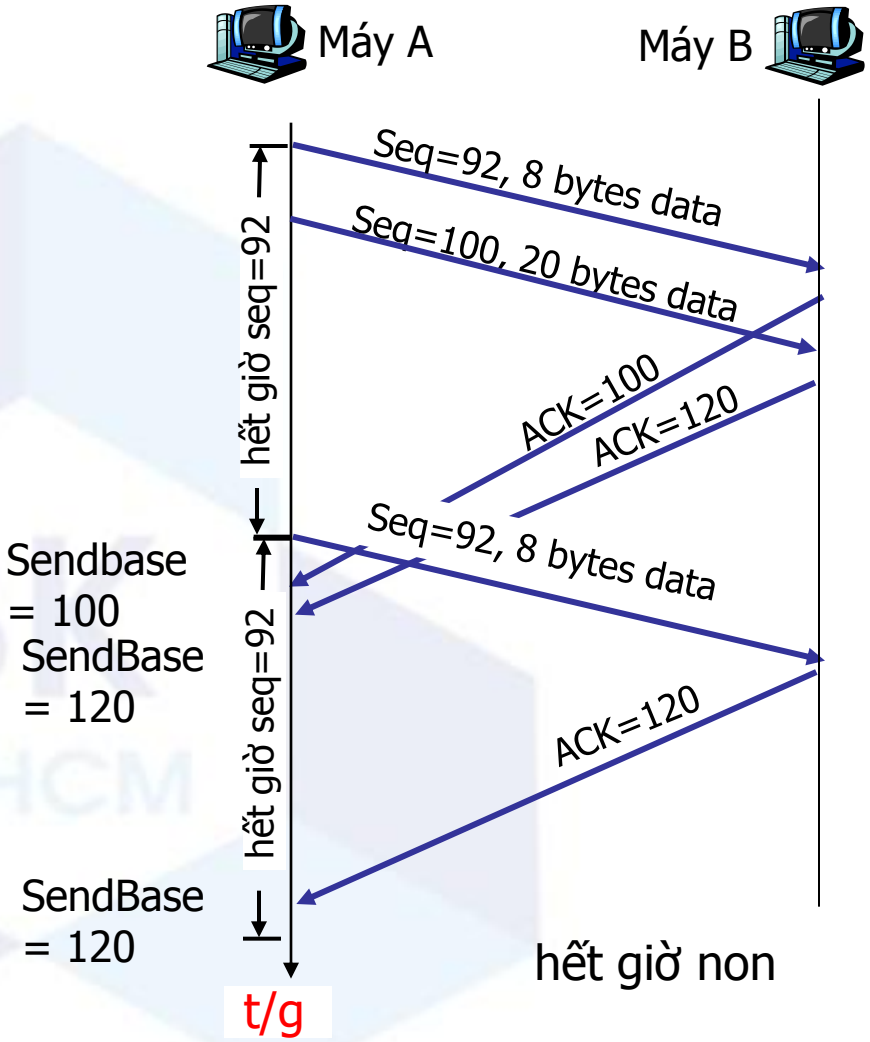
- SendBase-1: byte được ack cộng dồn cuối cùng

Ví dụ:

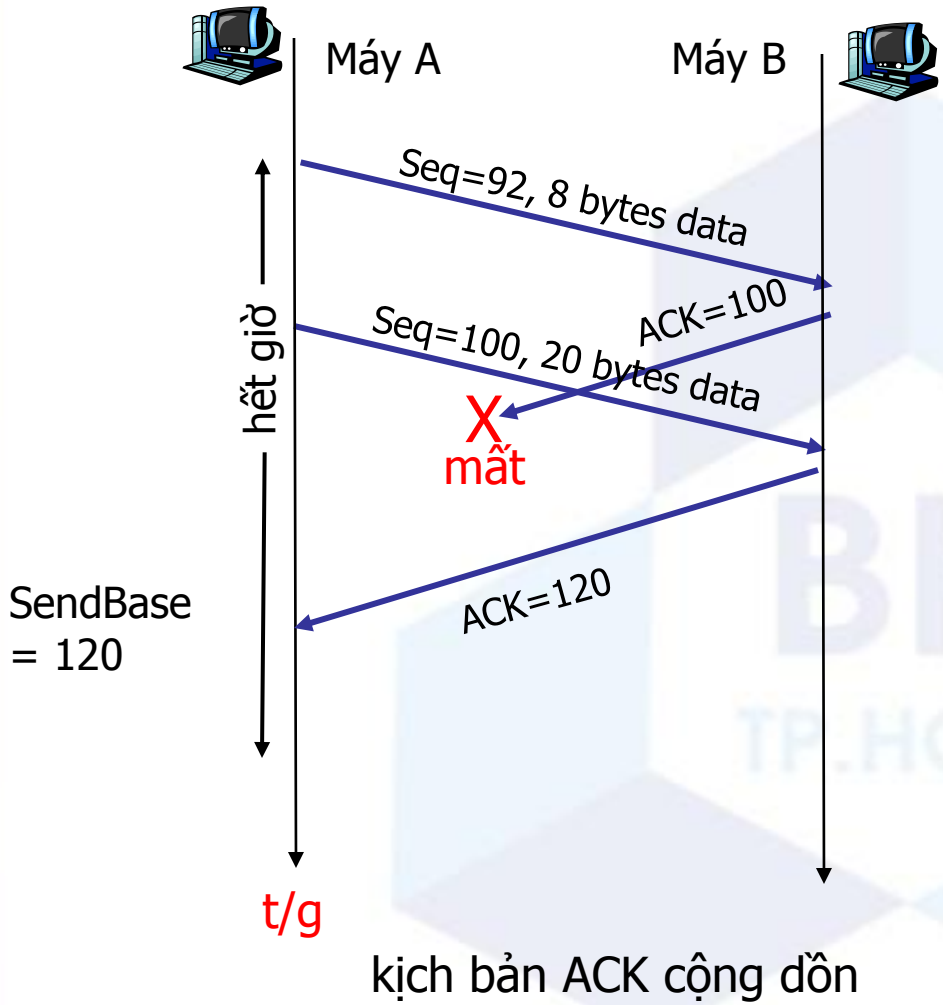
- SendBase-1 = 71; y= 73, vậy người nhận cần 73+ ; y > SendBase, vì vậy có thêm dữ liệu được ack

TCP: các kịch bản truyền tải lại





TCP: các kịch bản truyền tải lại (tt)



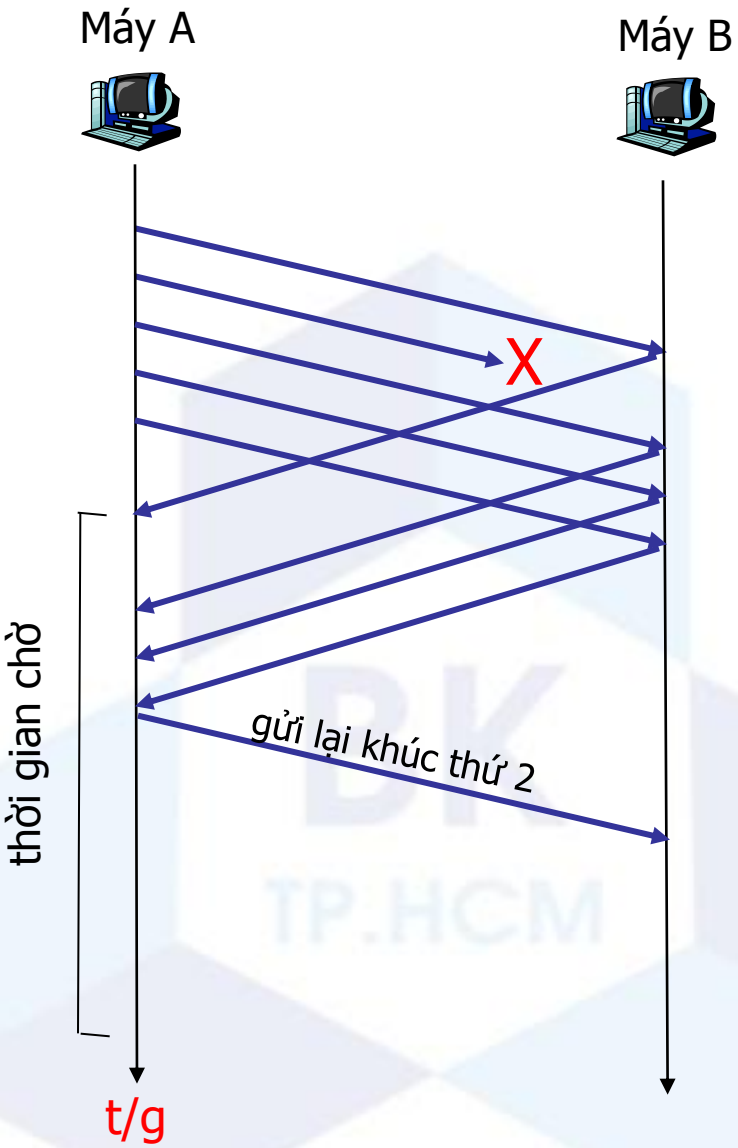
kịch bản ACK cộng dồn

Tạo ACK trong TCP [RFC 1122, RFC 2581]

Sự kiện tại ng/nhận	Hành vi của ng/ nhận TCP
Sự đến của khúc đúng thứ tự với STT hợp lí. Tất cả dữ liệu từ STT về trước đã được ACK	Trì hoãn việc ACK. Chờ khúc tiếp theo trong 500ms. Nếu không có khúc nào tiếp theo, gửi ACK
Sự đến của khúc đúng thứ tự với STT hợp lí. Một đoạn khác đang chờ được ACK	Ngay lập tức gửi một ACK cộng dồn, xác nhận cả hai đoạn dữ liệu đúng thứ tự
Sự đến của khúc sai-thứ-tự với STT cao hơn STT mong đợi. Phát hiện ra sự thiếu hụt	Ngay lập tức gửi một *ACK lặp*, chỉ rõ STT của byte mong đợi tiếp theo
Sự đến của khúc mà khỏa lấp sự thiếu hụt một phần hoặc toàn bộ	Ngay lập tức gửi ACK

Truyền lại nhanh

- Thời gian chờ thường tương đối dài:
  - sẽ bị trị hoãn lâu trước khi gửi lại gói bị mất
- Phát hiện mất khúc thông qua ACK lặp.
  - Ng/gửi thường gửi nhiều khúc liên tục
  - Nếu một khúc bị mất thì thường sẽ có nhiều ACK trùng lặp.
- Nếu người nhận nhận được 3 ACK trùng lặp cho cùng một khúc dữ liệu, nó sẽ suy ra là các khúc dữ liệu theo sau đã bị mất:
  - truyền lại nhanh: gửi lại khúc dữ liệu trước khi bộ đếm thời gian hết hạn


Máy A
Máy B
X
thời gian chờ
gửi lại khúc thứ 2
t/g

Giải thuật truyền tải lại nhanh:

```
sự kiện: nhận được ACK, với trường ACK có giá trị y
            if (y > SendBase) {
                SendBase = y
                if (không có khúc nào chưa được ACK)
                    khởi động bộ đếm thời gian
            }
            else {
                tăng bộ đếm số ACK cho y trùng
                if (số ACK trùng = 3) {
                    gửi lại khúc với STT y
                }
```

một ACK trùng
cho một khúc đã được ACK

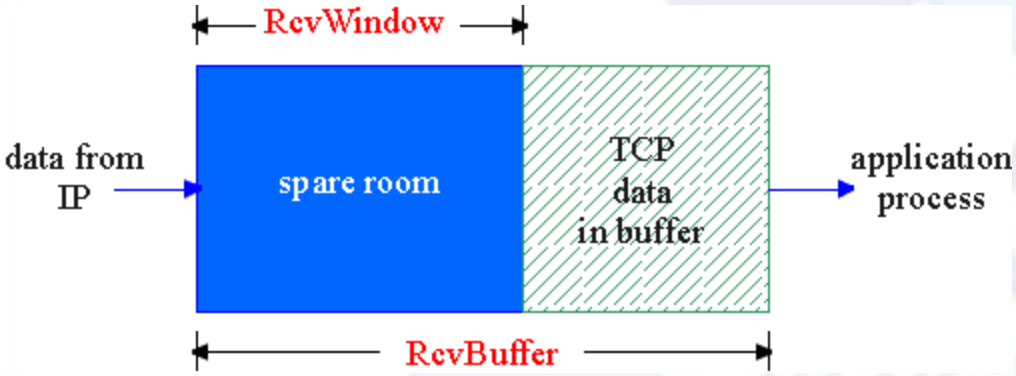
truyền tải lại nhanh

Chương 3: Mục lục

- 3.1 Các dịch vụ tầng-truyền tải
- 3.2 Sự dồn và tách
- 3.3 Sự truyền tải không kết nối: UDP
- 3.4 Sự truyền tải hướng kết nối : TCP
  - cấu trúc đoạn tin
  - truyền tải dự liệu tin cậy
  - kiểm soát lưu lượng
  - quản lý kết nối
- 3.5 Các nguyên lý của kiểm soát tắc nghẽn
- 3.6 Kiểm soát tắc nghẽn trong TCP

Kiểm soát lưu lượng trong TCP

- phía nhận của receive side of TCP connection has a receive buffer:



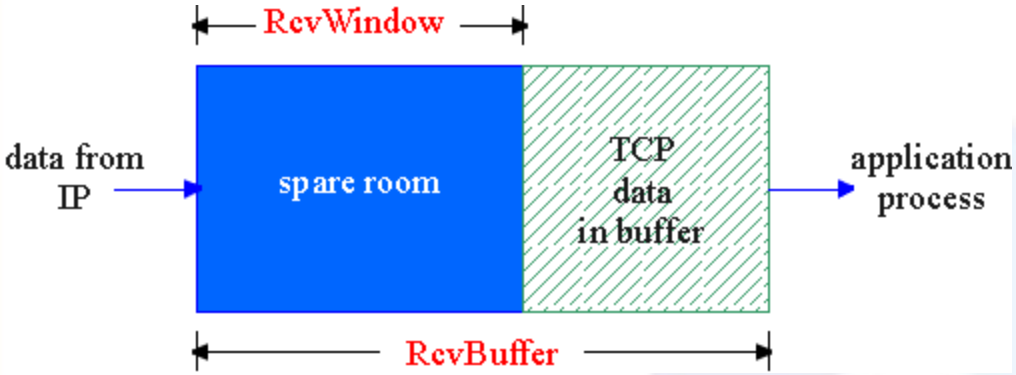
- tiến trình ứ/d có thể chậm trong việc đọc từ bộ nhớ tạm

kiểm soát LL

ng/gửi không làm tràn bộ nhớ tạm của ng/nhận bởi truyền quá nhanh và nhiều

- dịch vụ làm tương đồng tốc độ: điều chỉnh tốc độ gửi sao cho phù hợp với tốc độ đọc của tiến trình nhận

KSLL trong TCP làm việc ntn?



(Giả sử ng/nhận TCP loại bỏ các khúc không-đúng-thứ-tự)

- số chỗ trống trong bnt

= **RcvWindow**

= **RcvBuffer-[LastByteRcvd - LastByteRead]**

Ng/nhận thông báo số chỗ trống trong bnt bằng cách thêm giá trị cửa sổ nhận **RcvWindow** trong khúc tin

- Ng/gửi hạn chế lượng dữ liệu chưa ACK tới giá trị của **RcvWindow**
  - đảm bảo bộ nhớ tạm của người nhận kô bao giờ bị tràn

Chương 3: Mục lục

- 3.1 Các dịch vụ tầng-truyền tải
- 3.2 Sự dồn và tách
- 3.3 Sự truyền tải không kết nối: UDP
- 3.4 Sự truyền tải hướng kết nối : TCP
  - cấu trúc đoạn tin
  - truyền tải dự liệu tin cậy
  - kiểm soát lưu lượng
  - quản lý kết nối
- 3.5 Các nguyên lý của kiểm soát tắc nghẽn
- 3.6 Kiểm soát tắc nghẽn trong TCP

Quản lý kết nối TCP

Gợi nhớ: ng/gửi, ng/nhận TCP thiết lập "kết nối" trước khi trao đổi các khúc dữ liệu

- khởi tạo các biến TCP:
  - STT
  - BNT, thông tin KSLL (vd: **`RcvWindow`**)
- *khách:* người bắt đầu kết nối

```
Socket clientSocket = new
 Socket("hostname","port

 number");
```

- *chủ:* được liên lạc bởi khách

```
Socket connectionSocket =
 welcomeSocket.accept();
```

Bắt tay 3 bước:

Bước 1: máy khách gửi khúc TCP SYN tới máy chủ
- chứa STT ban đầu
- không có dữ liệu

Bước 2: chủ nhận được SYN, gửi trả lại một khúc SYNACK
- chủ cấp bộ nhớ tạm
- STT ban đầu của chủ

Bước 3: khách nhận được SYNACK, phản hồi lại với khúc ACK, có thể kèm theo dữ liệu

Quản lý kết nối TCP (tt)

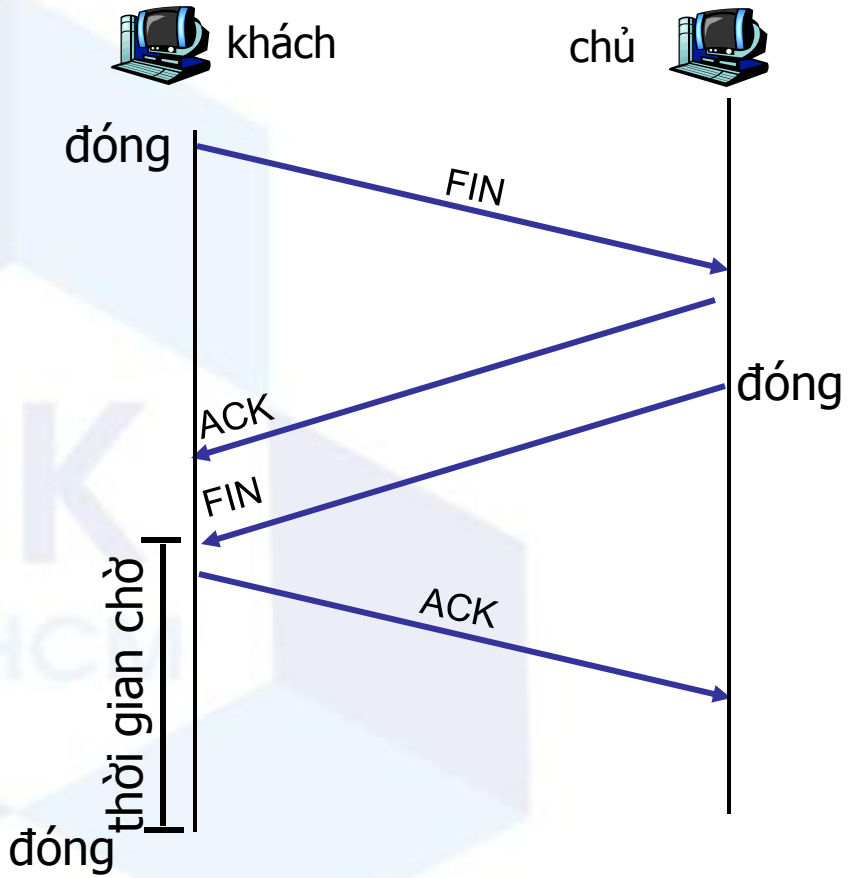
Đóng một kết nối:

khách đóng socket:

```
clientSocket.close();
```

Bước 1: khách gửi một khúc điều khiển TCP FIN đến chủ

Bước 2: chủ nhận được FIN, phản hồi với ACK. Đóng kết nối, gửi FIN.



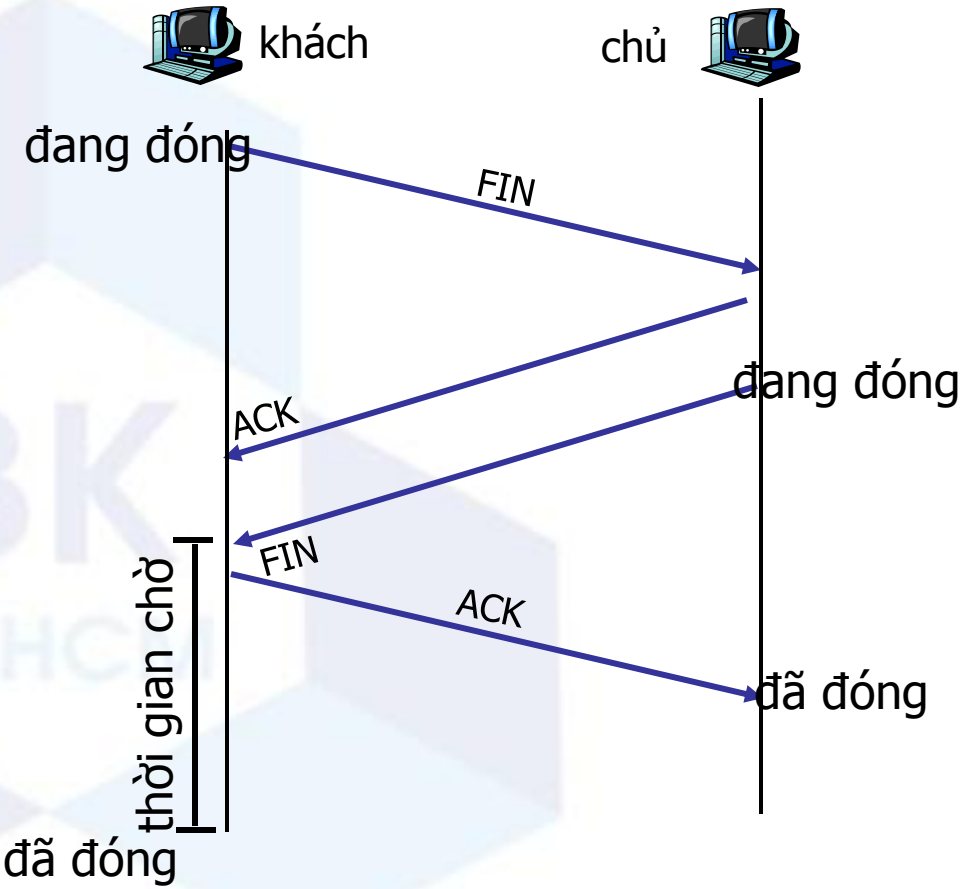
Quản lý kết nối TCP (tt)

Bước 3: khách nhận được FIN, phản hồi với ACK.

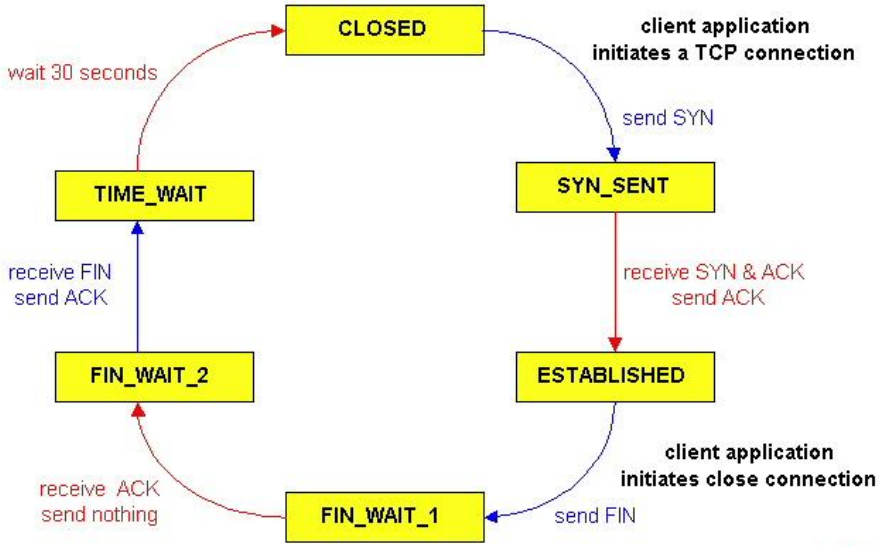
- Bước vào trạng thái "chờ có đếm thời gian" – sẽ phản hồi bằng ACK với những FIN nhận được

Step 4: chủ, nhận được ACK. Đóng kết nối.

Ghi chú: với vài chỉnh sửa nhỏ, có thể xử lý nhiều FIN đồng thời.

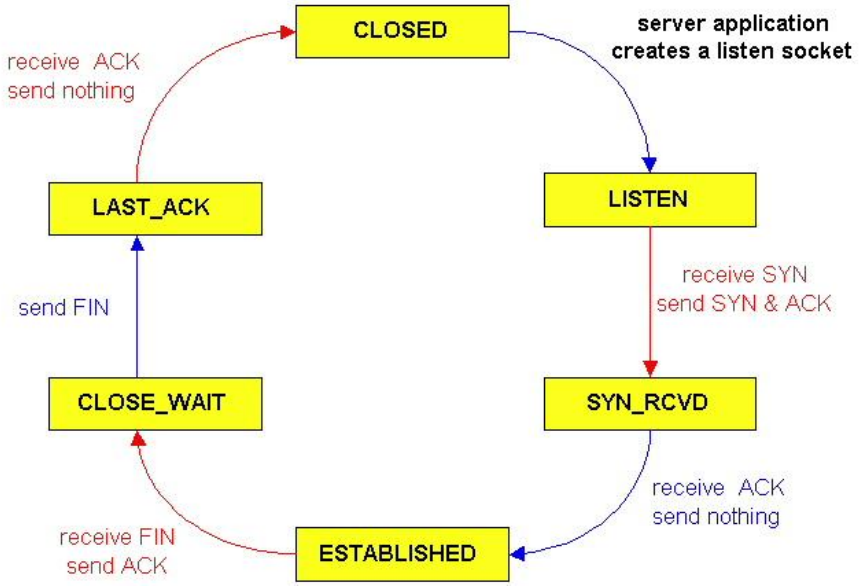


Quản lý kết nối TCP (tt)



chu kì sống của khách TCP

chu kì sống của chủ TCP



Chương 3: Mục lục

- 3.1 Các dịch vụ tầng-truyền tải
- 3.2 Sự dồn và tách
- 3.3 Sự truyền tải không kết nối: UDP
- 3.4 Sự truyền tải hướng kết nối : TCP
  - cấu trúc đoạn tin
  - truyền tải dự liệu tin cậy
  - kiểm soát lưu lượng
  - quản lý kết nối
- 3.5 Các nguyên lý của kiểm soát tắc nghẽn
- 3.6 Kiểm soát tắc nghẽn trong TCP

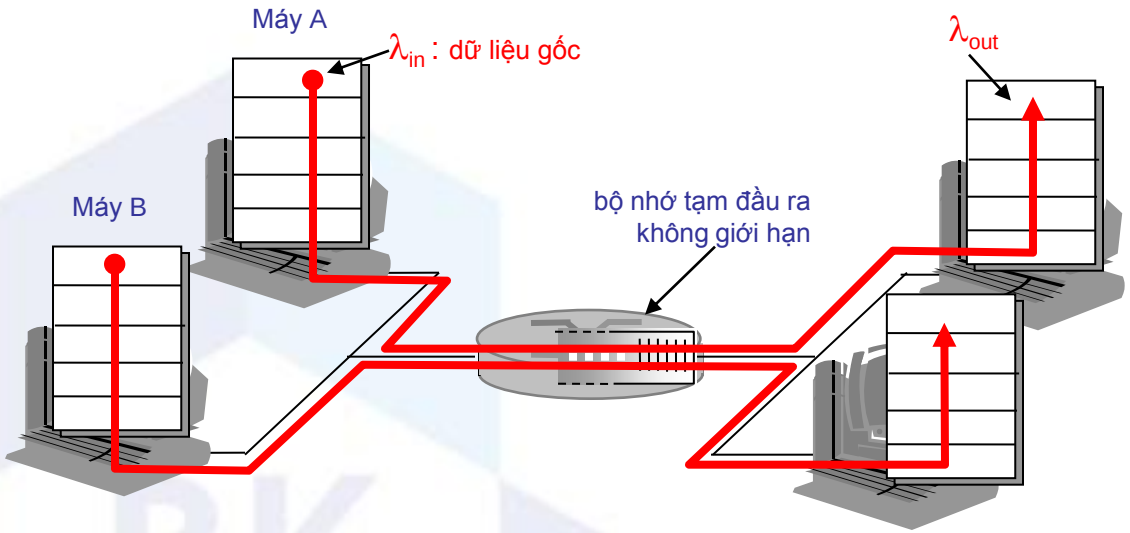
Các nguyên tác kiểm soát tắc nghẽn

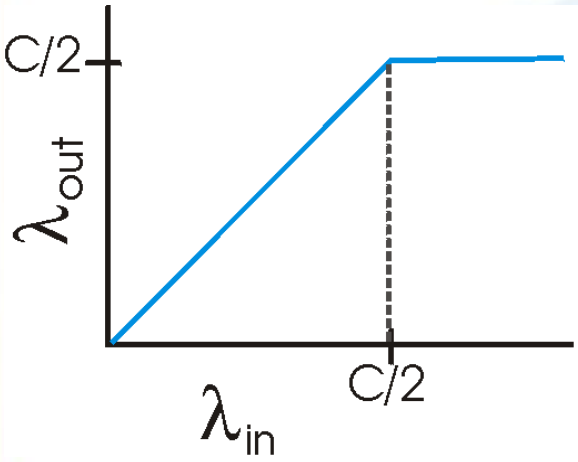
Tắc nghẽn:

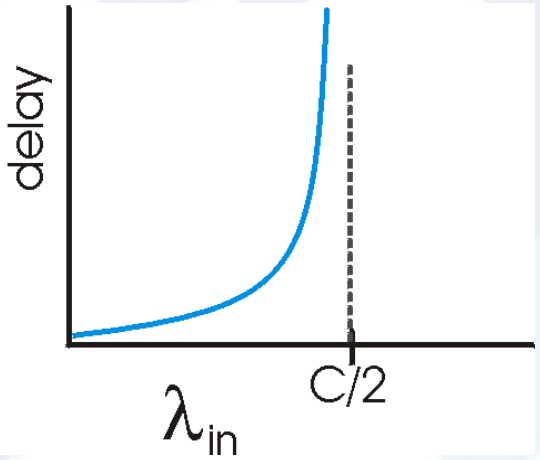
- phát biểu đơn giản: "quá nhiều nguồn gửi quá nhiều dữ liệu quá nhanh để *mạng* có thể xử lý"
- khác với kiểm soát lưu lượng!
- biểu hiện:
  - mất gói tin (tràn bộ nhớ tạmbuffer tại bđt)
  - độ trễ lâu (xếp hàng trong bộ nhớ tạm bđt)
- là một trong 10 vấn đề quan trọng của Internet!

Nguyên nhân/thiệt hại của tắc nghẽn: 1

- hai ng/gửi, hai ng/nhận
- một bđt, bộ nhớ tạm không giới hạn
- không truyền tải lại



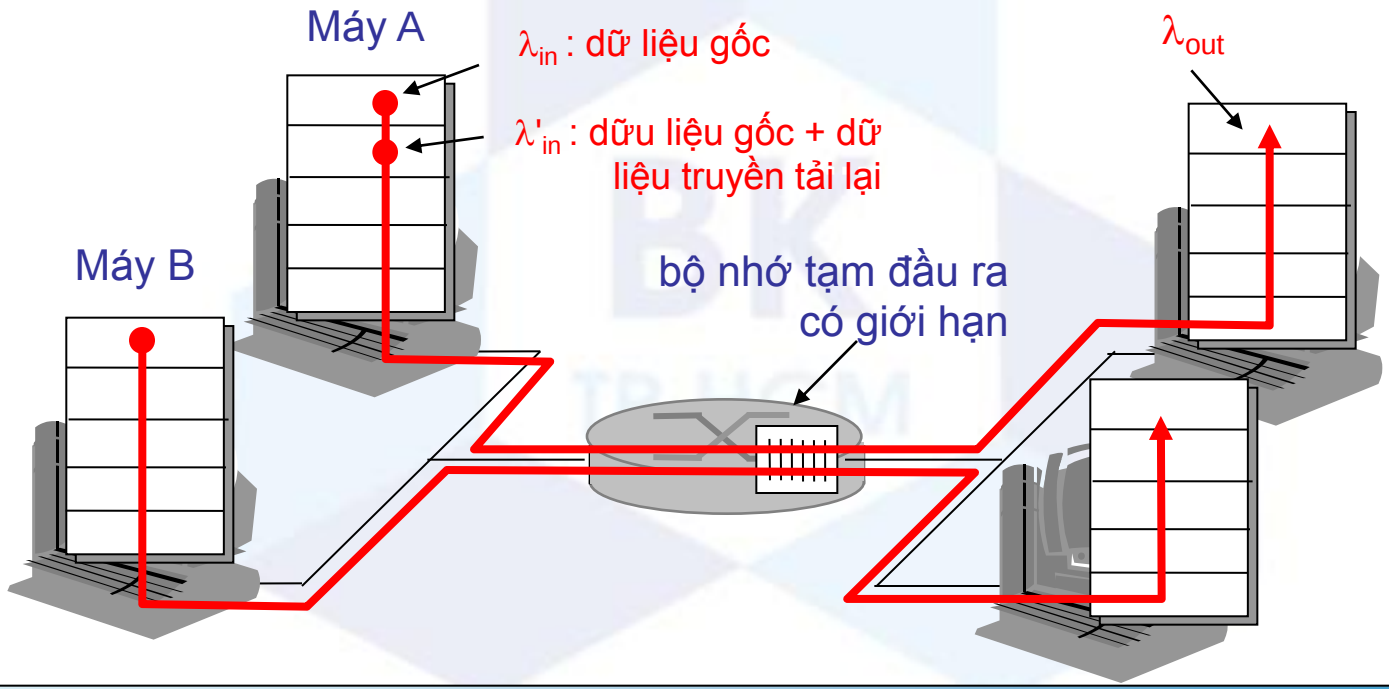




- độ trễ lớn khi tắc nghẽn
- đạt được thông lượng tối đa

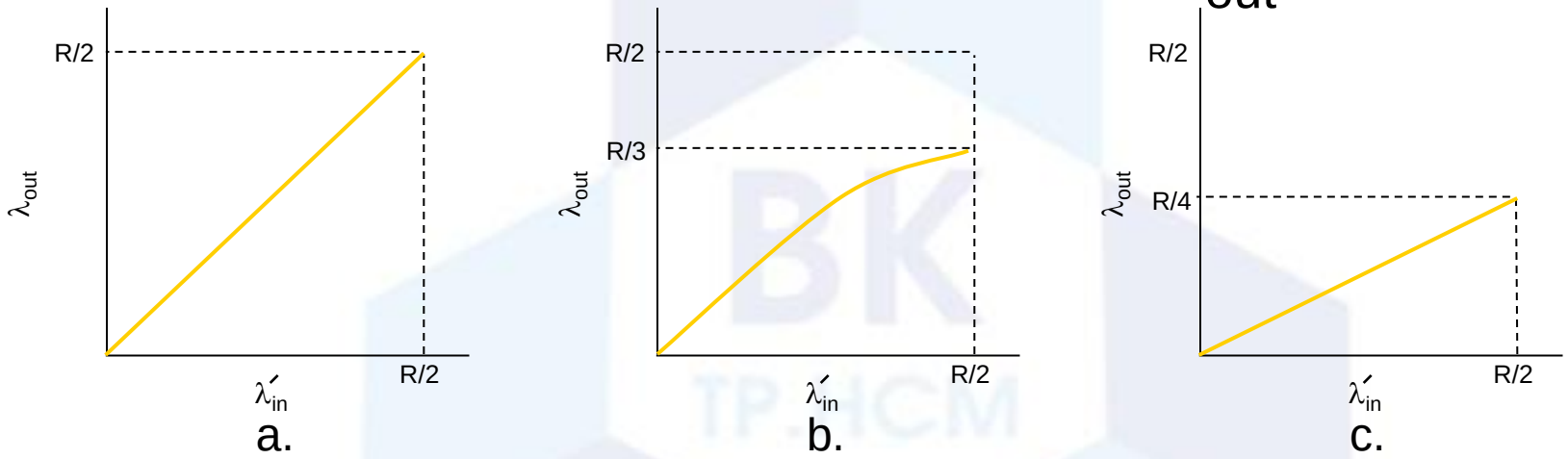
Nguyên nhân/thiệt hại của tắc nghẽn: 2

- một bđt, bộ nhớ tạm *có giới hạn*
- người gửi truyền tải lại những gói bị mất



nguyên nhân/thiệt hại của tắc nghẽn: 3

- luôn luôn: $\lambda_{in} = \lambda_{out}$
- truyền lại "tối ưu" chỉ khi có mất mát: $\lambda'_{in} > \lambda_{out}$
- sự truyền lạiretransmission của các gói trễ (không mất) làm cho λ'_{in} lớn hơn (so với tr/hợp tối ưu) với cùng một λ_{out}



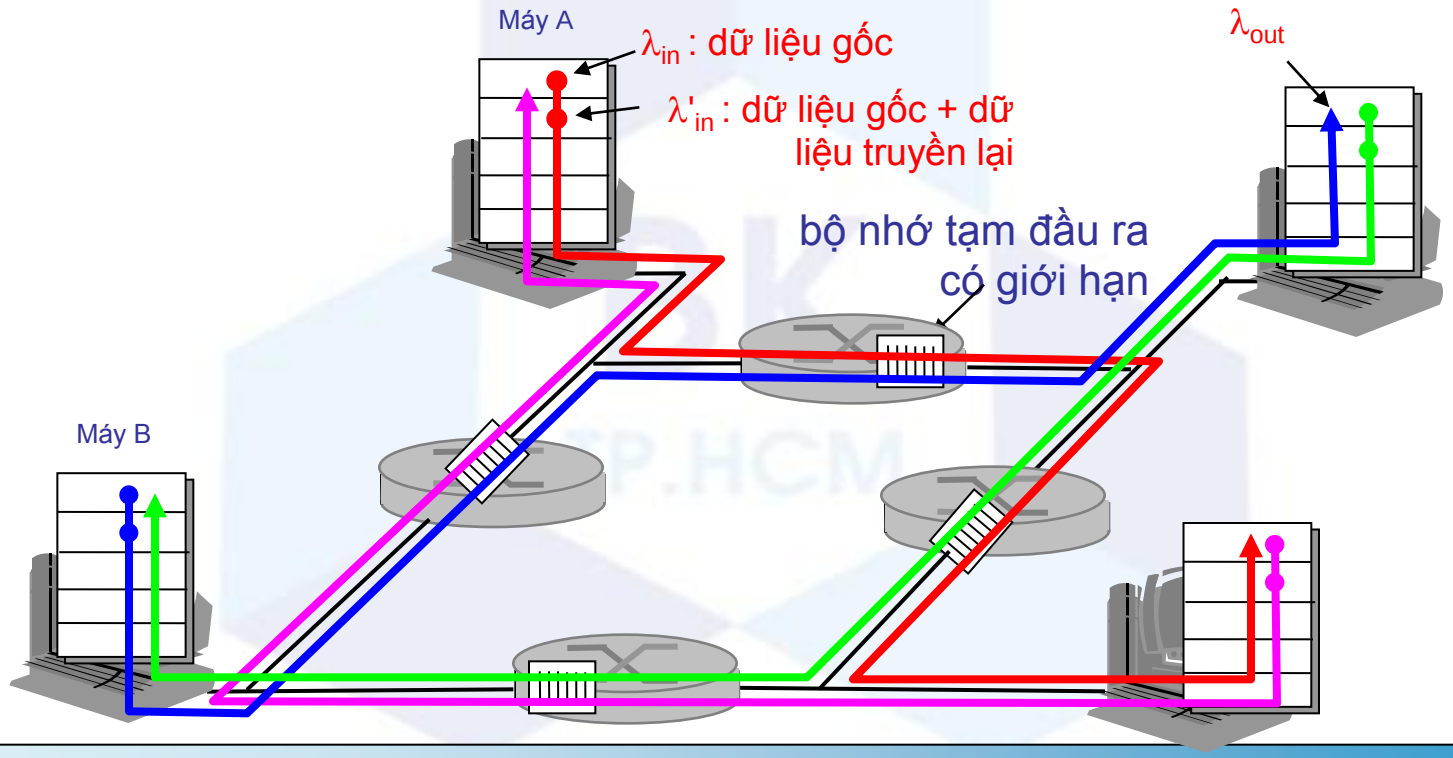
"thiệt hại" của tắc nghẽn:

- phải truyền lại khi mà gói tin bị loại do tràn bộ nhớ tạm tại bđt
- sự truyền tải lại ko cần thiết: đường kết nối chứa nhiều bản sao của gói tin

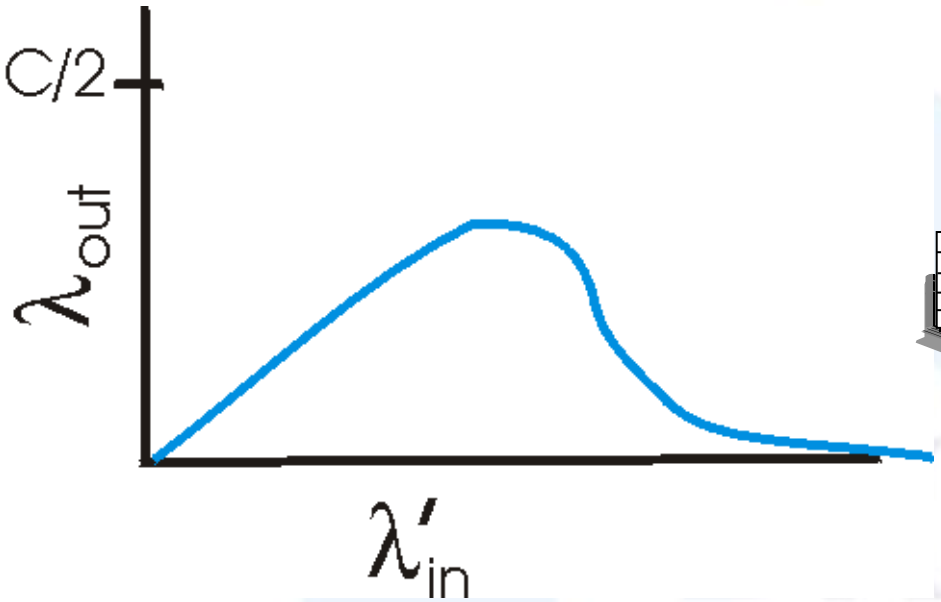
nguyên nhân/thiệt hại của tắc nghẽn: 4

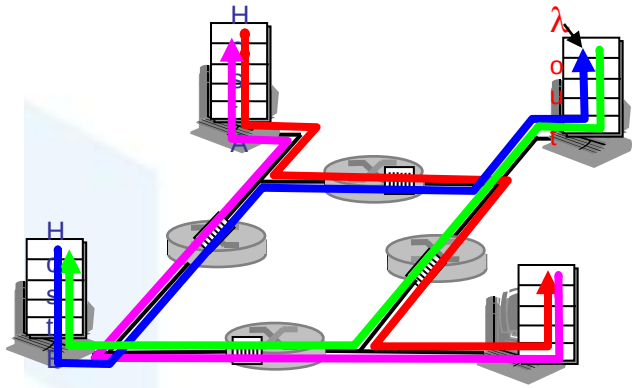
- bốn người gửi
- đường đi qua nhiều bước
- thời-gian-chờ/truyền-tải-lại

Hỏi: chuyện gì xảy ra khi λ_{in} và λ'_{in} tăng lên ?



nguyên nhân/thiệt hại của tắc nghẽn: 5





Một thiệt hại khác của tắc nghẽn:

- khi gói tin bị loại bỏ, tất cả băng thông dùng để truyền tải nó tới điểm mà nó bị loại bỏ là phí phạm!

Các phương án tiếp cận đối với kiểm soát tắc nghẽn

Hai phương án tiếp cận rộng:

kiểm soát tắc nghẽn đầu cuối-đầu cuối:

- không có phản hồi rõ ràng từ mạng
- tắc nghẽn được cho là xảy ra nếu máy đầu cuối phát hiện có mất gói, trễ
- pp tiếp cận này được sử dụng bởi TCP

kiểm soát tắc nghẽn được hỗ trợ từ mạng:

- các BĐT cung cấp phản hồi cho máy đầu cuối
  - một bit báo hiệu tắc nghẽn (SNA, DECbit, TCP/IP ECN, ATM)
  - tốc độ cụ thể mà người gửi nên dùng

Ví dụ nghiên cứu: kiểm soát tắc nghẽn ATM ABR

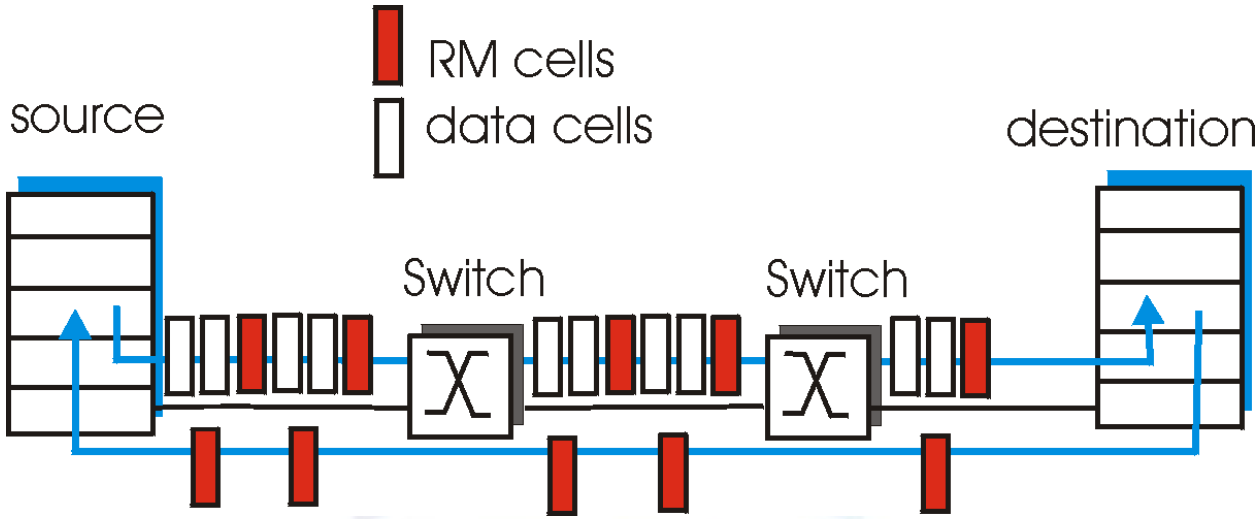
ABR: tốc độ bit cho phép:

- "dịch vụ mềm dẻo"
- nếu đường truyền của ng/gửi "chưa hết tải":
  - người gửi nên sử dụng băng thông còn dư
- nếu đường truyền của ng/gửi bị tắc nghẽn:
  - ng/gửi giảm xuống tốc độ đảm bảo tối thiểu

ô RM (quản lý tài nguyên) :

- gửi bởi ng/gửi, chen lẫn với các ô dữ liệu
- bits trong ô RM được thiết lập bởi các bộ chuyển mạch ("được *hỗ trợ từ mạng*")
  - NI bit: ko tăng tốc (tắc nghẽn nhẹ)
  - CI bit: biểu hiện tắc nghẽn (nặng)
- các ô RM được gửi lại cho ng/gửi bởi ng/nhận mà ko có thay đổi gì

Ví dụ nghiên cứu: kiểm soát tắc nghẽn ATM ABR



- trường ER 2-byte (tốc độ cụ thể) trong ô RM
  - BCM tắc nghẽn có thể giảm giá trị ER trong ô RM
  - ER sẽ được thiết lập bằng với tốc độ hỗ trợ tối thiểu của tất cả BCM trên đường đi từ nguồn-tới-đích
- bit EFCI trong ô dữ liệu: được đặt là 1 trong BCM tắc nghẽn
  - nếu ô dữ liệu tới trước ô RM chứa bit EFCI bật, người gửi bật bit CI trong ô RM rồi gửi lại

Chương 3: Mục lục

- 3.1 Các dịch vụ tầng-truyền tải
- 3.2 Sự dồn và tách
- 3.3 Sự truyền tải không kết nối: UDP
- 3.4 Sự truyền tải hướng kết nối : TCP
  - cấu trúc đoạn tin
  - truyền tải dự liệu tin cậy
  - kiểm soát lưu lượng
  - quản lý kết nối
- 3.5 Các nguyên lý của kiểm soát tắc nghẽn
- 3.6 Kiểm soát tắc nghẽn trong TCP

KSTN TCP: tăng hệ số cộng, giảm hệ số nhân

- *P/pháp:* tăng tốc độ truyền tải (kích thước cửa sổ), thử băng thông khả dụng, tới khi xuất hiện mất gói
  - *tăng hs cộng:* tăng **CongWin** lên 1 MSS mỗi RTT đến khi phát hiện mất gói
  - *giảm hs nhân*: giảm **CongWin** xuống ½ sau khi mất gói

Hình răng cưa: thăm dò băng thông

KSTN TCP: chi tiết

- ng/gửi hạn chế truyền tải:
  `LastByteSent-LastByteAcked`
  `≤ CongWin`

- hay,

$$\text{vận tốc} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- **`CongWin`** là một hàm phụ thuộc vào sự tắc nghẽn của mạng

Làm sao ng/gửi nhận ra sự tắc nghẽn?

- mất gói = hết t/g chờ *hoặc* 3 ack trùng
- ng/gửi TCP giảm vận tốc (**`CongWin`**) sau khi có mất gói

ba cơ chế:

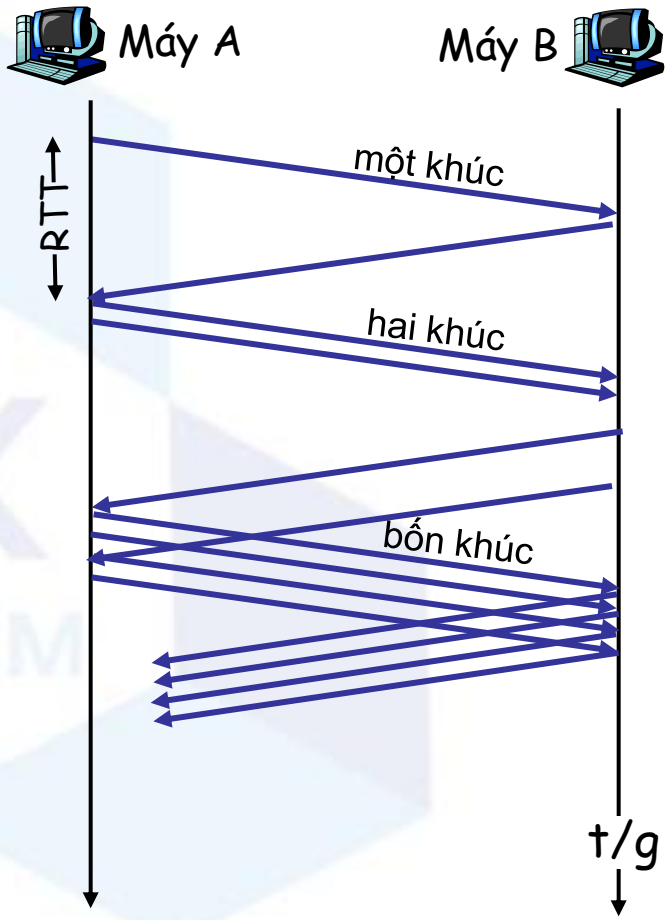
- AIMD
- bắt đầu chậm
- giữ nhịp độ tăng tốc độ sau khi mất gói

TCP Bắt đầu chậm

- Khi kết nối bắt đầu, **`CongWin`** = 1 MSS
  - Vd: MSS = 500 bytes & RTT = 200 msec
  - vận tốc ban đầu = 20 kbps
- băng thông cho phép có thể >> MSS/RTT
  - mong muốn tăng nhanh lên đến vận tốc cao nhất cho phép
- Khi kết nối bắt đầu, tăng vận tốc theo hệ số mũ đến khi xuất hiện mất gói

TCP Bắt đầu chậm (tt)

- Khi kết nối bắt đầu, tăng vận tốc theo hệ số mũ đến khi xuất hiện mất gói :
  - nhân đôi **`CongWin`** mỗi RTT
  - thực hiện bởi tăng lên 1 **`CongWin`** cho mỗi ACK nhận được
- Tóm lại: bận tốc ban đầu chậm nhưng tăng lên nhanh theo hàm mũ



Cải thiện: phỏng đoán mất gói

- Nếu nhận được 3 ACK trùng:
  - **`CongWin`** giảm ½
  - sau đó tăng tuyến tính
- Nhưng nếu xảy ra "hết t/g chờ":
  - **`CongWin`** = 1 MSS;
  - tăng theo hàm mũ
  - tăng tới ngưỡng cuối cùng trước khi mất gói, sau đó tăng tuyến tính

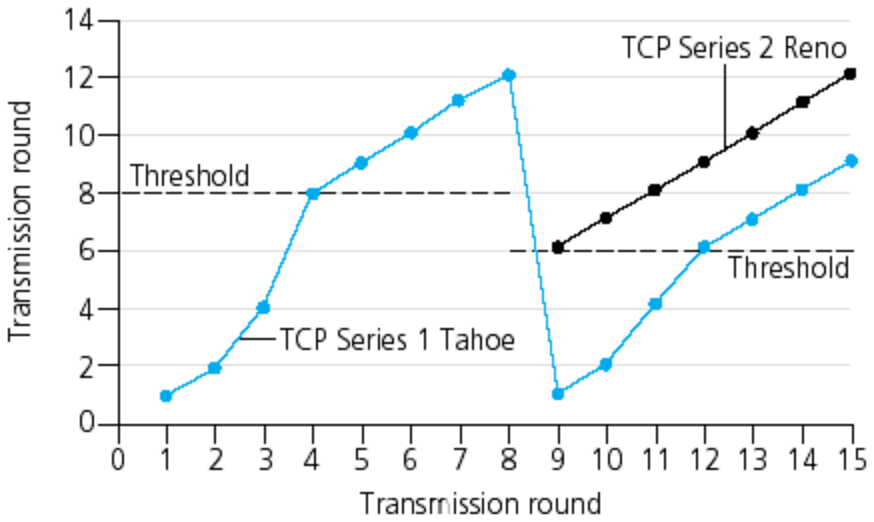
Triết lí:

- 3 ACK lặp nghĩa là mạng có khả năng phân phối vài khúc dữ liệu
- "hết t/g chờ" cho biết tình hình tắc nghẽn đáng báo động hơn

Cải tiến

Hỏi: Khi nào thì nên chuyển từ tăng hàm mũ sang tăng tuyến tính?

A: Khi **CongWin** đạt được ½ giá trị của nó trước khi xảy ra "hết t/g chờ"



Hiện thực:

- Giá trị ngưỡng biến thiên
- Khi mất gói, g/t ngưỡng được gán bằng ½ của CongWin ngay trước khi xảy ra mất gói

Tóm tắt: KSTN TCP

- Khi **`CongWin`** nhỏ hơn **`Threshold`**, ng/gửi ở pha bắt-đầu-chậm, cửa sổ tăng theo số mũ.
- Khi **`CongWin`** lớn hơn **`Threshold`**, ng/gửi trong pha tránh-tắc-nghẽ, cửa sổ tăng tuyến tính.
- Khi xảy ra lặp 3 ACK, **`Threshold`** gán bằng **`CongWin/2`** và **`CongWin`** gán bằng **`Threshold`**.
- Khi hết-t/g-chờ, **`Threshold`** gán bằng **`CongWin/2`** và **`CongWin`** gán bằng 1 MSS.

KSTN ng/gửi TCP

Trạng thái	Sự kiện	Hành vi ng/gửi TCP	Bình luận
Bắt đầu chậm (SS)	nhận được ACK cho những dữ liệu chưa ack	CongWin = CongWin + MSS, If (CongWin > Threshold) chuyển trạng thái sang "CA"	Nhân đôi CongWin mỗi RTT
Tránh tắc nghẽn (CA)	nhận được ACK cho những dữ liệu chưa ack	CongWin = CongWin+MSS * (MSS/CongWin)	Tăng theo cấp số cộng, dẫn đến tăng CongWin lên 1 MSS mỗi RTT
SS hoặc CA	Phát hiện mất gói do có "trùng 3 ACK"	Threshold = CongWin/2, CongWin = Threshold, chuyển trạng thái sang "CA"	Hồi phục nhanh, sử dụng giảm theo hệ số nhân. CongWin không giảm nhỏ hơn 1 MSS.
SS hoặc CA	Hết t/g chờ	Threshold = CongWin/2, CongWin = 1 MSS, chuyển trạng thái "SS"	Chuyển sang SS
SS hoặc CA	Lặp ACK	Tăng biến đếm số ACK lặp cho khúc được ACK	CongWin và Threshold không thay đổi

Thông lượng TCP

- Thông lượng trung bình của TCP như là hàm số của k/t cửa sổ và RTT là bao nhiêu?
  - bỏ qua bắt-đầu-chậm
- Xem W là k/t cửa sổ khi xuất hiện mất gói.
- Khi cửa sổ là W, thông lượng là W/RTT
- Sau khi mất gói, cửa sổ giảm xuống còn W/2, thông lượng xuống W/2RTT.
- Thông lượng trung bình: .75 W/RTT

Tương lai TCP: TCP qua các "đường ống dài, rộng"

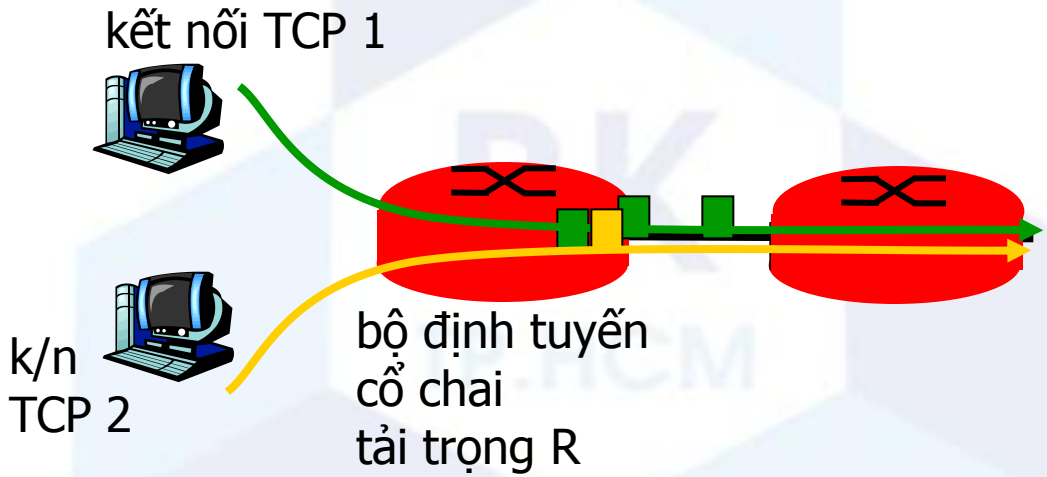
- Ví dụ: khúc 1500 byte, RTT 100ms, thông lượng cần có 10 Gbps
- Yêu cầu kích thước cửa sổ W = 83,333
- Thông lượng trong giới hạn của tần số mất gói:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ $L = 2 \cdot 10^{-10}$ *Vô cùng nhỏ*
- Các phiên bản TCP mới cho đường truyền tốc độ cao

Sự công bằng trong TCP

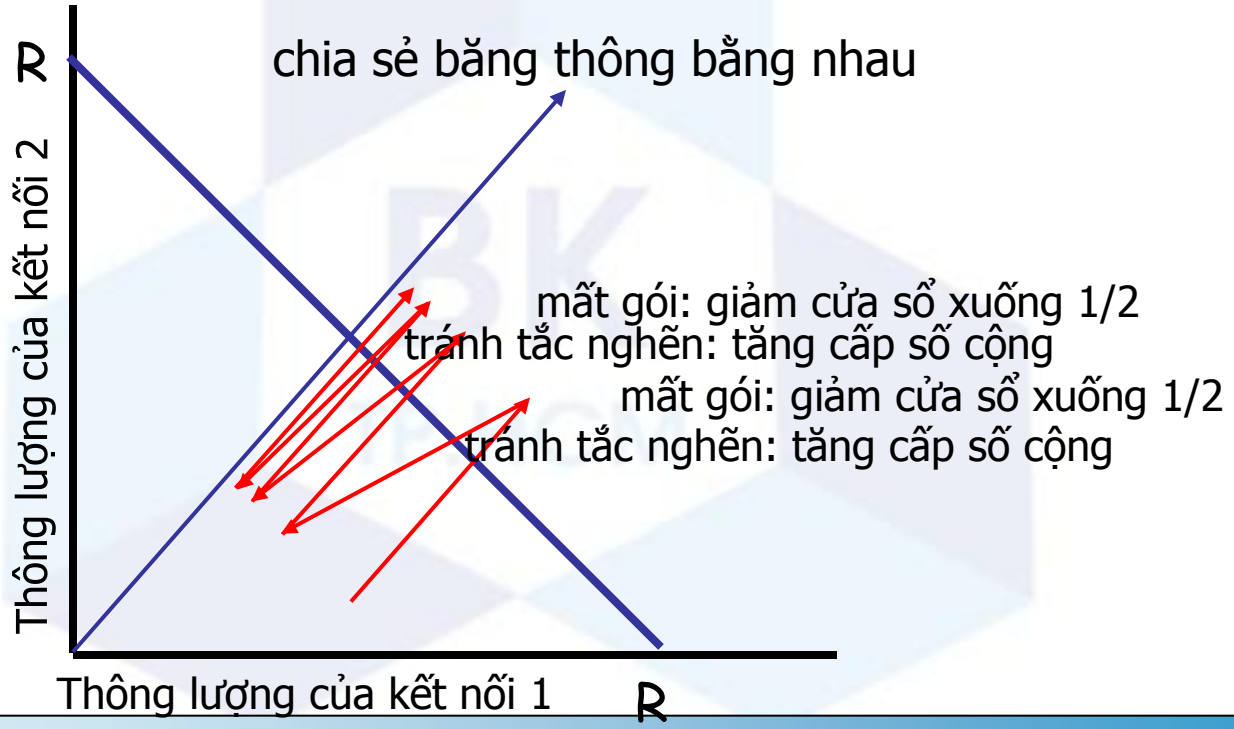
Mục đích của sự công bằng: nếu K phiên TCP chia sẽ một đường kết nối cổ chai với băng thông R, mỗi phiên phải có được vận tốc trung bình là R/K



Tại sao TCP lại công bằng?

Hai kịch bản cạnh tranh:

- Tăng cấp số cộng tạo độ dốc 1, as throughout increases
- Giảm theo cấp số nhân giảm thông lượng một cách cân xứng



Tính công bằng (tt)

Tính công bằng và UDP

- Các ứ/d đa phương tiện thường không dùng TCP
  - không muốn tốc độ bị giới hạn bởi quá trình KSTN
- Thay vào đó dùng UDP:
  - gửi âm thanh/phim ảnh ở một vận tốc cố định, chấp nhận mất gói
- Lĩnh vực nghiên cứu: UDP tương tự như TCP

Sự công bằng và các kết nối TCP song song

- ko thể cấm ứ/d mở những kết nối song song giữa 2 máy.
- Trình duyệt Web là một ví dụ
- VD: liên kết với vận tốc R hỗ trợ 9 kết nối;
  - ứ/d mới yêu cầu 1 TCP, có tốc độ R/10
  - ứ/d khác yêu cầu11 TCPs, có tốc độ R/2 !

Chương 3: Tổng kết

- Các nguyên lý đằng sau các dịch vụ tầng truyền tải:
  - dồn, tách
  - truyền tải dữ liệu tin cậy
  - kiểm soát lưu lượng
  - kiểm soát tắc nghẽn
- Thuyết minh và hiện thực trong Internet
  - UDP
  - TCP

Tiếp theo:

- chúng ta rời "ngoại vi mạng" (ứ/dụng, tầng truyền tải)
- đi vào "hạt nhân" mạng