UIT
Trường Đại học
Công nghệ Thông tin

Khoa Khoa học
và Kỹ thuật Thông tin

NHẬP MÔN MẠNG MÁY TÍNH

Chương 3: Tầng Transport

I. Tầng vận chuyển:

- Cung cấp truyền thông logic **giữa các tiến trình** ứng dụng đang chạy trên các host khác nhau
- PDU (Protocol data unit): **Segment**
- Quan hệ giữa Tầng Vận chuyển và tầng Mạng:
+ Tầng Mạng: truyền thông logic giữa các host (dùng các giao thức dịch vụ HTTP, SMTP,…)
+ Tầng Vận chuyển: truyền thông logic giữa các tiến trình (process). Dựa trên dịch vụ tầng
mạng.
- Các giao thức được sử dụng: UDP và TCP
- So sánh TCP và UDP:

TCP:
+ Có cơ chế truyền tin cậy (không mất gói), theo thứ tự
+ Có cơ chế phân luồng (bên gửi không làm quá tải bên nhận)
+ Full duplex: Truyền và nhận cùng lúc
+ Gửi đi dạng luồng byte
+ Có cơ chế bắt tay (Connection – oriented)
+ Giải quyết được tắc nghẽn
+ Dùng cho công việc cần bảo đảm dữ liệu: mail, web, file
*** Chậm hơn UDP (thường do cơ chế sữa lỗi)**

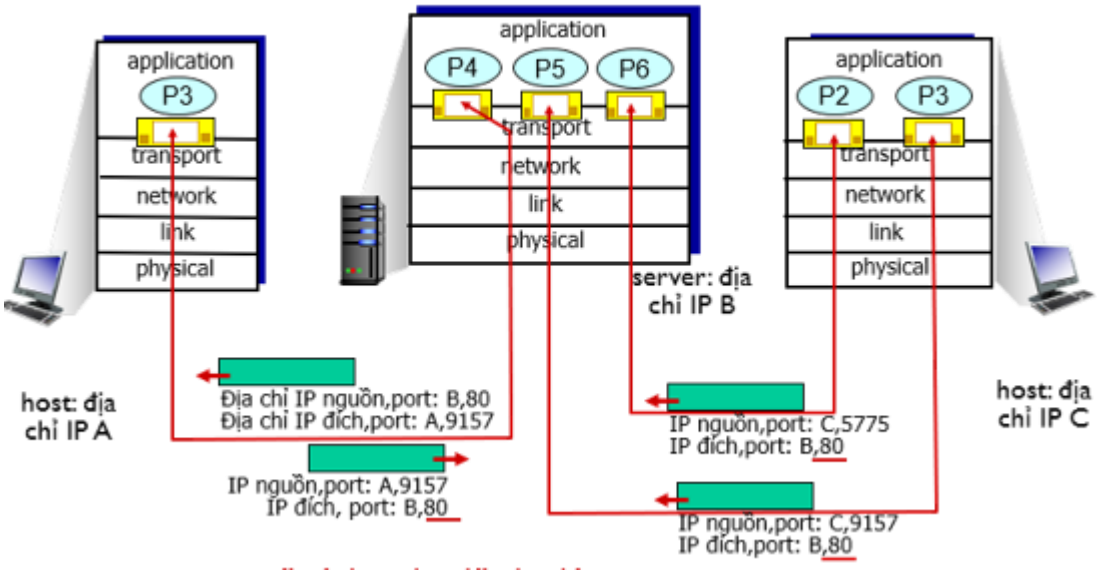
UDP:
+ Gói tin gọn
+ Dùng cho công việc cần tốc độ, băng thông tối thiểu: game online, video call, stream
+ Gửi đi dạng datagram
*** Nhanh hơn TCP**
*** Truyền không tin cậy**

II. Multiplexing và Demultiplexing:

+ Multiplexing: tổng hợp data chunks từ socket lại thành
segment, đánh header phục vụ demultiplexing (encapsulation)
+ Demultiplexing: phân các segment về đúng socket dựa
trên những trường dữ liệu nằm trên segment (địa chỉ port và địa chỉ IP)

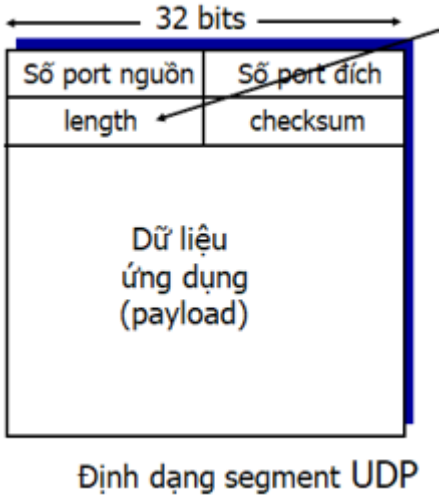
* UDP socket được định danh bằng 2 thông tin: **IP đích và port đích**
* Khác biệt giữa socket TCP và UDP là socket TCP sử dụng định danh bằng **bộ bốn thông tin**: địa chỉ IP nguồn, cổng nguồn, địa chỉ IP đích đến, cổng đích đến. (Cả 4 thông tin mới xác định được 1 socket)

Khi một host nhận một phân đoạn từ tầng mạng, host sử dụng cả bốn thông tin này để chuyển (phân kênh) phân đoạn đó về socket tương ứng. Ngược với UDP, hai phân đoạn TCP khác nhau IP nguồn hoặc cổng nguồn sẽ được chuyển đến hai socket khác nhau.
* HTTP không bền vững sẽ có socket khác nhau cho mỗi yêu cầu



Hình 1: Chuyển các segment về đúng socket

III. Giao thức UDP:



Định dạng segment UDP

+ Tổng kích thước header: 8 bytes

+ Length: Tính cả dung lượng header lẫn payload
+ Checksum: Kiểm tra lỗi gói tin. Tính bằng nội dung cả header lẫn payload
+ Cách tính checksum: Tính tổng bù 1 các đoạn 16 bits trong segment
+ Kích thước lớn nhất của UDP segment: 2^{16} = 65536 bytes

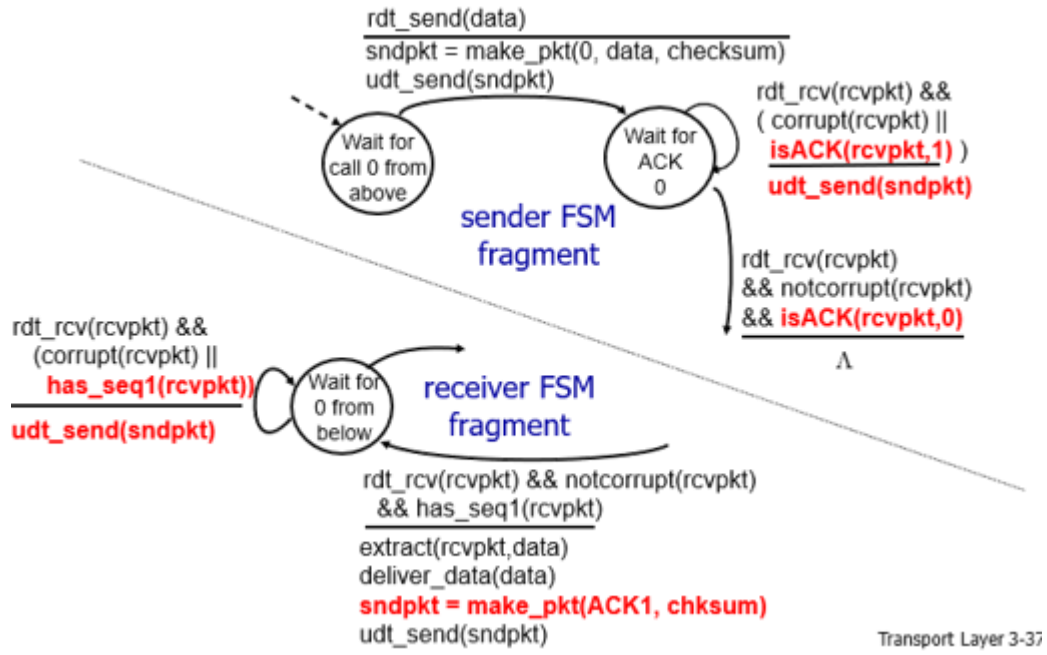
IV. Các nguyên lí truyền dữ liệu tin cậy:

- **RTD**: Reliable Transmission Data

Tên	Đặc điểm	Bên gửi	Bên nhận
rtd 1.0	Truyền dữ liệu tin cậy. (Giả sử không có lỗi hay mất mát gì)	Gửi gói tin	Nhận gói tin
rtd 2.0	Kênh truyền không làm mất gói. (Nhưng gói tin truyền có thể bị sai sót)	Gửi gói, đợi phản hồi (Stop and Wait Protocol)	Dùng checksum, ACK (Acknowledgement) và NAK (Negative ACK) để check lỗi gói tin gửi về sender.
rtd 2.1	Kênh truyền có ACK, NAK bị lỗi (ACK, NAK bị sai checksum)	Đánh số thứ tự 0, 1 cho các gói từ sender, gửi lại gói nếu NAK hoặc ACK/NAK bị lỗi (nhận biết bằng checksum trong gói), nhận được ACK thì mới gởi gói đánh số tiếp theo	Có cơ chế loại packet bị trùng
rtd 2.2	Không dùng NAK, thay bằng gửi ACK gói gần nhất nhận thành công	Gửi lại gói nếu nhận ACK trùng lặp	Gửi ACK gói gần nhất thành công, đánh số thứ tự 0,1 cho ACK
rtd 3.0 => Hiệu suất thấp: Usender = (L/R)/(RTT+L/R) do phải nhận	Xuất hiện mất gói	- Chờ ACK trong khoảng thời gian “Hợp lí”. Dùng Timeout (Bộ định	

ACK rồi mới gửi gói tiếp được		thì). Gửi lại pckt khi hết thời gian. - Các trường hợp gởi lại gói: Mất gói, mất ACK, thời gian chờ ngắn/ delayed ACK	

* Nhớ đọc thêm các FSM (Finite State Machine) để hiểu rõ hơn các bước làm việc nhé.

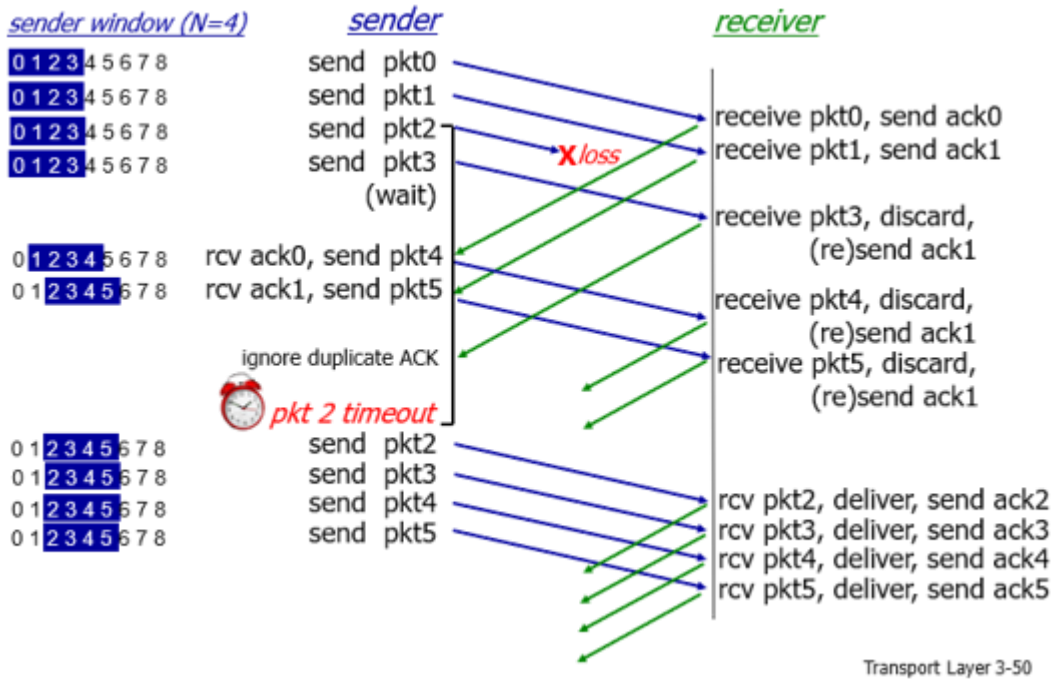


Hình 2: Ví dụ về FSM rdt 2.0
** Cần quan tâm các đối tượng:*
Trạng thái (state): Các vòng tròn
Mũi tên: Việc chuyển trạng thái
Điều kiện (condition): Phần trên dấu gạch ngang
Hành động (action): Phần dưới dấu gạch ngang

- **Giao thức pipelined:** Cải tiến của rdt 3.0, cho phép gửi nhiều gói tin cùng lúc không cần đợi ACK và nhận nhiều phản hồi cùng lúc, được chia làm 2 loại là Go-Back-N và Seletive Repeat.
- Các cải tiến:
+ Dải số thứ tự phải được tăng lên.

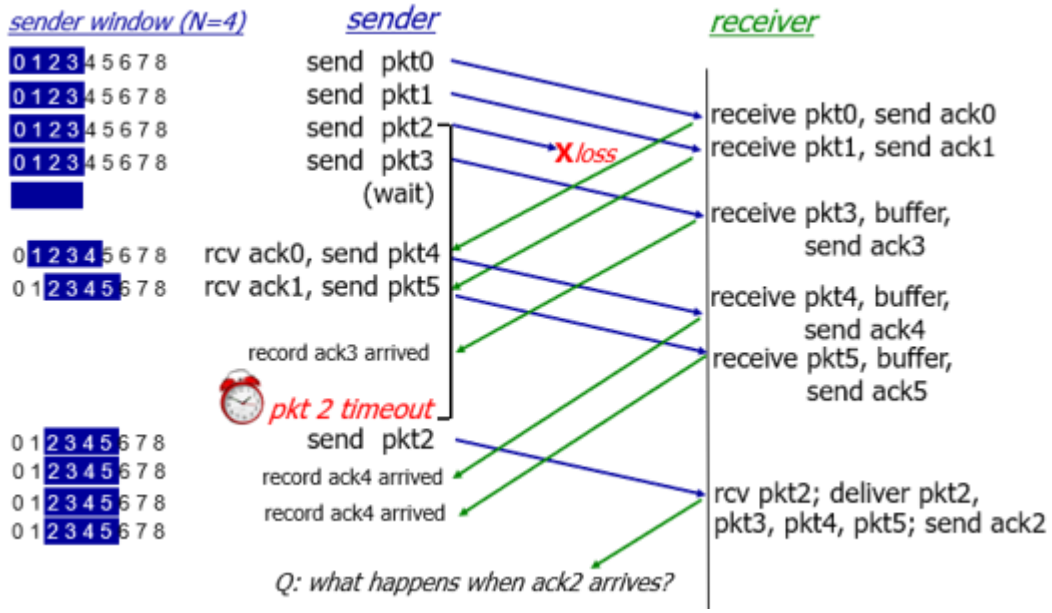
+ Phải có bộ nhớ đệm tại nơi gửi và/hoặc nhận
+ Kỹ thuật này có lợi khi lượng dữ liệu cần truyền rất lớn, và gửi dữ liệu bằng cách chia chúng thành nhiều phần khác nhau.

a. Go-Back-N:



- Các điểm cần nhớ:
+ Bên nhận không có bộ nhớ đệm, phải nhận đúng thứ tự các gói, nếu nhận sai thứ tự, hủy gói vừa nhận, gửi ACK của gói gần nhất nhận được thành công.
VD: Trong ảnh, pkt3, pkt4, pkt5 nhận là sai thứ tự nên bị hủy, gửi lại ACK gói gần nhất nhận được thành công là pkt1
+ Bên gửi có cơ chế không quan tâm ACK trùng. Gửi lại toàn bộ pckt sau ACK gần nhất nhận được
VD: Sau khi pkt 2 gửi không nhận được phản hồi và timeout, bên gửi gửi toàn bộ gói tin từ ACK gần nhất nhận được là ACK1: gửi toàn bộ từ pkt2 -> pkt5.
=> Cumulative ACK (ACK tích lũy): ACK gần nhất nhận được sẽ cho ta biết những gói tin được đánh số trước đó đã nhận thành công (ACK 1 cho biết pkt1 và pkt0 đã nhận thành công)

b. Selective Repeat:



- Các điểm cần nhớ:

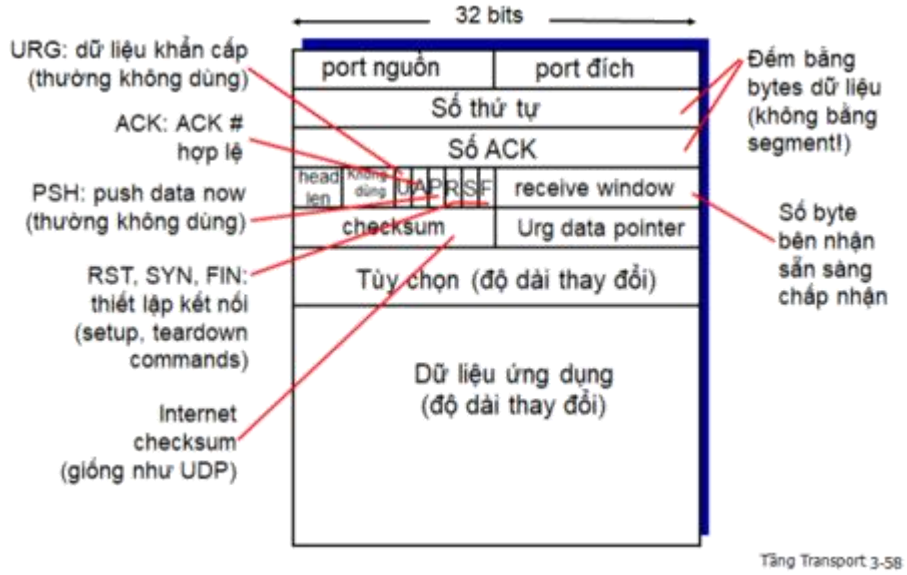
+ Bên gửi dùng bộ đệm (buffer): nhận và tích trữ các pkt sai thứ tự, gửi ACK tương ứng. Gói tin bị mất sẽ được bên gửi gửi lại sau thời gian timeout của nó.Sau khi nhận được các pkt đúng thứ tự, chuyển tất cả các pkt đó lên tầng Application. => Nhận thành công các gói tin,

+ Bên gửi liên tục gửi các gói tiếp theo sau khi nhận ACK các gói cũ, các gói bị mất sẽ được gửi lại sau timeout tương ứng.

=> Không có ACK tích lũy, nhưng có buffer

V. Giao thức TCP:

a. Định dạng:



Định dạng segment TCP

- Kích thước header TCP: 20 bytes
- Cổng, port, là một con số 16 bit, có giá trị từ 0 đến 65535. Trong đó, cổng có giá trị từ 0 đến 1023, được gọi là cổng phổ biến1 (well-known port number), bị giới hạn sử dụng; nghĩa là các cổng này được dành riêng cho các giao thức ứng dụng phổ biến như HTTP (cổng 80), FTP (cổng 21), và DNS (cổng 53). Khi chúng ta phát triển một ứng dụng mới, chúng ta phải gán cho ứng dụng một cổng.
- Seq (Sequence Number): Số thứ tự byte đầu tiên của trường dữ liệu trong luồng byte gởi. Được khởi tạo ngẫu nhiên (không bằng 0).

*** ISN:** Initial Sequence Number (ISN): Seq được khởi tạo ban đầu => Được tạo bằng thuật toán của hệ điều hành

Ví dụ: giả sử rằng một tiến trình P1 trong máy A muốn gởi tập tin có kích thước 500000 byte đến tiến trình P2 trong máy B thông qua một kết nối TCP. Và giả sử rằng giá trị MSS của kết nối TCP là 1000 byte và byte đầu tiên trong chuỗi dữ liệu được đánh số là 0. Như minh hoạ trong hình 3.30, TCP tạo 500 gói tin cho chuỗi dữ liệu của tập tin. Trường số thứ tự của gói tin TCP đầu tiên được gán bằng 0, gói tin thứ hai là 1000, thứ ba là 2000 và tương tự như vậy.

- ACK (Acknowledgement Number): Số thứ tự của byte tiếp theo mà bên gởi gói tin muốn nhận từ bên nhận gói tin

Có **6 cờ**:

Cờ	Có ý nghĩa khi được bật (bit cờ bằng 1)
URG (thường bằng 0)	Cờ khẩn: Gói tin cần truyền khẩn và giá trị trong urg data pointer có ý nghĩa
ACK	Cờ báo nhận: Gói tin chứa thông tin báo nhận và ACK num có ý nghĩa

PSH	Cờ đẩy: Dữ liệu trong gói tin này cần chuyển lên tầng ứng dụng ngay lập tức
RST	Cờ thiết lập lại Bên gởi gặp sự cố và muốn thiết lập lại kết nối
SYN	Cờ đồng bộ Gói tin này yêu cầu đồng bộ số thứ tự và thiết lập kết nối. Trường Số thứ tự chứa số thứ tự khởi tạo của bên gởi
FIN	Cờ kết thúc Gói tin yêu cầu đóng kết nối

* Khi bit cờ (trường cờ): có giá trị bằng 1 mới kích hoạt tính năng của cờ

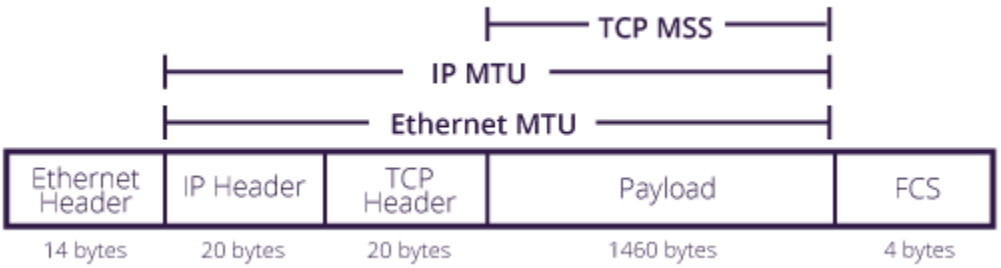
VD: bit SYN = 1: gói tin yêu cầu kết nối

- Receive window: Số byte trống trong bộ đệm nhận dữ liệu của bên receiver. Được sử dụng trong kiểm soát luồng

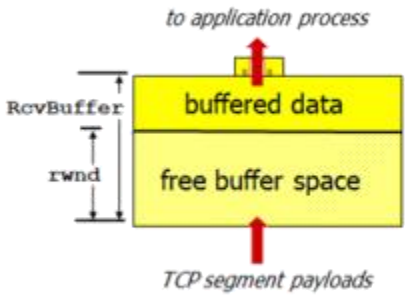
* TCP là một giao thức hai chiều (full-duplex), nghĩa là máy A có thể nhận dữ liệu từ máy B trong khi nó đang gởi dữ liệu cho máy B.

* MSS: Maximum Segment Size

MTU: Maximum Transmission Unit



b. TCP kiểm soát luồng (flow control):

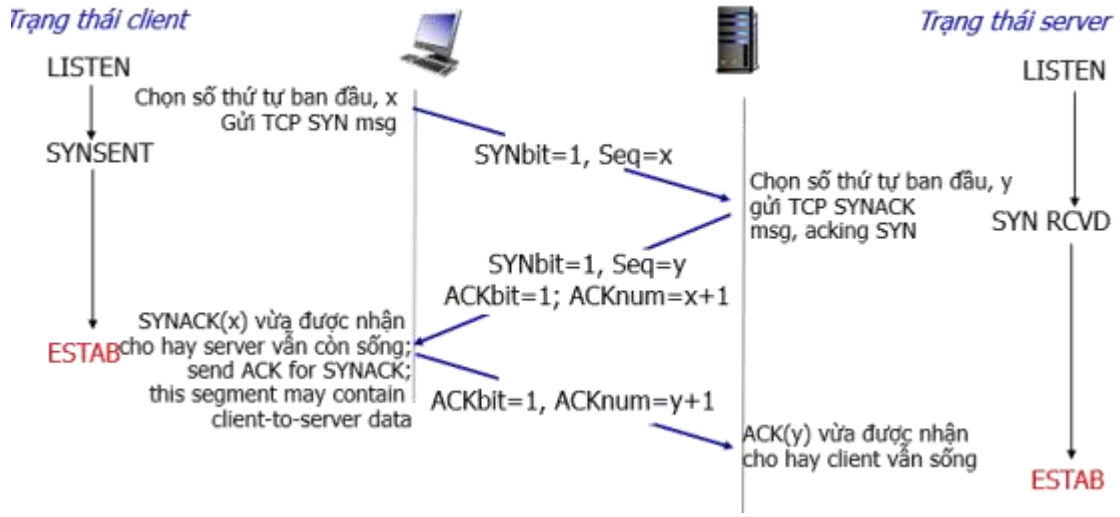


Bộ đệm phía bên nhận

- Trường Receive window (rwnd): Cho biết không gian buffer bên nhận còn lại là bao nhiêu. Bên gửi sẽ giới hạn lượng dữ liệu gửi đi, đảm bảo bên nhận không bị tràn bộ đệm.

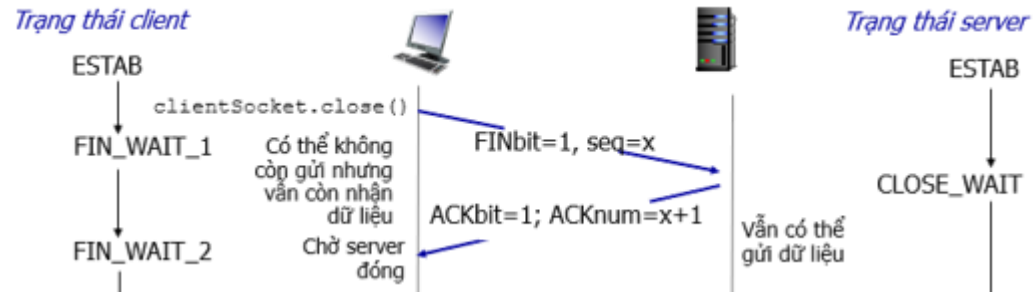
c. Kiểm soát kết nối:

- Thiết lập kết nối: three – way handshake

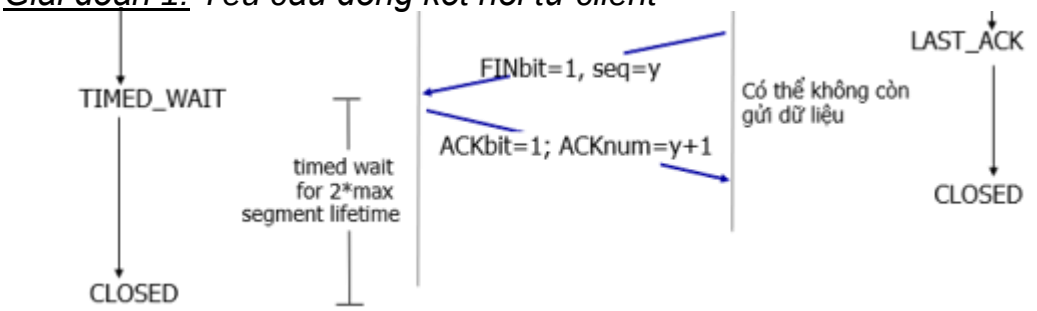


* Cờ SYN của segment đầu tiên sẽ bằng 1 => Báo hiệu yêu cầu kết nối, còn cờ ACK bằng 0 do là gói tin đầu tiên, không yêu cầu dữ liệu nhận lại.

- Đóng kết nối: four – way handshake



<u>Giai đoạn 1:</u> Yêu cầu đóng kết nối từ client



<u>Giai đoạn 2:</u> Yêu cầu đóng kết nối từ server

* Bên nào yêu cầu đóng kết nối trước cũng được

* Giai đoạn 2 client đợi bằng 2 rtt để chắc server đã nhận được ACK từ client

d. Điều khiển tắc nghẽn TCP (congestion control TCP):

+ Bên gửi tăng tốc độ truyền (kích thước window), thăm dò băng thông có thể sử dụng, cho đến khi mất mát gói xảy ra

+ Congestion window (CWND): quyết định số lượng byte có thể được gửi đi

- Giải thích thuật toán TCP Reno qua các giai đoạn:

1. Slow Start

- Khi kết nối bắt đầu, cwnd = 1 MSS. Tăng kích thước cwnd theo cấp số nhân cho đến khi sự kiện mất gói đầu tiên xảy ra. Tốc độ ban đầu chậm, nhưng nó sẽ tăng lên theo cấp số nhân

- Điều kiện: cwnd < ssthresh (slow start threshold) hoặc timeout (trường hợp timeout bắt đầu slow start lại)

- Các giá trị:

+ cwndnext = cwndprev * 2

+ ssthresh = ½ cwnd lần tắc nghẽn liền trước

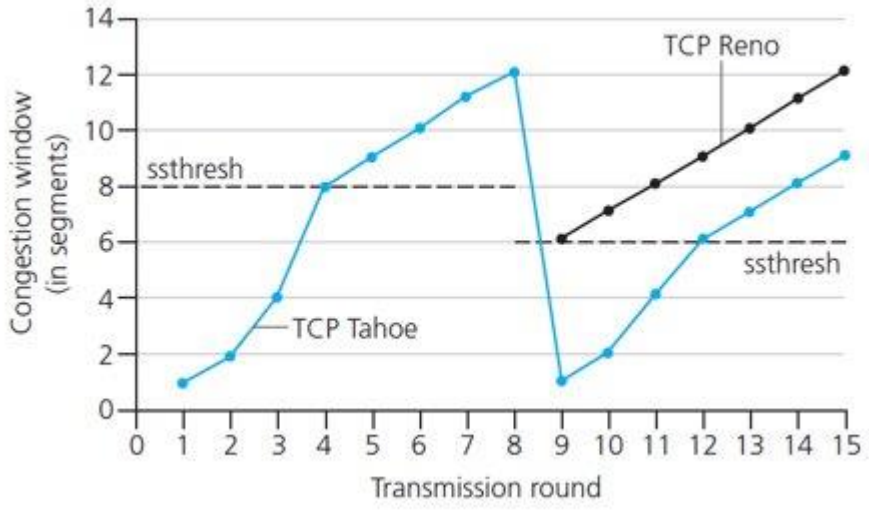
2. Congestion avoidance (CA)

- Điều kiện: cwnd >= ssthresh

- Các giá trị:

+ cwndnext = cwndprev + 1 MSS (đơn vị dữ liệu gửi trong thời gian)

+ ssthresh giữ nguyên



Chuyển từ Slow Start sang CA

* TCP Reno giống TCP Tahoe ở 2 giai đoạn này, giai đoạn 3 chỉ có TCP Reno mới có

* Bất cứ khi nào gặp timeout sẽ bắt đầu Slow Start lại ở cả 2 thuật toán

* TCP Tahoe gặp 3 ACK trùng hoặc timeout đều Slow Start lại do không có giai đoạn Fast Recovery

3. Fast Recovery

- Cách nhận biết: 3 ACK trùng

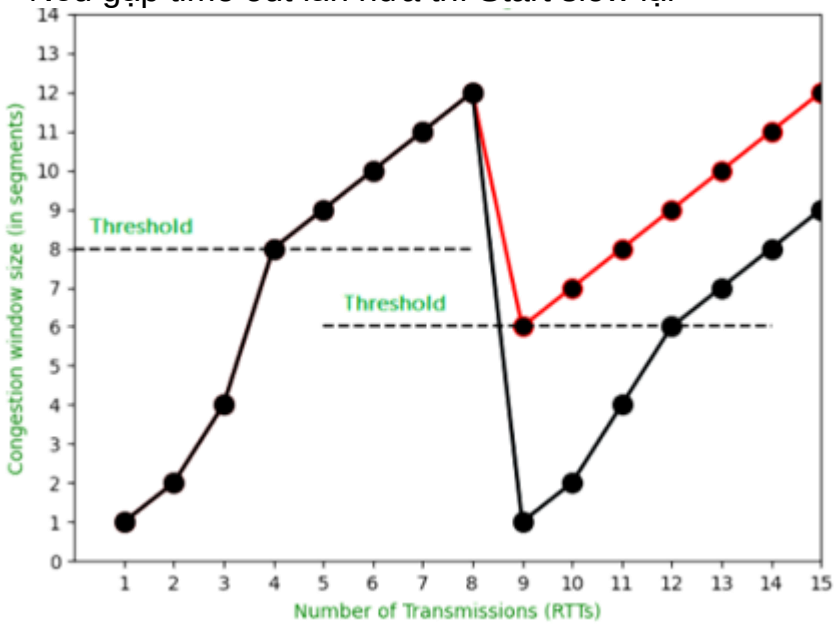
=> Fast – Retransmit (Gửi lại gói bị mất) rồi vào Fast - Recovery

- Tham số:

+ cwnd = ½ cwndprev và cwndnext = cwndprev + 1 MSS

+ ssthresh = new cwnd

=> Nếu gặp time out lần nữa thì Start slow lại



Phân biệt TCP Reno (đoạn đỏ) và TCP Tahoe (đoạn đen dưới đoạn đỏ tương ứng) khi gặp 3 ACK trùng