

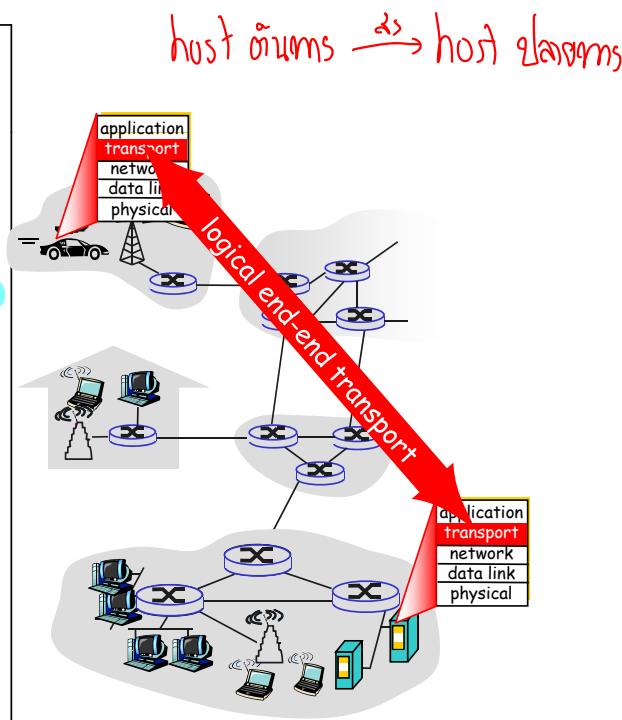
1

ក្នុងមេដាច់ នឹងបានចាប់ផ្តើម \rightarrow ពាក្យល់ទៅ (ពី) segment ជាសម្រាប់



Transport Services and Protocols

- Provide *Logical Communication* between application **processes** running on different hosts
- Transport protocols run in end systems
 - Sender side: breaks application messages into **Segments**, passes to network layer
 - Receiver side: reassembles segments into **Messages**, passes to application layer
- more than one transport protocol available to applications
 - Internet: **TCP and UDP**

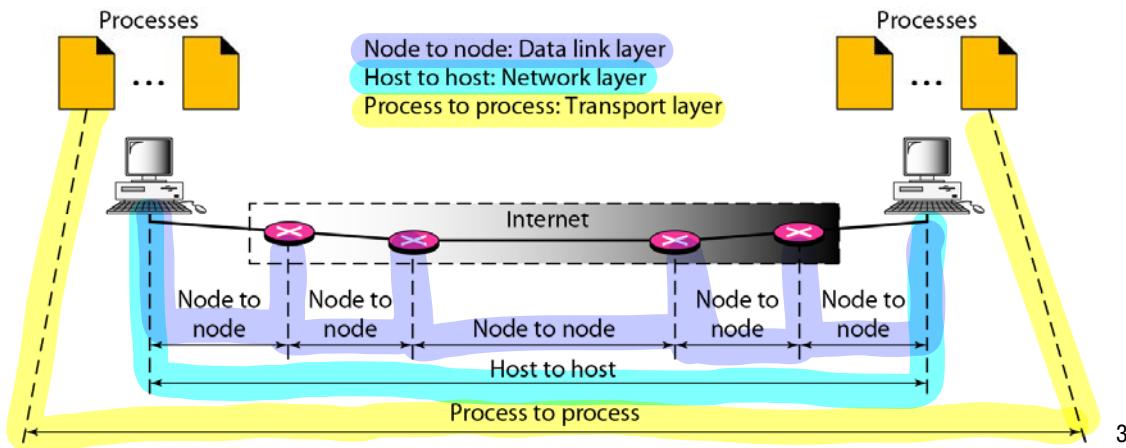


2

Type of Data Deliveries

from mac address

- **Data link layer:** logical communication between **nodes**
- **Network layer:** logical communication between **hosts**
- **Transport layer:** logical communication between **processes**
 - relies on, enhances, network layer services



From a process to another process
via a port

Process-to-Process Delivery

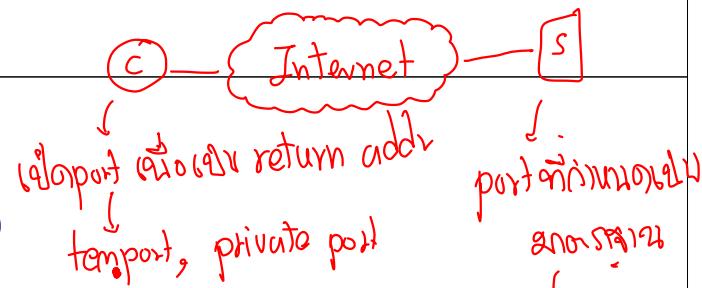
- OS today support both **multiuser** and **multiprogramming** environments
 - Computer can run several programs at the **same time**
- Client-Server Paradigm
- Process on local host (client) needs services from process on remote host (server)
- Both **processes** (client and server) have the same name, ex. ftp, DNS
- For communication, we must define
 - **Local Host**
 - **Local Process**
 - **Remote Host**
 - **Remote Process**

Addressing

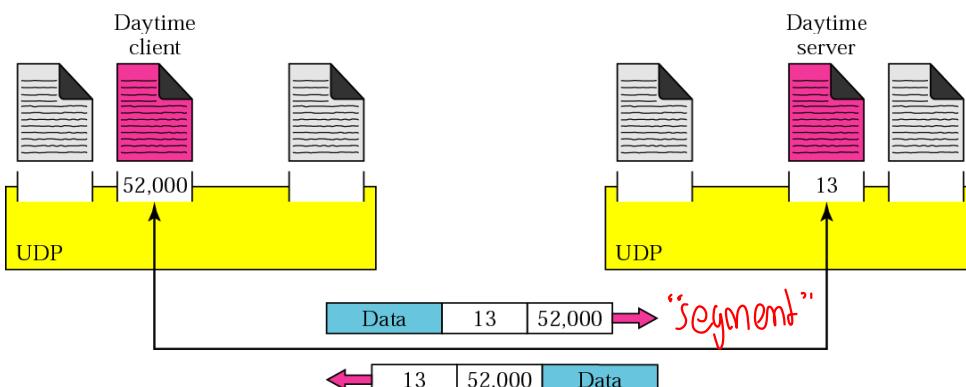
- We need **address**, when we **need to deliver** something **to one specific destination** among many
 - Data link layer : MAC address (**48bit**)
 - Network Layer : IP address (**32bit, 128bit**)
 - Transport Layer : Port Number (**16bit**)
- Internet model, port number are **16-bit integer** → **0 to 65,535** between **0 and 65,535**
- **Client process** defines itself with port number, chosen randomly called **ephemeral (temporary) port number**
- **Server process** must also define itself with port number, which **cannot** chosen randomly

5

Addressing (cont.)



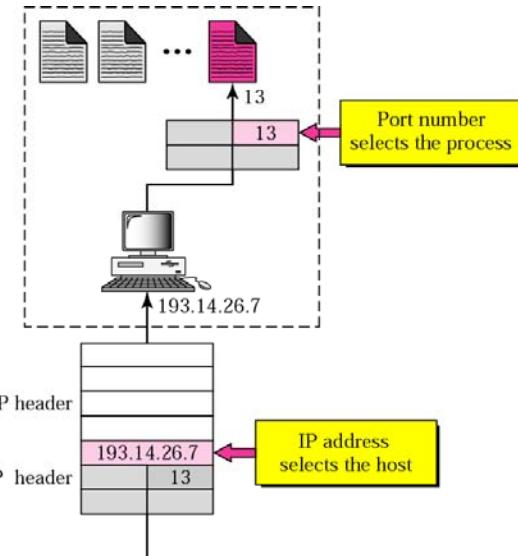
- For **Server**, Internet has decided to use universal port number called **well-known port numbers**
- Every client process knows well-known port number



6

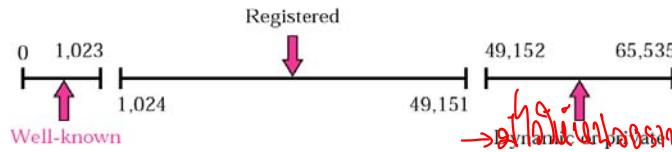
Addressing (cont.)

- It should be clear that *IP addresses* and *Port Numbers* play different roles in selecting final destination of data
- *Destination IP address* defines **host** among different hosts in the world
- *Port number* defines one of **process** on this particular host



7

IANA Ranges (cont.)



- **Well-known Ports**
 - ranging from **0 to 1023**
 - Are assigned and controlled by IANA
- **Registered Ports** → මිලදා විවෘත කළ ඇත්තා ඇතුළතා නොවා app සිංහල පෙරේ පෙන්වනු ලබයි
 - Ranging from **1,024 to 49,151**
 - Not assigned and controlled by IANA
 - They can only be registered with IANA to prevent duplication
- **Dynamic (or private) Ports** → මැසම් සිංහල පෙන්වනු ලබයි
 - Ranging from **49,152 to 65,535**
 - Neither controlled nor registered
 - They can be used by any process
 - These are **ephemeral ports or temporary ports**

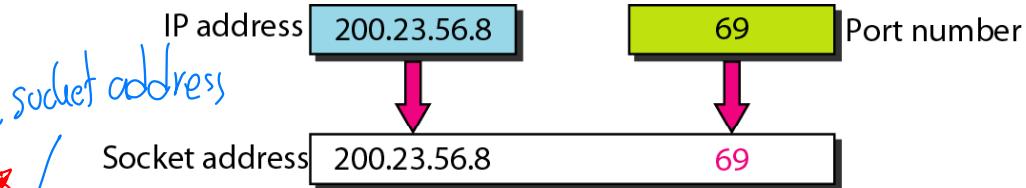


(<http://www.iana.com>)

8

Socket Address

- *Socket Address* is combination of IP Address and Port Number



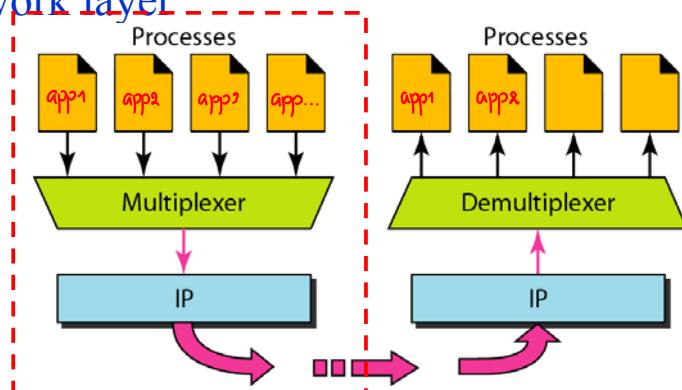
- Transport layer protocol needs a pair of socket addresses : client socket address and server socket address
- These four pieces of information are part of **IP header** and **Transport layer protocol header**
- IP header contains IP address; UDP or TCP header contains port number

9

Multiplexing and Demultiplexing

Multiplexing

- There may be several processes that need to send packet
- Protocol accepts messages from different processes, differentiated by their port number *լինգուիստական պահումներ*
- After adding header, transport layer passes packet to network layer

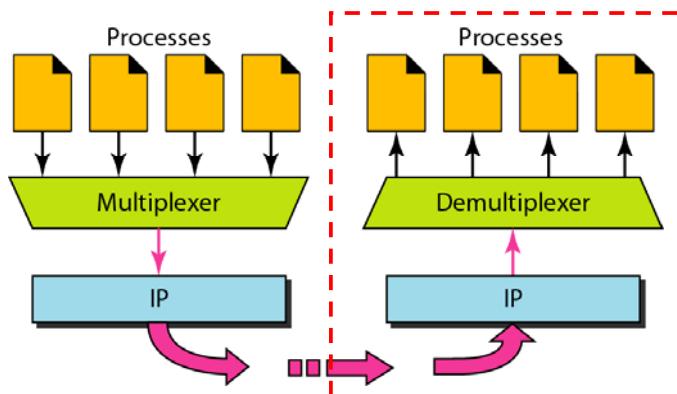


10

Multiplexing and Demultiplexing (cont.)

■ Demultiplexing

- Transport layer receives datagrams from network layer
蜀udagram ຂັ້ນເກີດໄຟລົງ
- After error checking and dropping of header, transport layer delivers each message to appropriate process based on port number

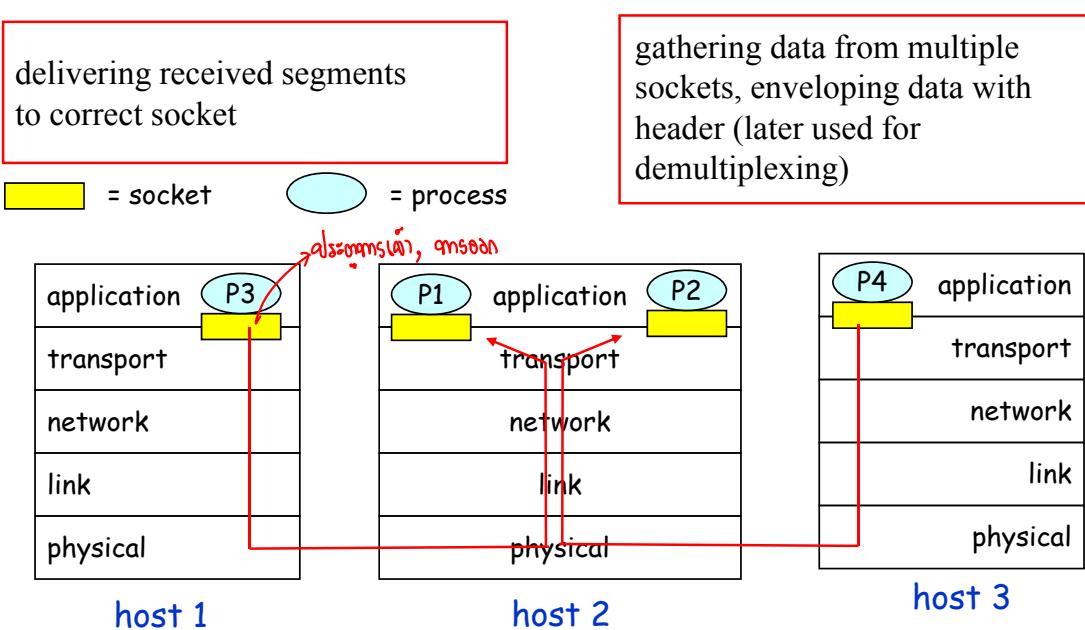


11

Multiplexing/Demultiplexing (cont.)

Demultiplexing at receiving host:

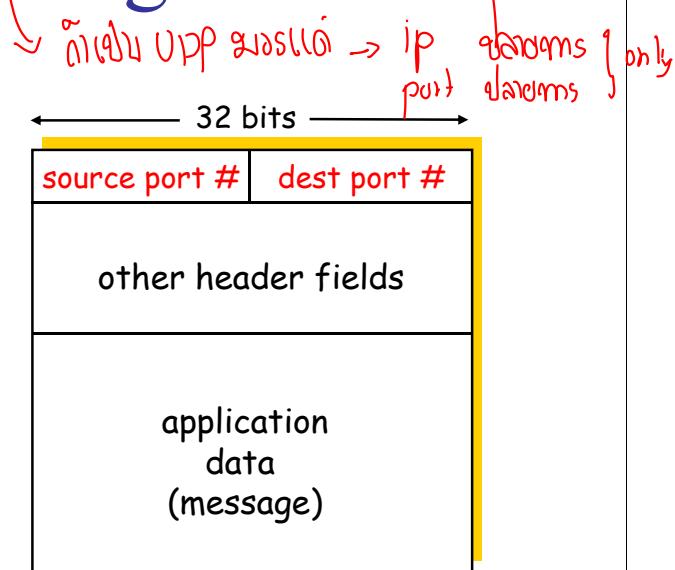
Multiplexing at sending host:



12

How demultiplexing works

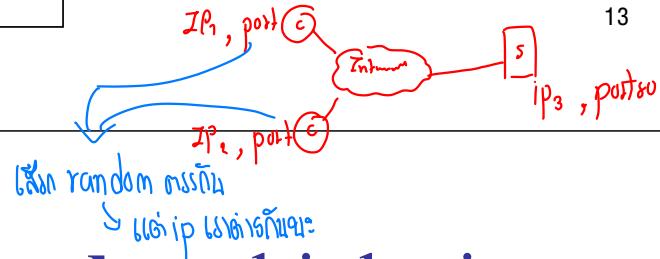
- Host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries 1 transport-layer segment
 - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format



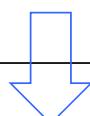
13



Connectionless demultiplexing

- Create sockets with port numbers:
- ```
 DatagramSocket mySocket1 =
 new DatagramSocket(49152);

 DatagramSocket mySocket2 =
 new DatagramSocket(49153);
```
- UDP socket identified by two-tuple:



- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

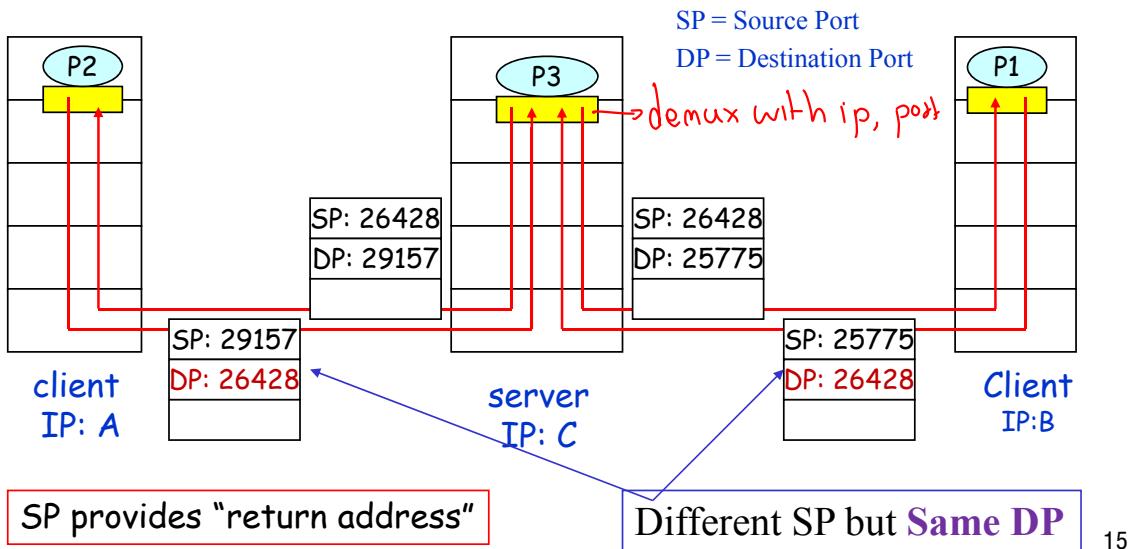
(destination IP address, destination Port number)

UPP

14

# Connectionless demux (cont)

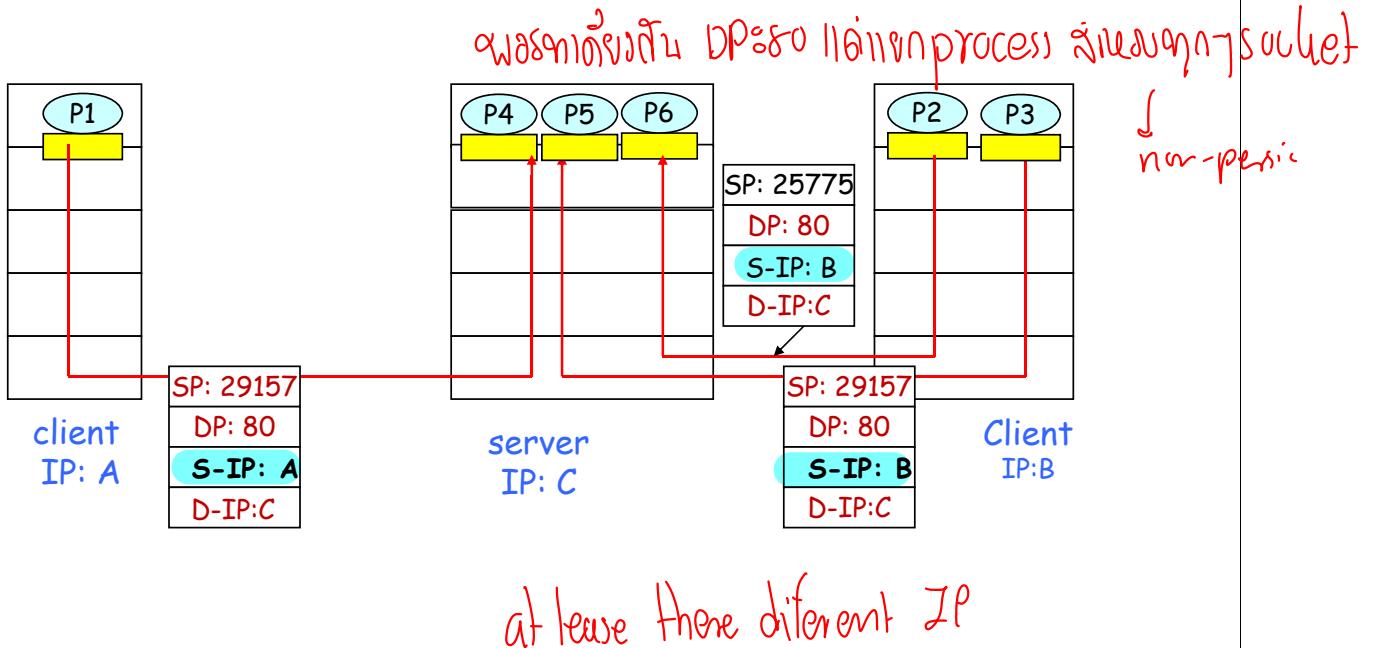
```
DatagramSocket serverSocket = new DatagramSocket(26428);
```



## Connection-oriented demux

- TCP socket identified by 4-tuple:
  - Source IP address
  - Source Port number
  - Destination IP address
  - Destination Port number
- Receiver host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

# Connection-oriented demux (cont)



## Connectionless versus Connection-Oriented Service

### ■ Connectionless Service

- Packet are sent from one party to another with **no need for connection establishment or connection release** → ໃຊ້ຂາຍກ່າວໄຫວ່າລະດົບ, ໃຊ້ຂາຍກ່າວສ່ວນການສ່ວນ
- Packet are **not numbered**
  - They may be **delayed or lost or may arrive out of sequence**
  - **No acknowledgment** → ໃຊ້ໄລຍະ
- **UDP** is connectionless
  - ↪ ຕິດຕາມຕາມຫົວໜ້າ + ອົກສະໜັກ

17  
protocol  
layer  
de-mux  
transport  
layer  
transport  
layer

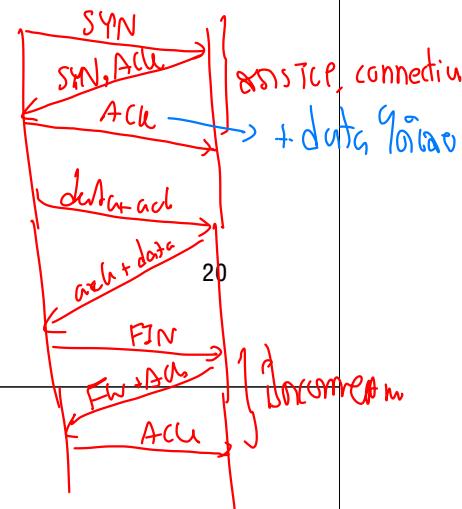
ພາກສັ່ນກ່າວໄຫວ່າລະດົບທີ່ TCP

# Connectionless versus Connection-Oriented Service

## ■ Connection-Oriented Service

- Connection is first established between sender and receiver
  - Data are transferred; at the end, connection is released
  - Acknowledgement is needed
  - TCP is connection-oriented protocol

ကောမ်းဝါယံတော်သုတေသနများမှာမူမျှလေ့လာမည့်  
ပို့ဆောင်ရေးနှင့်ပြုပဲလေယာဉ်မှုပါန်များ  
TCP အိမ်များမှာမူမျှလေ့လာမည့်



# **Reliable versus Unreliable**

- Transport layer service can be reliable or unreliable (ඖෂධ්‍යතාවයෙන්)
    - ↳ මුද්‍රණය
    - ↳ ත්‍රුප්පිය
  - Reliable transport layer protocol
    - implementing flow and error control at transport layer
      - ↳ වැට්ටාම් ලැබේ
    - Slower and more complex service
  - Unreliable transport layer protocol
    - No flow and error control
    - Faster service

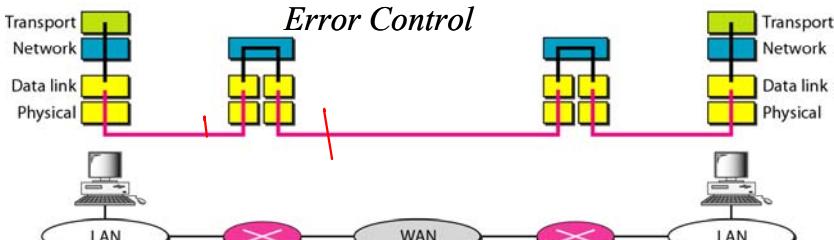
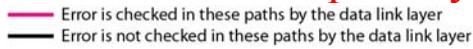
ጀመንት የሚከተሉ በnetwork  
↳ segmentation , VLANs  
↳ traffic                      ↳ somany

၆၁

## **Reliable versus Unreliable (cont.)**

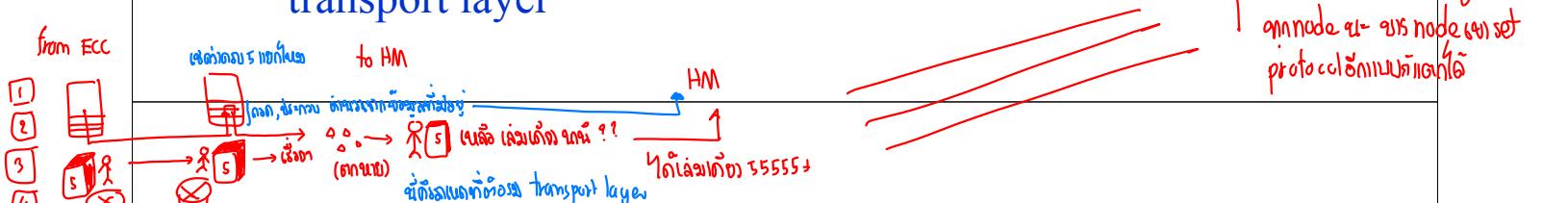
Question? → in stop-and-wait, go-back-N, selective → initial transport goes on

- If data link layer is reliable and has flow and error control, do we need this at transport layer, too?



## *Answer*

- Reliability at data link layer is between two nodes
  - We need reliability between two ends ①
  - Because network layer in Internet is unreliable (best effort delivery), we need to implement reliability at transport layer

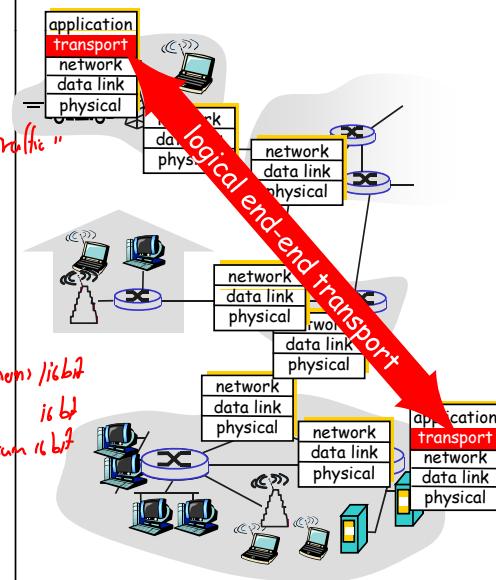


# Internet Transport-layer Protocols

- Reliable, in-order delivery :  
**(Transmission Control Protocol, TCP)**

- Connection setup
  - Flow control
  - Error control → “retransmission”  
→ “ վեճուկային լարացում”
  - Congestion control → “ վեճուկային լարացում”
  - **Unreliable, unordered delivery:**  
**(User Datagram Protocol, UDP)**

- **Unreliable, unordered delivery:** (User Datagram Protocol, UDP)
    - no-frills extension of “best-effort” IP
    - No connection setup
    - No flow control
    - No error control
    - No congestion control
  - services not available:
    - delay guarantees
    - bandwidth guarantees



## UDP: User Datagram Protocol [RFC 768]

- UDP does **not add anything** to service of Internet Protocol except to provide **process-to-process communication**
- **Unreliable transport Protocol** (best effort service), UDP segments may be:
  - lost
  - delivered out of order to application
- **connectionless:**
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

24

## UDP: User Datagram Protocol

### Why is there a UDP?

- No connection establishment (which can **add delay**)  
↳ *ନେଇମାର୍କିସିମ୍ବନ୍ଧ*
- Simple protocol using minimum of overhead: no connection state at sender, receiver  
↳ *କୁଣ୍ଡଳିତ ପରିପରାମରଣ*
- small segment header
- no congestion control: UDP can blast away as fast as desired

25

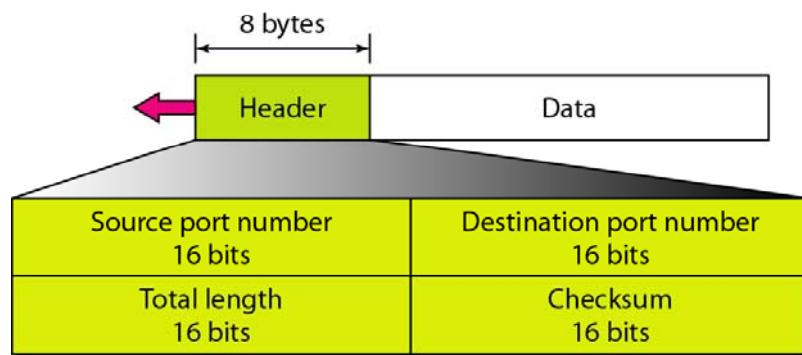
# Well-Known Ports for UDP

| Port | Protocol   | Description                                   |
|------|------------|-----------------------------------------------|
| 7    | Echo       | Echoes a received datagram back to the sender |
| 9    | Discard    | Discards any datagram that is received        |
| 11   | Users      | Active users                                  |
| 13   | Daytime    | Returns the date and the time                 |
| 17   | Quote      | Returns a quote of the day                    |
| 19   | Chargen    | Returns a string of characters                |
| 53   | Nameserver | Domain Name Service                           |
| 67   | BOOTPs     | Server port to download bootstrap information |
| 68   | BOOTPc     | Client port to download bootstrap information |
| 69   | TFTP       | Trivial File Transfer Protocol                |
| 111  | RPC        | Remote Procedure Call                         |
| 123  | NTP        | Network Time Protocol                         |
| 161  | SNMP       | Simple Network Management Protocol            |
| 162  | SNMP       | Simple Network Management Protocol (trap)     |

26

## UDP Segment Format

- UDP segments have **fixed-size header of 8 bytes**



- **Total length** defines total length of UDP segment, header plus data
- **Checksum** is used to detect error over the entire segment (header plus data)

27

# UDP checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

## Sender:

- treat segment contents as sequence of 16-bit integers
- **Checksum:** addition (1's complement sum) of segment contents
- Sender puts checksum value into UDP checksum field

## Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected.

29

# Internet Checksum Example

## ■ Note

- When adding numbers, a carry out from the most significant bit needs to be added to the result

## ■ Example: add two 16-bit integers

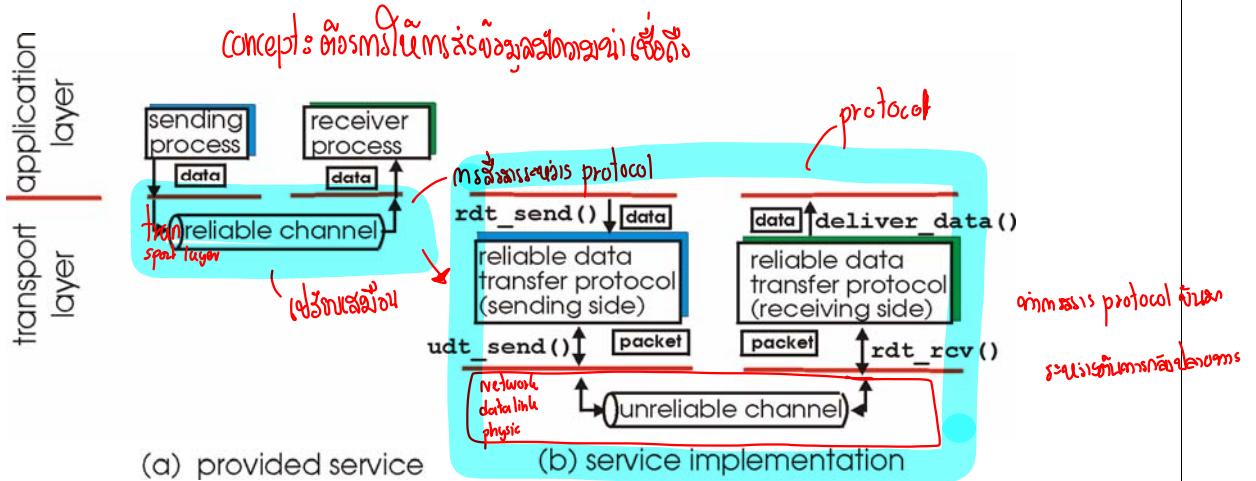
|            |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|            | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
|            | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| <hr/>      |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| wraparound | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| sum        | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| checksum   | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

30

ឧប្បជ្ជកម្មសារព័ត៌មាន

# Principles of Reliable Data Transfer

- important in application, transport, link layers

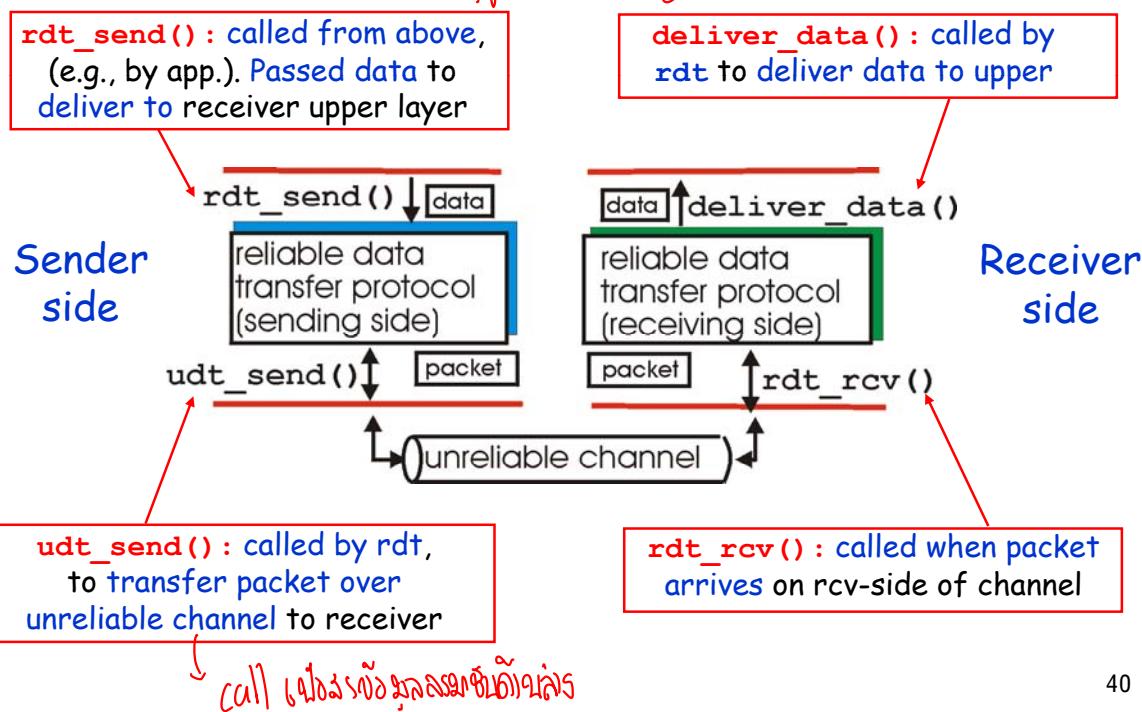


- characteristics of unreliable channel will determine complexity of **Reliable Data Transfer protocol (rdt)**

39

## Reliable Data Transfer: getting started

និយាយនូវការផ្តល់ព័ត៌មាន

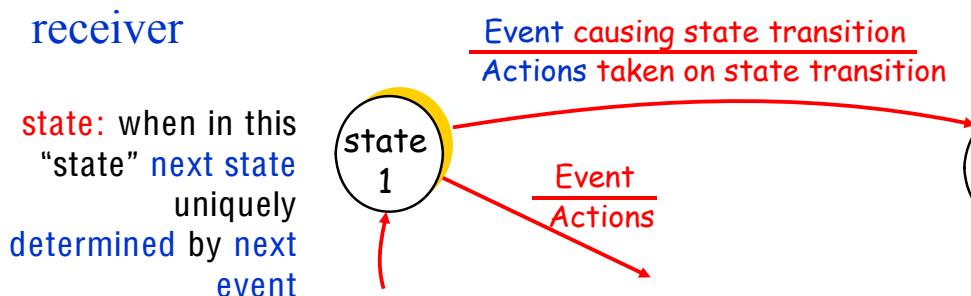


40

# Reliable Data Transfer: getting started

We'll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use Finite State Machines (FSM) to specify sender, receiver



41

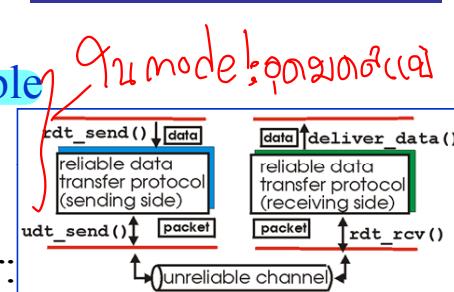
## Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable

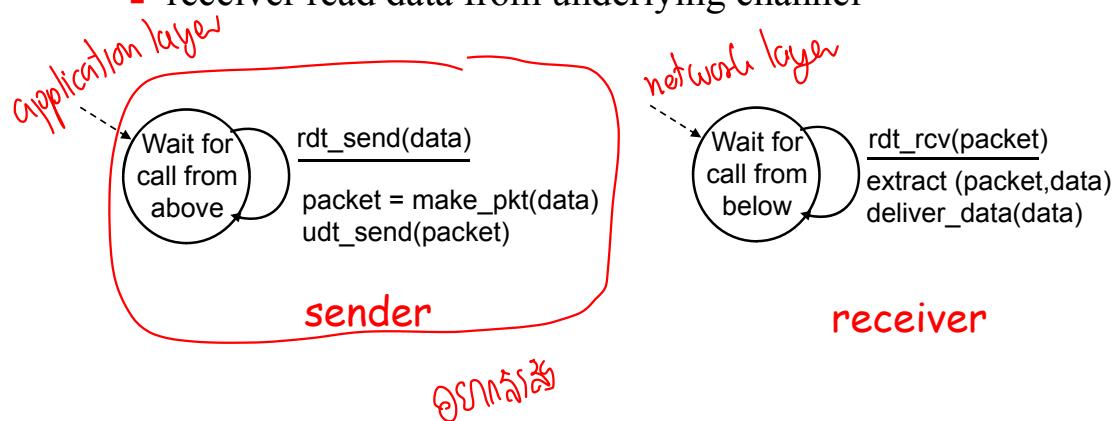
- no bit errors
  - no loss of packets

- separate FSMs for sender, receiver:

- sender sends data into underlying channel
  - receiver read data from underlying channel



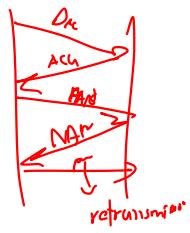
call backs



42

Reliable data communication

## Rdt2.0: channel with bit errors

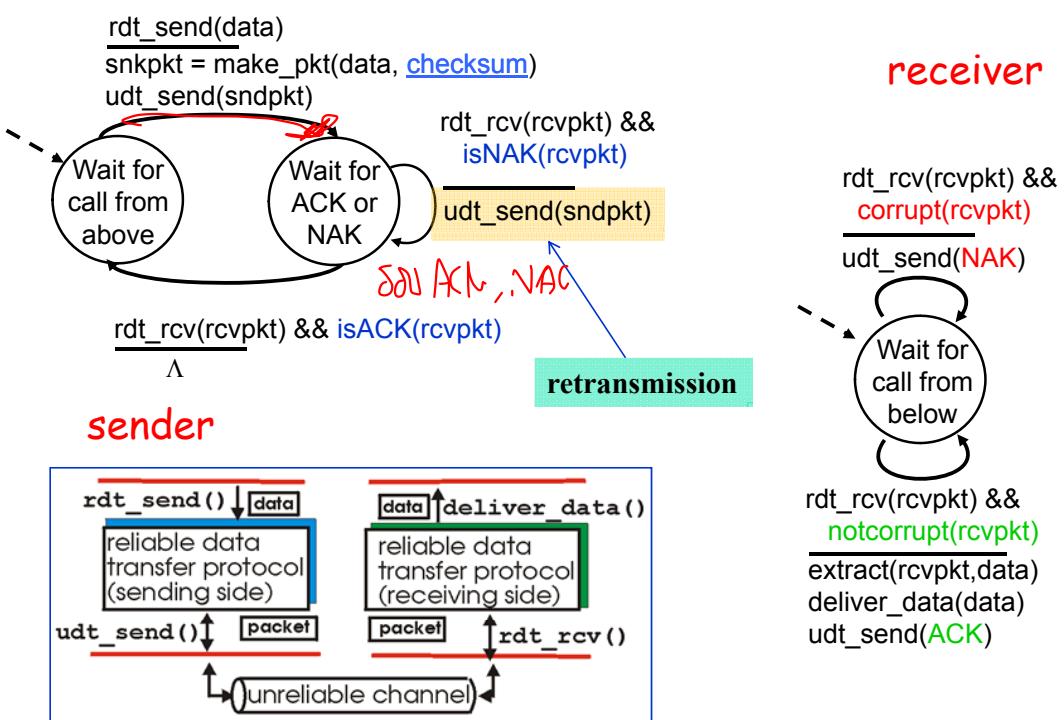


on packets  
channel

- Underlying channel may flip bits in packet
  - Checksum to detect bit errors
- Question: how to recover from errors:
  - **ACKnowledgements (ACKs)**: receiver explicitly tells sender that packet received OK
  - **Negative ACKnowledgements (NAKs)**: receiver explicitly tells sender that packet has errors
  - sender **retransmits** packet on receipt of NAK
- new mechanisms in **rdt2.0** (beyond **rdt1.0**):
  - Error Detection → ~~missed bit~~
  - Receiver Feedback: → ~~high error module, receiver module~~
    - control messages (ACK, NAK) receiver->sender

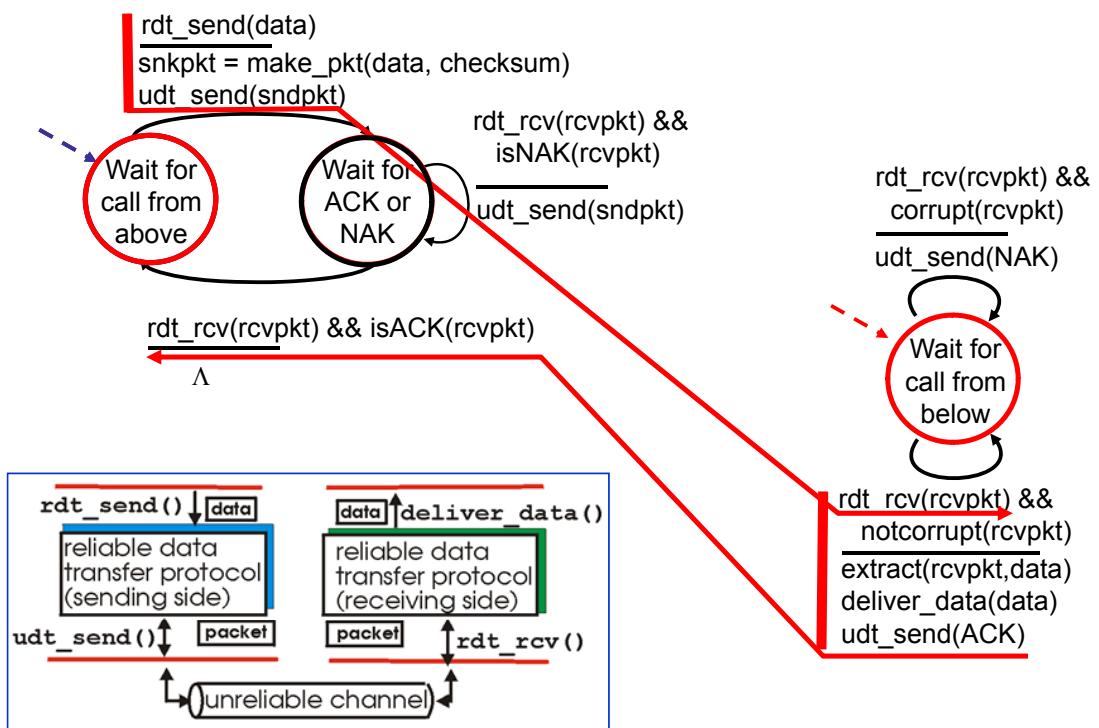
43

## rdt2.0: FSM specification



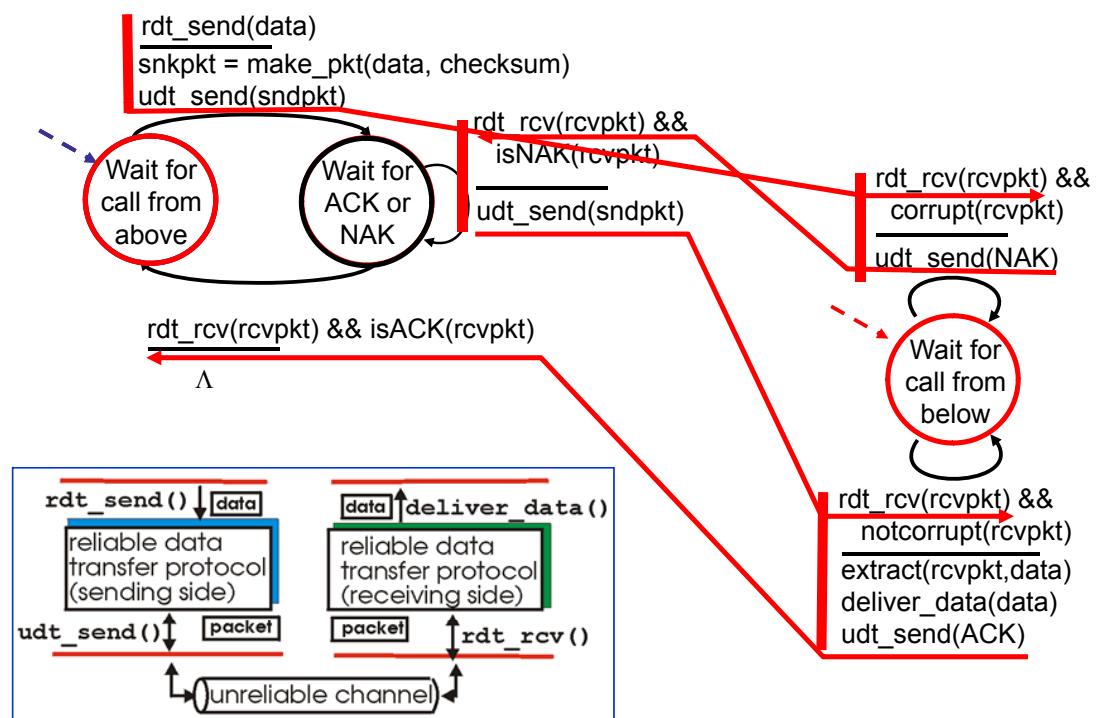
44

# rdt2.0: operation with no errors



45

# rdt2.0: error scenario



46

# rdt2.0 has a fatal flaw!

What happens if  
ACK/NAK  
corrupted?

- Sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

Handling duplicates:

- sender retransmits current pkt if ACK/NAK garbled
- sender adds *Sequence Number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait

Sender sends one packet, then waits for receiver response

Opportunistic drop from initial condition  
↓  
↓ frame seq  
↓ flow control

47

## rdt3.0: channels with errors and loss

### New assumption:

underlying channel can also lose packets (data or ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help



But not enough

### Approach:

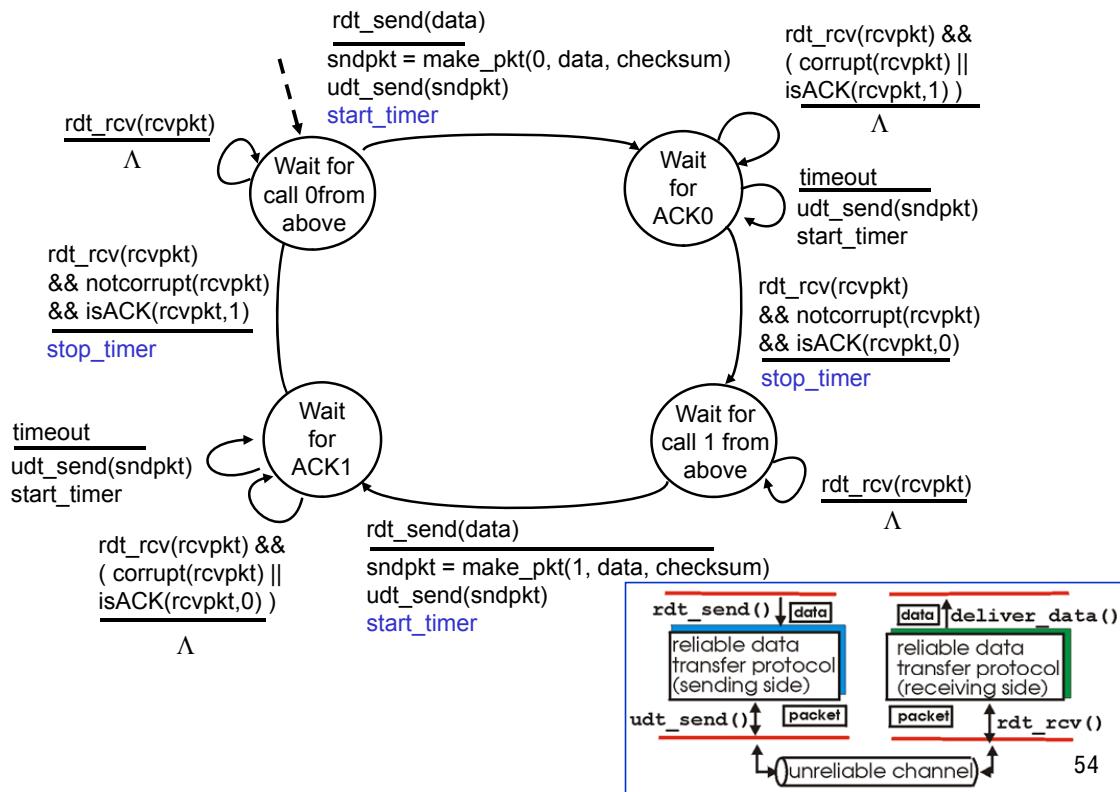
sender waits “reasonable” amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

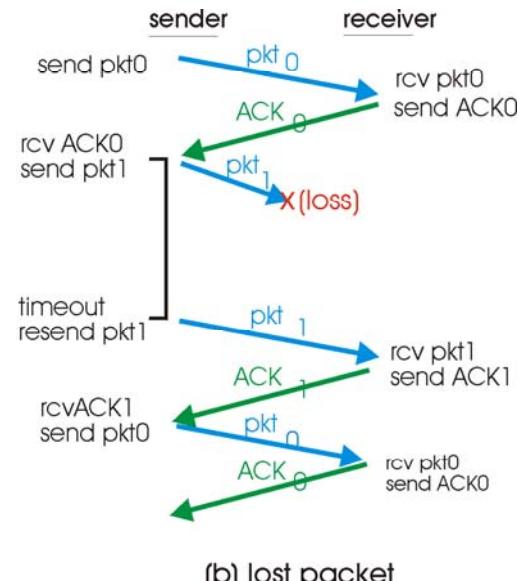
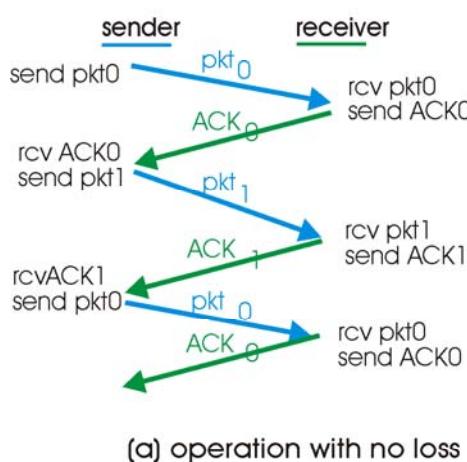
→ if timeout (เวลาหมด) → time out

53

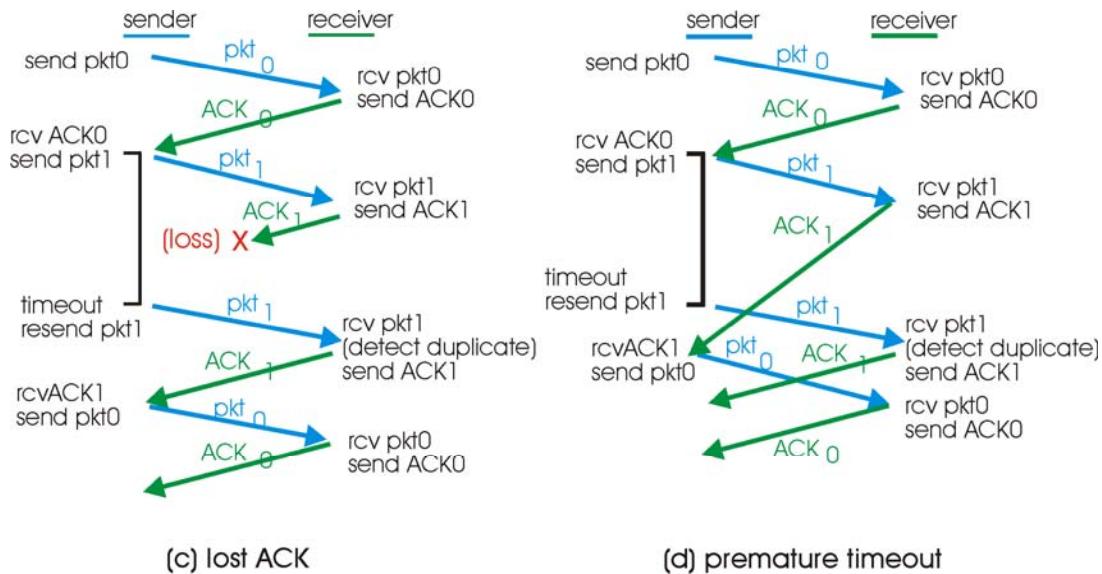
# rdt3.0 sender



# rdt3.0 in action



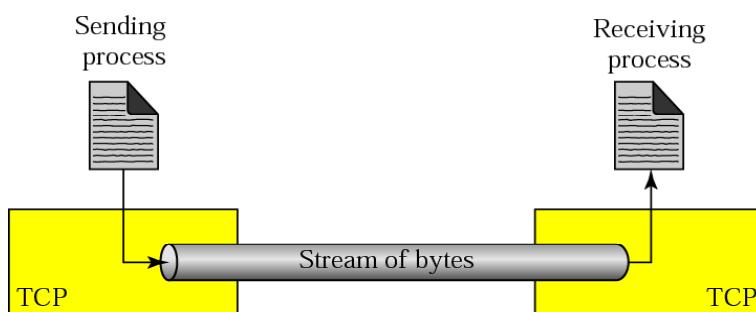
# rdt3.0 in action



56

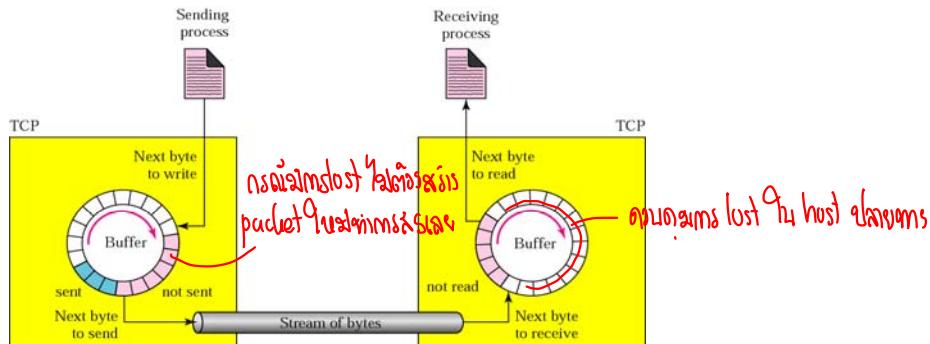
## TCP Services

- Process-to-Process Communication
- Stream Delivery Service
  - TCP allows sending process to delivery data as a stream of bytes and allows receiving process to obtain data as a stream of bytes
  - TCP creates an environment in which two process seem to be connected that carries data across Internet



58

## TCP Services : Stream Delivery Service (continued)

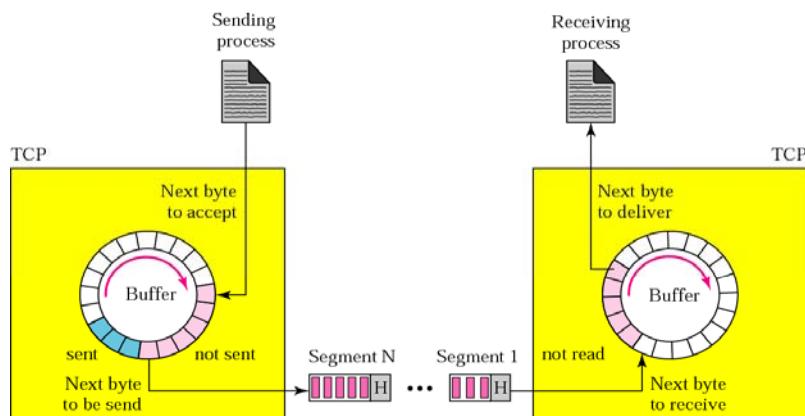


### Sending and Receiving Buffers

- Because sending and receiving processes may not write or read data at the same speed, TCP needs buffers for storage
- There are two buffers, sending buffer and receiving buffer
- Buffers are also necessary for *flow and error control* mechanisms used by TCP

59

## TCP Services : Stream Delivery Service (continued)



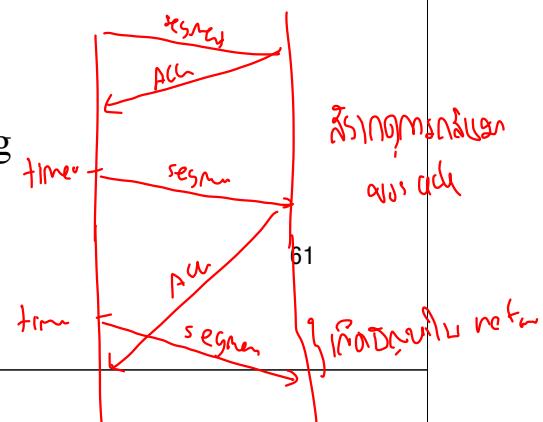
### Segment

- TCP groups a number of bytes together into a packet called **Segment**
- TCP adds **header** to each segment (for control purposes) and **delivers** segment to **IP layer** for transmission
- Segments are encapsulated in **IP datagram** and transmitted
- The entire operation is transparent to receiving process

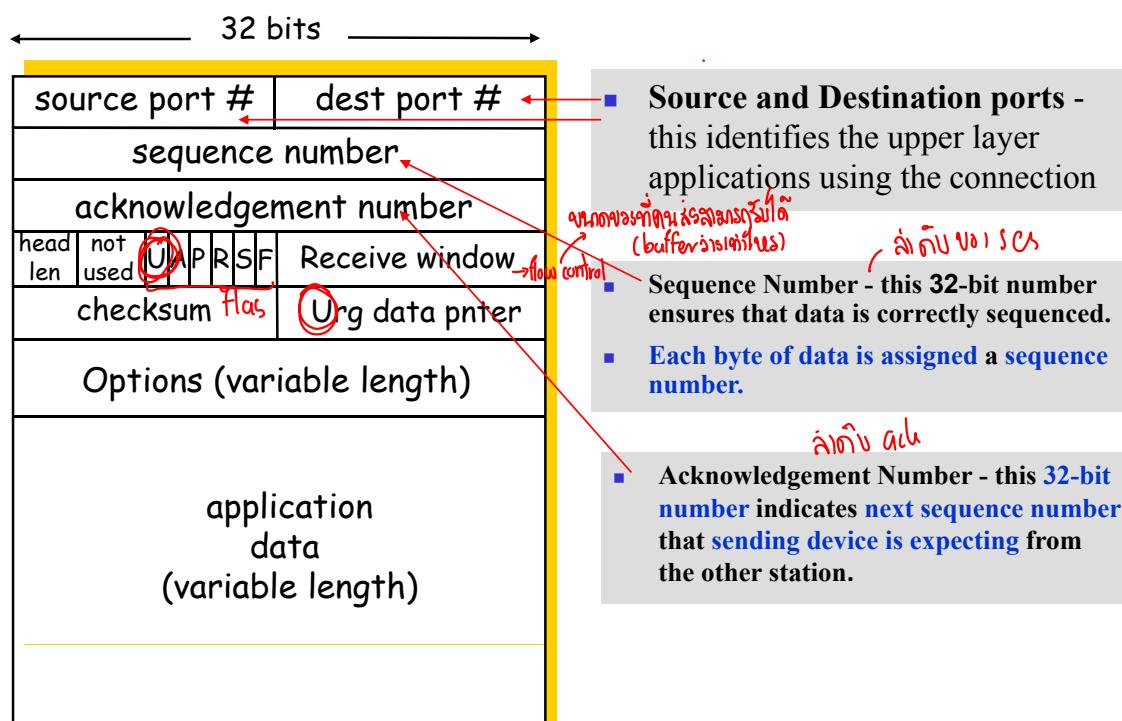
60

# TCP Services (continued)

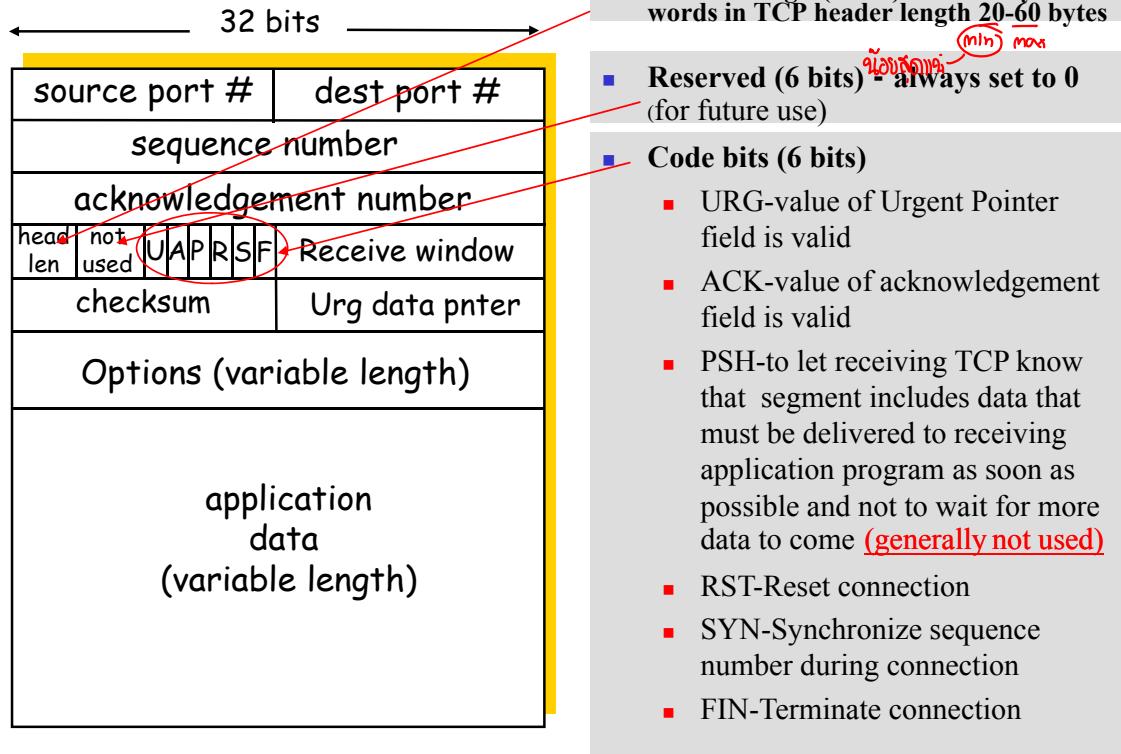
- **TCP offers full-duplex service**
  - Data can flow in both directions at the same time
  - Each TCP has sending and receiving buffer and segments move in both direction
- **Connection-Oriented Service**
  - Before sending and receiving data, following processes occur
    - Two TCP establish a connection between them
    - Data are exchanged in both directions
    - Connection is terminated
- **Reliable Service**
  - TCP is reliable transport protocol by using acknowledgement mechanism



## TCP Segment Structure

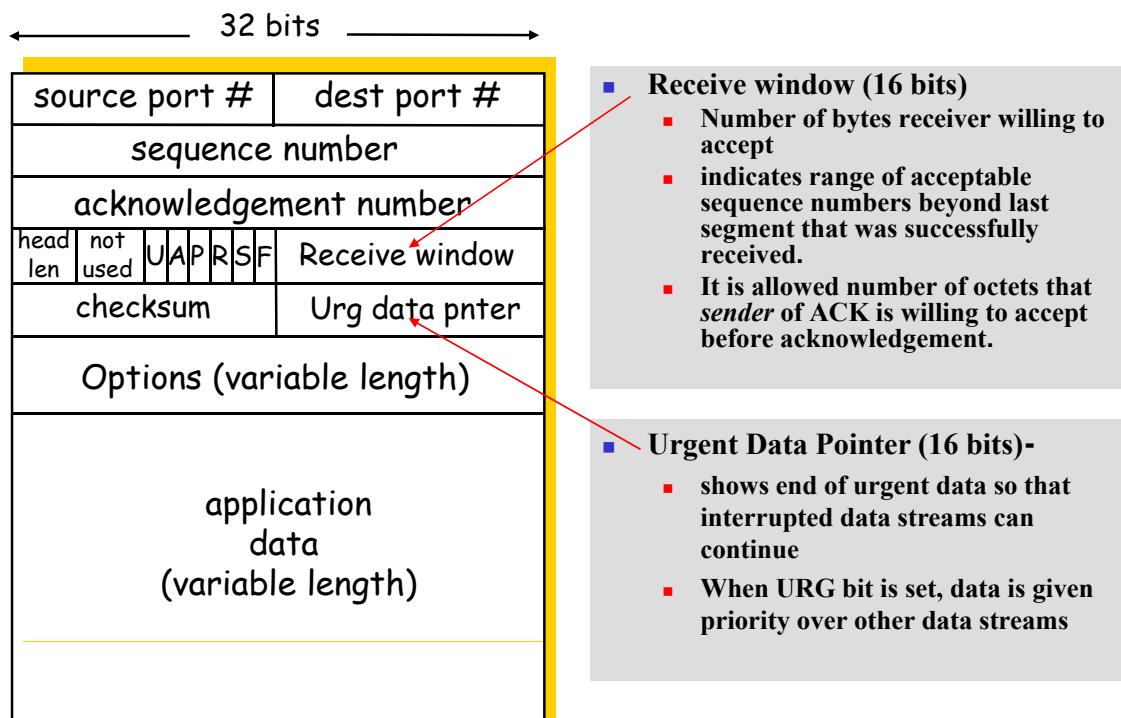


# TCP Segment Structure



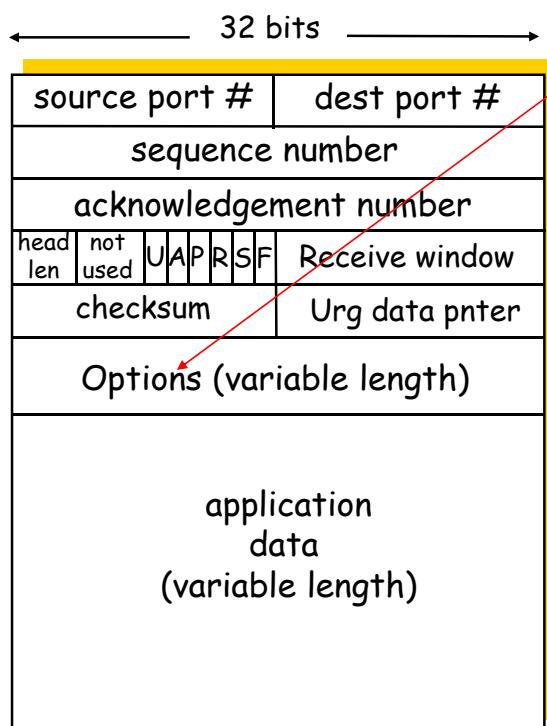
64

# TCP Segment Structure



65

# TCP Segment Structure



⑩ **Option (0-32 bits)**- mainly only TCP Maximum Segment Size (MSS) sometimes called Maximum Window Size or Send Maximum Segment Size (SMSS).

⑪ Segment is series of data bytes within TCP header.

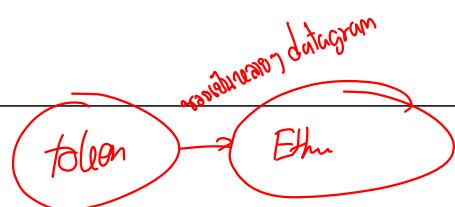
IP datagram



**Maximum Transfer Unit : Max. length of data that can be encapsulated in a frame**

- Token Ring (16 Mbps) – 17,914 bytes
- Token Ring (4 Mbps) - 4,464 bytes
- Ethernet - 1,500 bytes
- Point-to-Point Protocol (PPP)–296 bytes

66



## TCP Features

- *To provide services mentioned in previous section, TCP has several features as follows :*

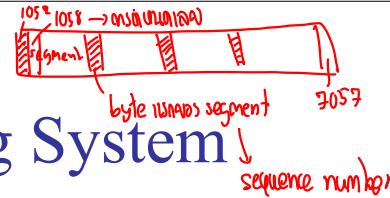
■ **Numbering System**

■ **Flow Control** → ms lost നിയന്ത്രണം

■ **Error Control**

■ **Congestion Control** → ms lost നിയന്ത്രണം

67



## TCP Features : Numbering System

- Although TCP keeps track of segments being transmitted or received
- **No field for segment number value** in segment header
- Instead, there are two fields called
  - Sequence Number } refer to byte number
  - Acknowledgement Number } and not segment number
- Bytes of data being transferred in each connection are numbered by TCP
- Numbering starts with a randomly generated number. ( $0 - 2^{32}-1$ )
  - Ex. Random generate No. = 1,057, for 6,000 bytes data
  - Range of data is start from 1,057 to 7,056

68

## TCP Features : Numbering System

- What is Sequence Number
  - After bytes have been numbered, TCP assigns a sequence number to each segment that is being sent
- Sequence number for each segment is number of the first byte carried in that segment

69

## TCP Features : Numbering System

- Example :

- Suppose TCP connection is transferring file of 5,000 bytes.
- The first byte is numbered 10001.
- What are sequence numbers for each segment if data is sent in five segments, each carrying 1000 bytes?

|                       |                       |                       |                       |                       |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| <u>10,001</u> -11,000 | <u>11,001</u> -12,000 | <u>12,001</u> -13,000 | <u>13,001</u> -14,000 | <u>14,001</u> -15,000 |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|

|---1,000 bytes---

qis̥mung first byte + 1,000

**Segment 1** → Sequence Number: 10,001 (range: 10,001 to 11,000)

**Segment 2** → Sequence Number: 11,001 (range: 11,001 to 12,000)

**Segment 3** → Sequence Number: 12,001 (range: 12,001 to 13,000)

**Segment 4** → Sequence Number: 13,001 (range: 13,001 to 14,000)

**Segment 5** → Sequence Number: 14,001 (range: 14,001 to 15,000)

**Value of sequence number field of segment defines number of the first data byte contained in that segment**

70

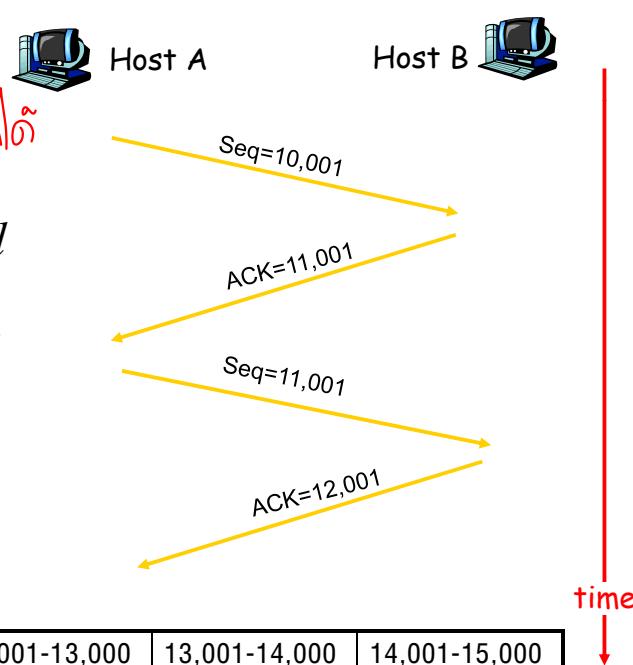
## TCP Features : Numbering System

- What is

**Acknowledgment Number?** ↴ seq qis̥mung

- Value of acknowledgment field in segment defines number of next byte a party expects to receive

- Acknowledgment number is cumulative



|               |               |               |               |               |
|---------------|---------------|---------------|---------------|---------------|
| 10,001-11,000 | 11,001-12,000 | 12,001-13,000 | 13,001-14,000 | 14,001-15,000 |
|---------------|---------------|---------------|---------------|---------------|

|---1,000 bytes---

71

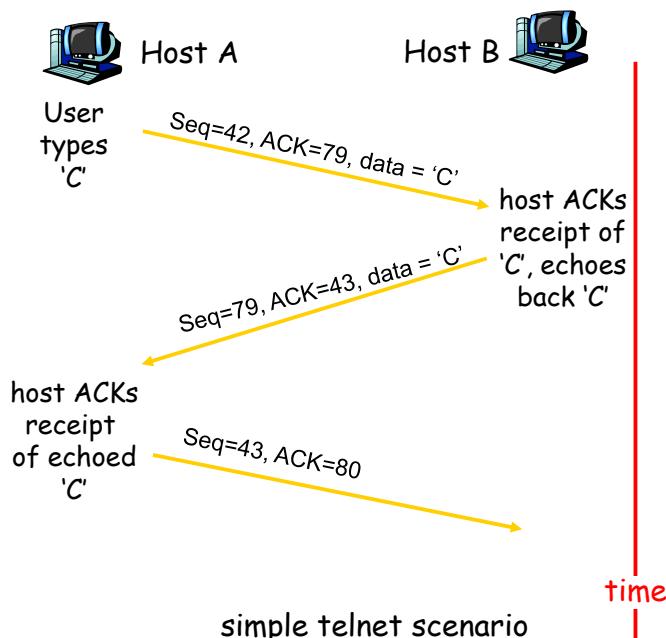
## TCP Sequence Number and Acknowledgement Number (ACK)

### Seq. #'s:

- byte stream  
“number” of first byte in segment’s data

### ACKs:

- seq # of next byte expected from other side
- cumulative ACK



- When Segment carries a combination of data and control information called “Piggybacking”

72

## TCP Round Trip Time and Timeout

### Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- too short:  
premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

### Q: how to estimate RTT?

- **SampleRTT:** measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want **estimated RTT** “smoother”
  - average several recent measurements, not just current **SampleRTT**

73

# TCP Round Trip Time and Timeout

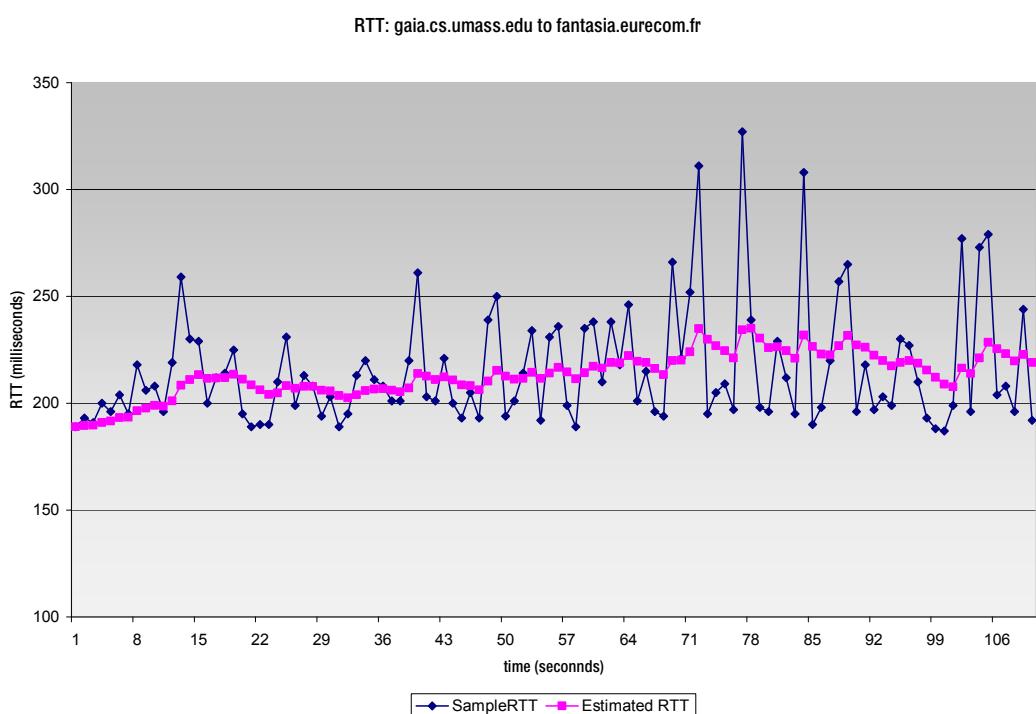
( $\alpha$  է առաջարկություն ունեցած ամենամեծ անդամագույնը)

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value:  $\alpha = 0.125$  [ $(1-\alpha) = 0.875$ ]

74

## Example RTT estimation:



75

何で? ?

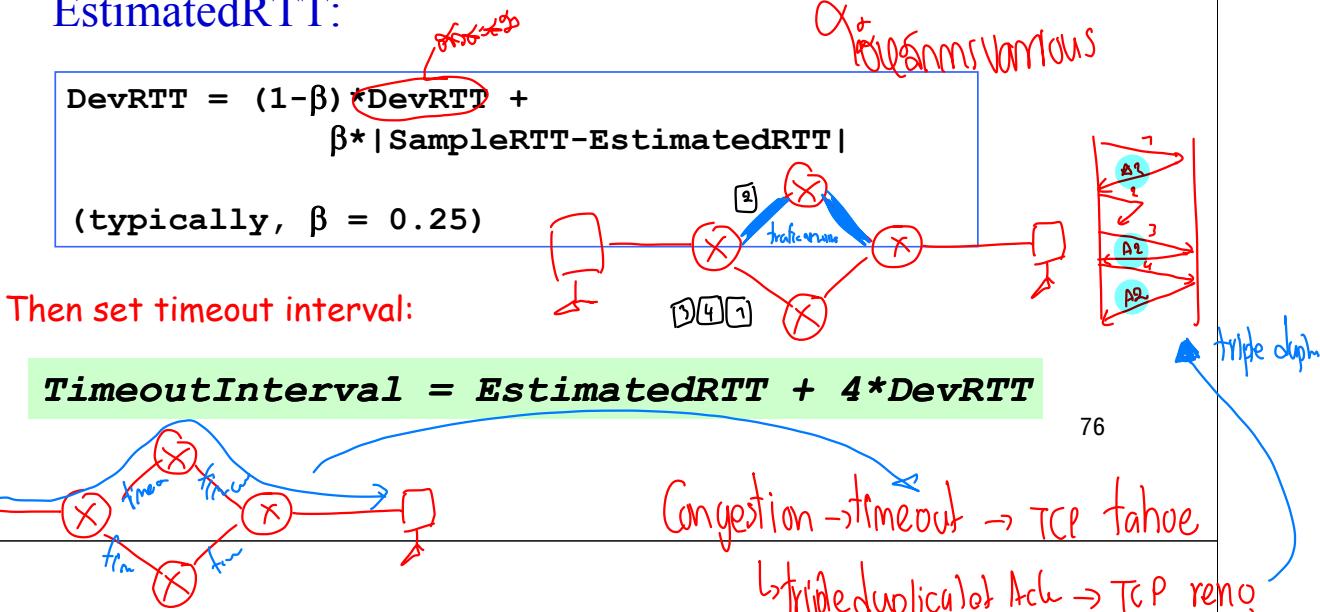
# TCP Round Trip Time and Timeout

## ■ Setting the timeout

### ■ EstimatedRTT plus “safety margin”

- large variation in EstimatedRTT -> larger safety margin

### ■ first estimate of how much SampleRTT deviates from EstimatedRTT:



# TCP reliable data transfer

- TCP creates reliable data transfer (rdt) service on top of IP's unreliable service
- Pipelined segments
- Cumulative acknowledges
- TCP uses single retransmission timer

- Retransmissions are triggered by:
  - **Timeout events**
  - **Duplicate acknowledges**
- Initially consider simplified TCP sender:
  - ignore duplicate acknowledges
  - ignore flow control, congestion control

# TCP sender events:

event ကို အသုတေသနသူ၏ program

data seg

ack received

timeout

acks layer

## Data received from app:

- Create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running (think of timer as for oldest unacked segment)
- expiration interval: TimeOutInterval

## Timeout:

- retransmit segment that caused timeout
- restart timer

## Ack received:

- If acknowledges previously unacked segments
  - update what is known to be acked
  - start timer if there are outstanding segments

78

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
```

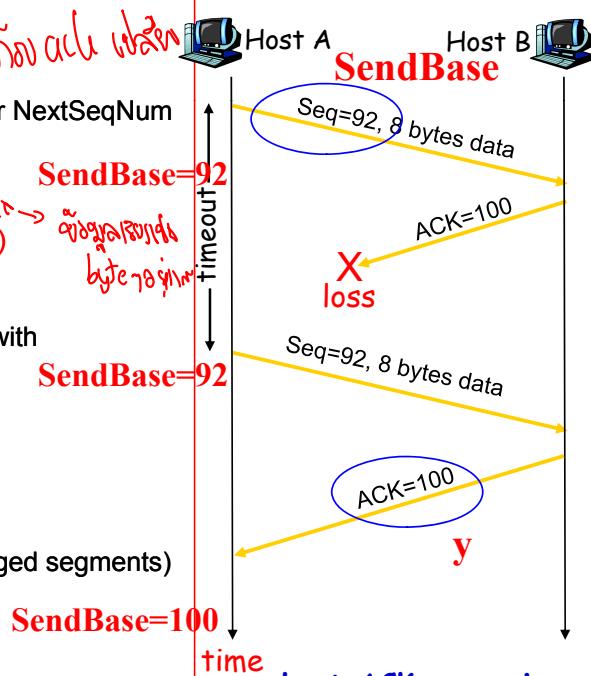
```
loop (forever) {
 switch(event)
```

event: data received from application above  
 create TCP segment with sequence number NextSeqNum  
 if (timer currently not running)  
 start timer  
 pass segment to IP  
 $NextSeqNum = NextSeqNum + \text{length(data)}$   
 event: timer timeout  
 retransmit not-yet-acknowledged segment with smallest sequence number  
 start timer

```
event: ACK received, with ACK field value of y
if (y > SendBase) {
 SendBase = y
 if (there are currently not-yet-acknowledged segments)
 start timer
}
```

```
} /* end of loop forever */
```

TCP sender  
(simplified)



79

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
```

```
loop (forever) {
 switch(event)
```

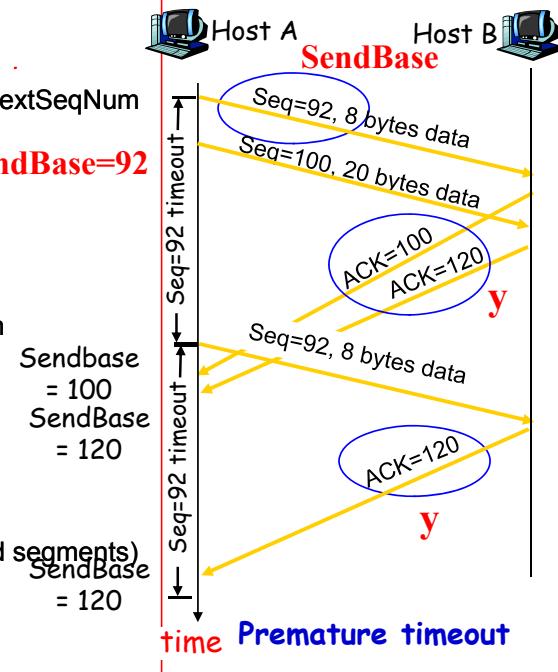
```
 event: data received from application above
 create TCP segment with sequence number NextSeqNum
 if (timer currently not running)
 start timer
 pass segment to IP
 NextSeqNum = NextSeqNum + length(data)
```

```
 event: timer timeout
 retransmit not-yet-acknowledged segment with
 smallest sequence number
 start timer
```

```
 event: ACK received, with ACK field value of y
 if (y > SendBase) {
 SendBase = y
 if (there are currently not-yet-acknowledged segments)
 start timer
 }
```

```
} /* end of loop forever */
```

## TCP sender (simplified)



80

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
```

```
loop (forever) {
 switch(event)
```

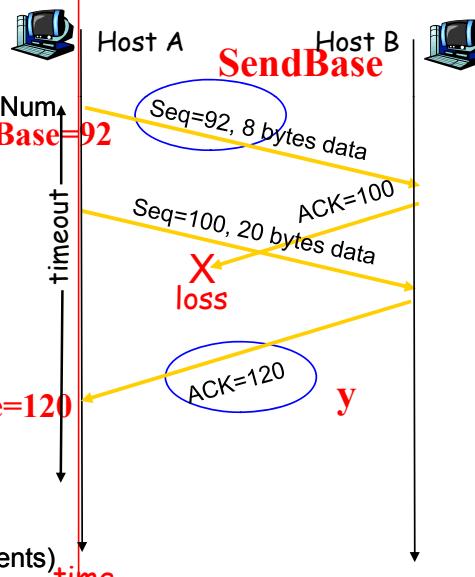
```
 event: data received from application above
 create TCP segment with sequence number NextSeqNum
 if (timer currently not running)
 start timer
 pass segment to IP
 NextSeqNum = NextSeqNum + length(data)
```

```
 event: timer timeout
 retransmit not-yet-acknowledged segment with
 smallest sequence number
 start timer
```

```
 event: ACK received, with ACK field value of y
 if (y > SendBase) {
 SendBase = y
 if (there are currently not-yet-acknowledged segments)
 start timer
 }
```

```
} /* end of loop forever */
```

## TCP sender (simplified)



Cumulative ACK scenario

81

# Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - **Fast Retransmit:** resend segment before timer expires

82

## Fast retransmit algorithm:

```
event: ACK received, with ACK field value of y
if (y > SendBase) {
 SendBase = y
 if (there are currently not-yet-acknowledged segments)
 start timer
}
else {
 increment count of dup ACKs received for y
 if (count of dup ACKs received for y = 3) {
 resend segment with sequence number y
 }
}
```

*duplicate ack'na 3 टा  
प्राप्तिमंगस्तुत्येष्वा*

a duplicate ACK for  
already ACKed segment

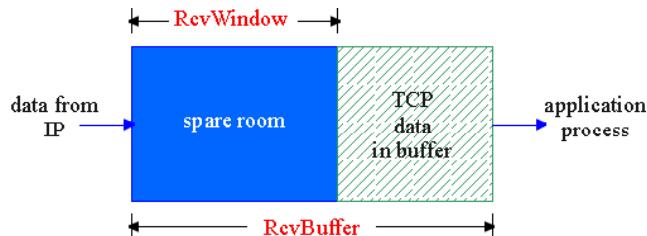
fast retransmit

83

# TCP Flow Control

- receive side of TCP connection has a receive buffer:

**Flow Control**  
sender won't overflow receiver's buffer by transmitting too much, too fast

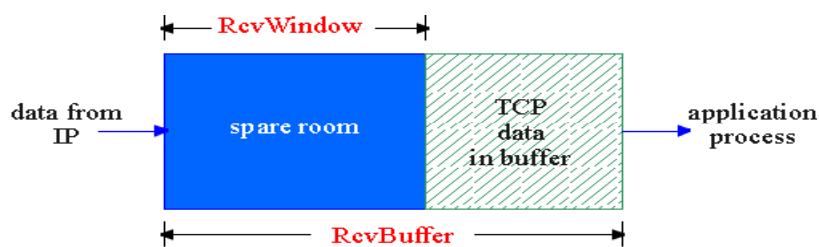


- application process may be slow at reading from buffer

- speed-matching service: matching the send rate to the receiving app's drain rate

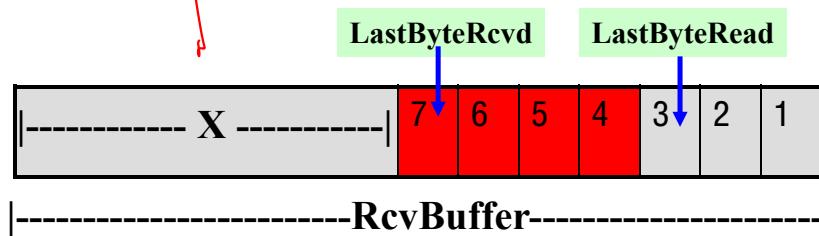
84

## TCP Flow control: how it works



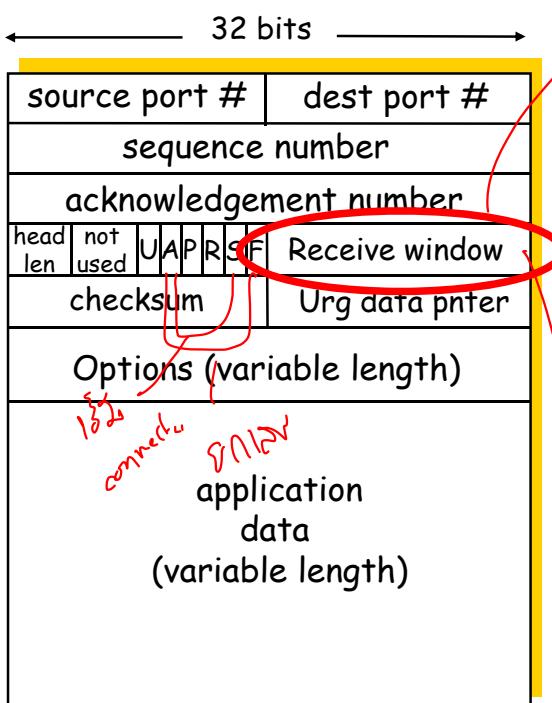
(Suppose TCP receiver discards out-of-order segments)

**Spare Room in Buffer = RcvWindow** → Field ନିବେଦିତ  
= RcvBuffer-[LastByteRcvd - LastByteRead]



85

# TCP Flow control: how it works



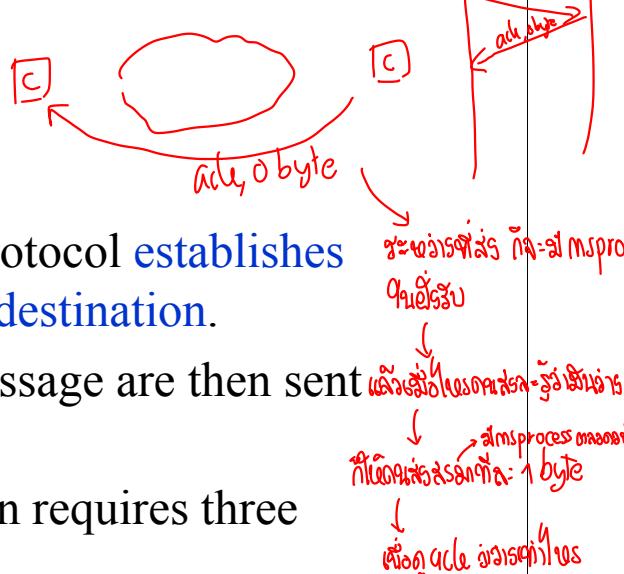
- Receiver advertises spare room by including value of **RcvWindow** in segments
- Sender limits unACKed data to **RcvWindow**
  - guarantees receive buffer doesn't overflow

86

ពាណិជ្ជកម្មណ៍បានចេញលក់ (byte) នៃ

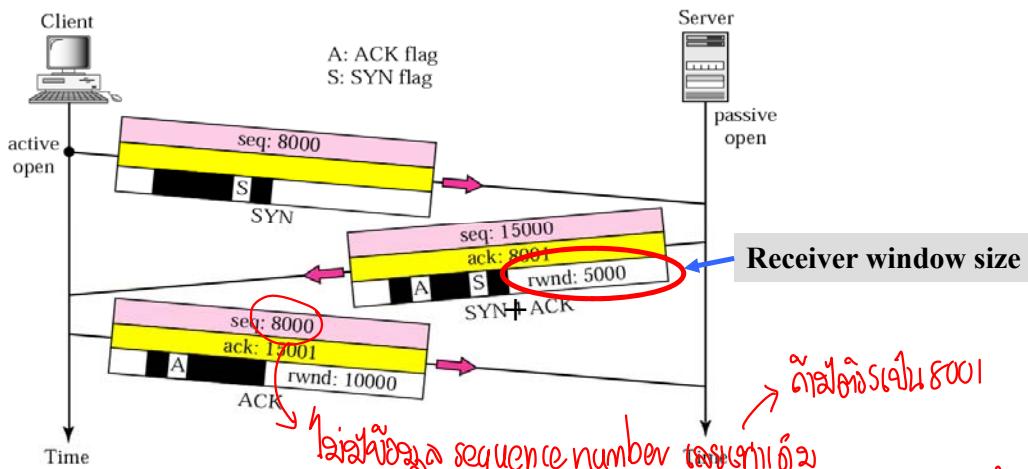
## TCP Connection

- TCP is **connection-oriented**.
- Connection-oriented transport protocol **establishes virtual path** between **source and destination**.
- All of segments belonging to message are then sent over this virtual path.
- Connection-oriented transmission requires three phases:
  - Connection Establishment**,
  - Data transfer**, and
  - Connection Termination**.



87

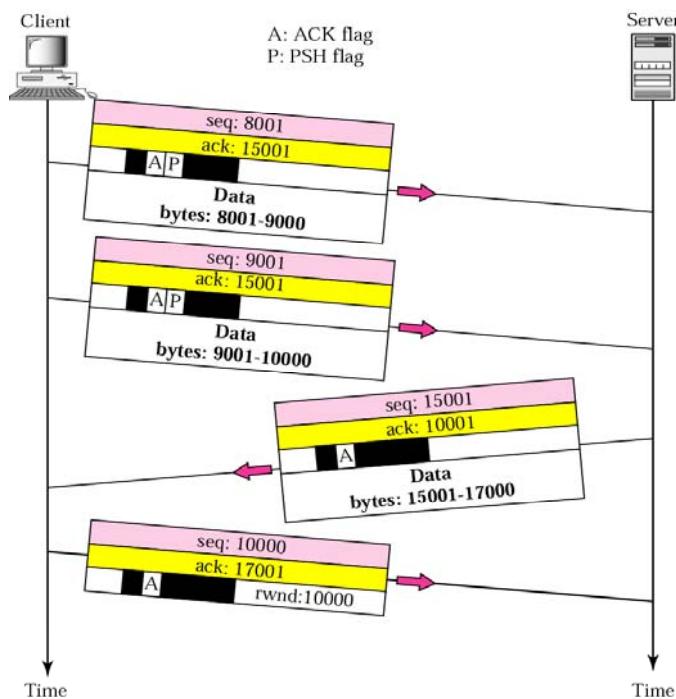
# Connection Establishment using Three-Way Handshaking



- **SYN segment cannot carry data, but it consumes one sequence number.**
- **SYN + ACK segment cannot carry data, but does consume one sequence number**
- **ACK segment, if carrying no data, consumes no sequence number.**

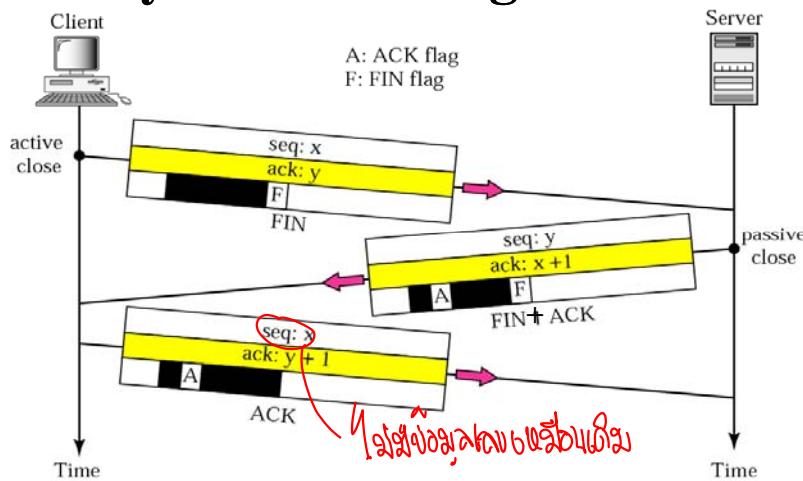
88

## Data Transfer



89

# Connection termination using three-way handshaking

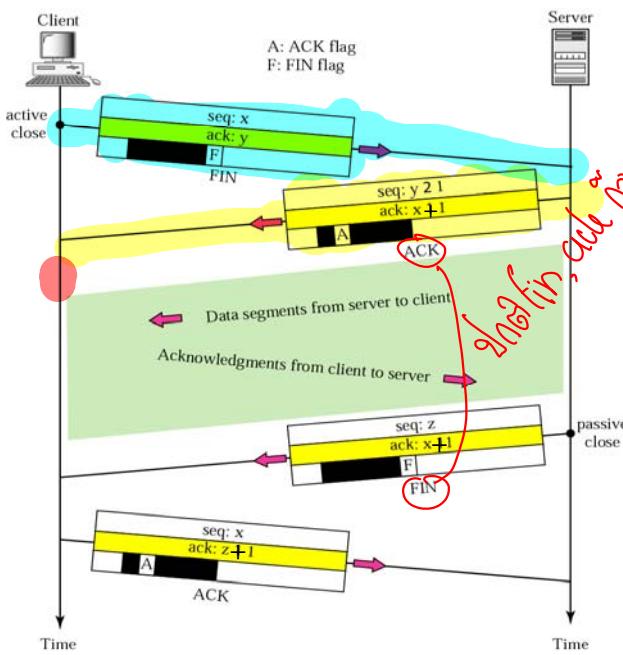


- **FIN segment consumes one sequence number if it does not carry data.**
- **FIN + ACK segment consumes one sequence number if it does not carry data**
- **ACK segment, if carrying no data, consumes no sequence number**

90

1. **Client sends its FIN ms sequence**

## Half-Close



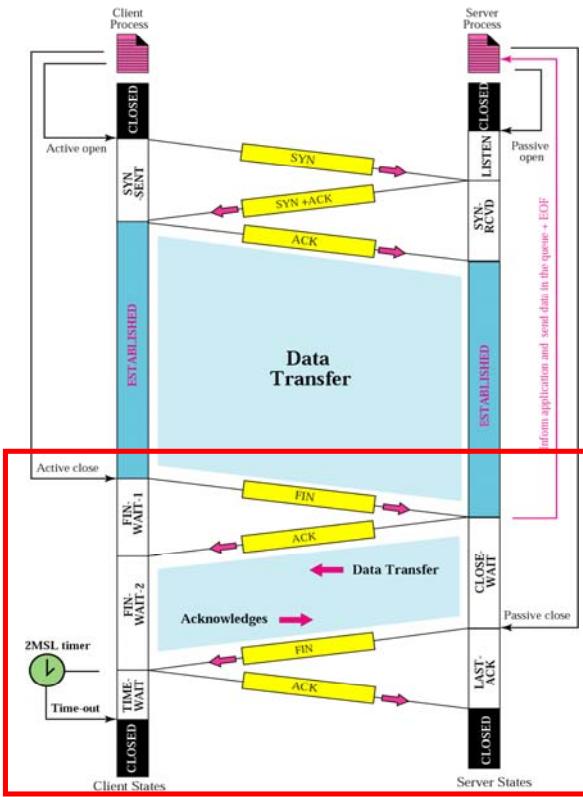
- In TCP, one end can stop sending data while still receiving data called **Half-close**

Client half-close connection by sending FIN segment

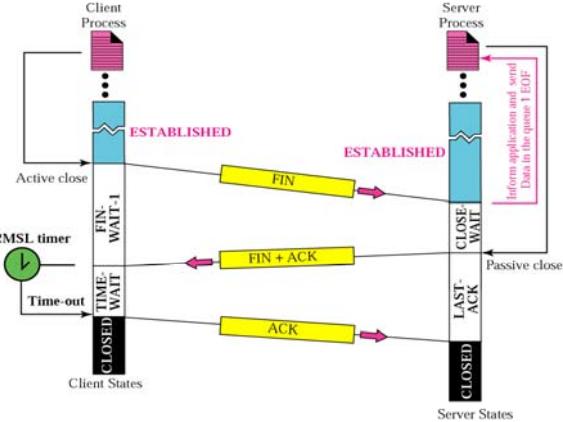
- Server accepts this half-close by sending ACK segment
- Data transfer from client to server stops
- Server can still send data
- After sending all data, server sends FIN segment, which is acknowledged by ACK from client

91

# Connection Establishment and Termination



- The common value of MSL (Maximum Segment Lifetime) is between 30 seconds and 1 minutes



- Three-way Handshake

92

## Principles of Congestion Control

### Congestion:

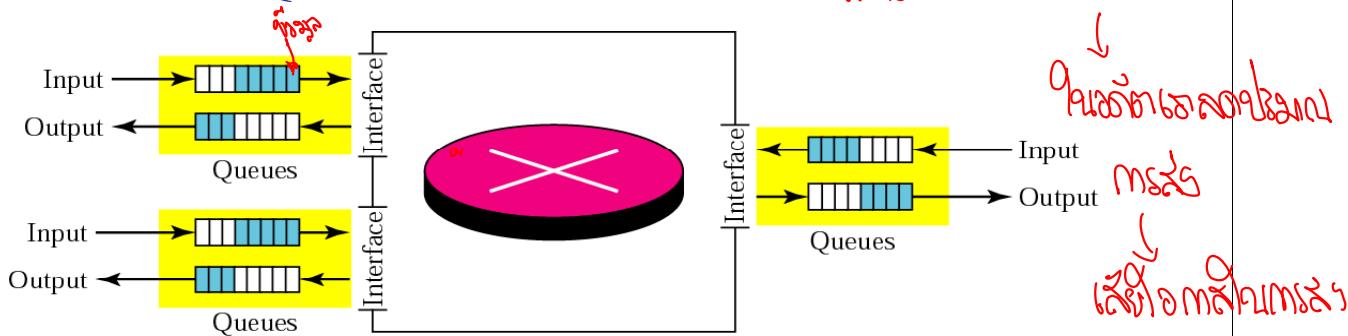
- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control! → *ack, nak* စုတေသန
- manifestations: (မြန်မြတ်)
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- Congestion control refers to mechanisms and techniques to keep load below capacity.**



97

ວິເຣີສະໝັກ wireless ຂາດ່າວັນຍົກ ກົດໃຈ noise ສ້າງໂທ ໃຫ້ ໄຕສູງທາງໄລ່ໂທ  
ໃຫ້ເກີດໃຫຍ່ ເກີດໃຫຍ່

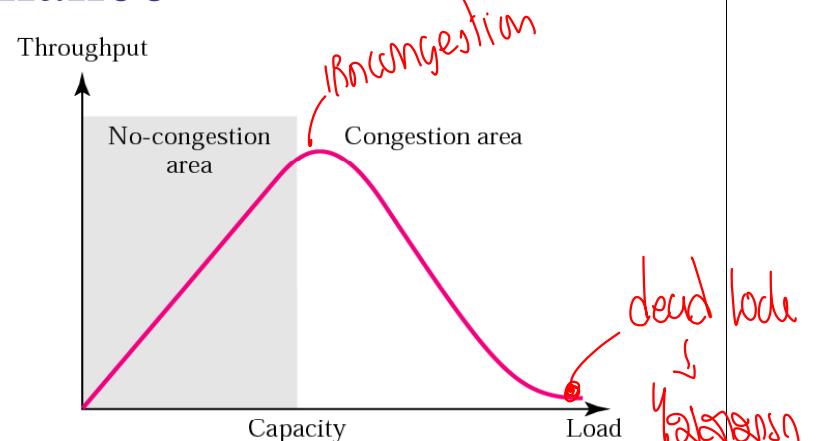
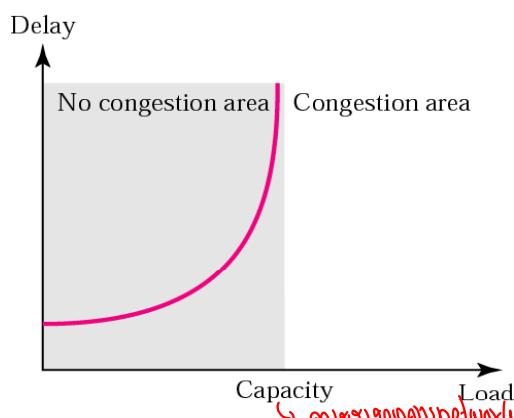
## Router Queues



- Packet is placed at end of input queue while waiting to be checked
- Processing module of router removes packet from input queue one it reaches the front of queue and uses its routing table and destination address to find route
- Packet is put in appropriate output queue and waits its turn to be sent

98

## Network Performance



- Congestion control involves two factors that measure performance of network

- Delay
- Throughput – number of packets passing through network in a unit of time

101

# Approaches towards congestion control

## Two broad approaches towards congestion control:

### 1. Network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN (Explicit Congestion Notification), ATM)

### 2. End-end congestion control:

- no explicit feedback from network → ~~network monitor~~
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

ກວດສອນທີ່ມີການຍົງຍາຍໃນການ

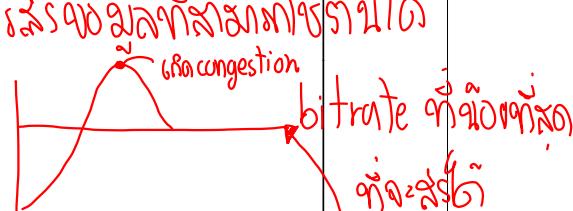
105

asynchronous transfer mode

## Case study: ATM ABR congestion control

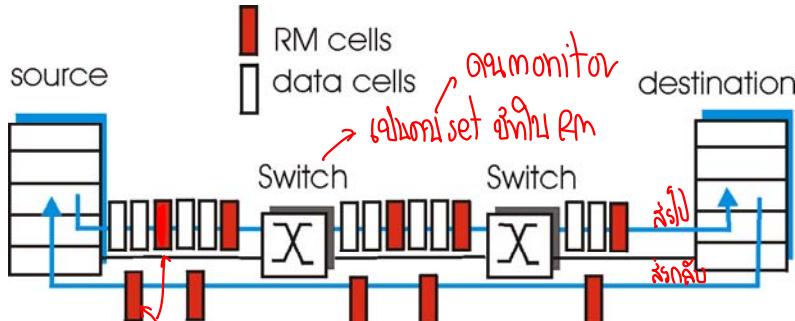
### ABR: Available Bit Rate:

- “elastic service”
- if sender’s path “underloaded”:
  - sender should use available bandwidth
- if sender’s path congested:
  - sender throttled to minimum guaranteed rate



106

## Case study: ATM ABR congestion control



**RM (Resource Management) cells:** field ທີ່ມີຢູ່ກວດສອງການເຄື່ອນໄຫວ

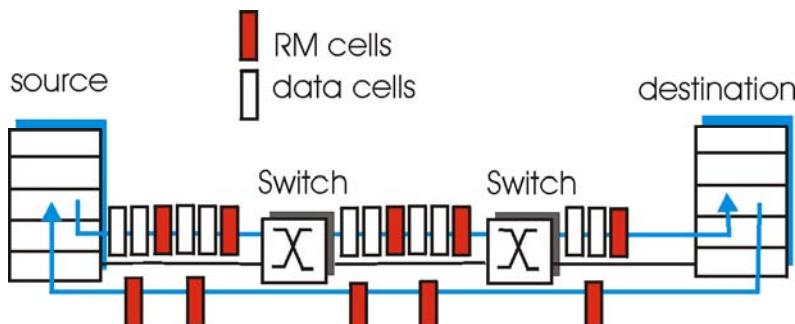
- Sent by sender, interspersed with data cells (one RM cell every 32 data cell)
- bits in RM cell set by switches (“*network-assisted*”)
  - NI bit: no increase in rate (mild congestion)
  - CI bit: congestion indication
- two-byte ER (Explicit Rate) field in RM cell
  - congested switch may lower ER value in cell
  - sender’ send rate thus maximum supportable rate on path
- RM cells returned to sender by receiver, with bits intact

ເພື່ອໃຊ້ set ຂອບ bit rate ເພື່ອ minimum guaranteed bitrate

ກ່າຍລວມຈະໄດ້ກວດສອງການເຄື່ອນໄຫວ

107 11/2021

## Case study: ATM ABR congestion control



### ■ EFCI bit in data cells:

- Each data cell contains an Explicit Forward Congestion Indication (EFCI) bit
- EFCI bit may be set to 1 in congested switch
- if data cell preceding RM cell has EFCI set, receiver sets CI bit in returned RM cell

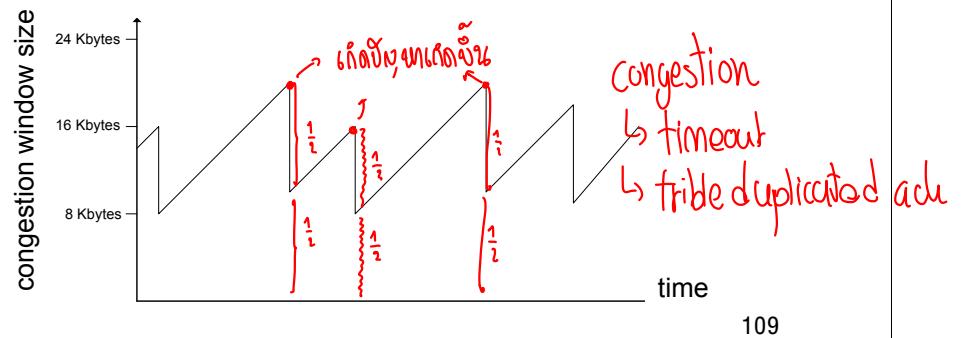
108

## TCP Congestion Control:

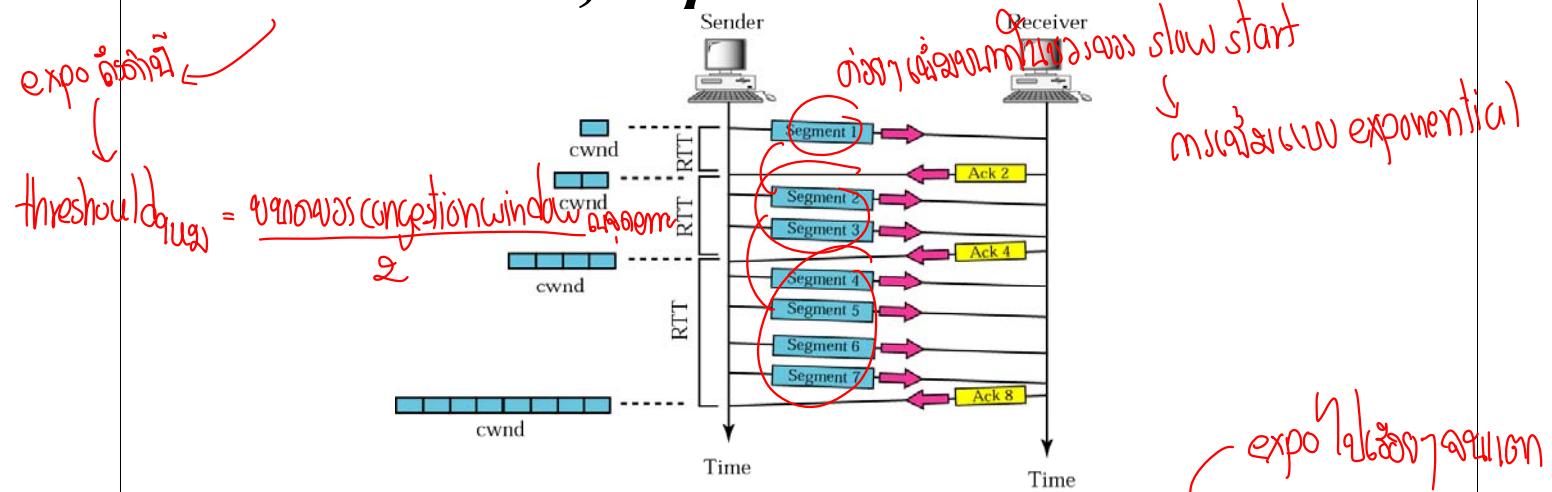
### Additive Increase (AI), Multiplicative Decrease (MD)

- **Approach:** increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - **additive increase:** increase **CongWin** by 1 MSS every RTT until loss detected
  - **multiplicative decrease:** cut **CongWin** in half after loss

Saw tooth behavior: probing for bandwidth



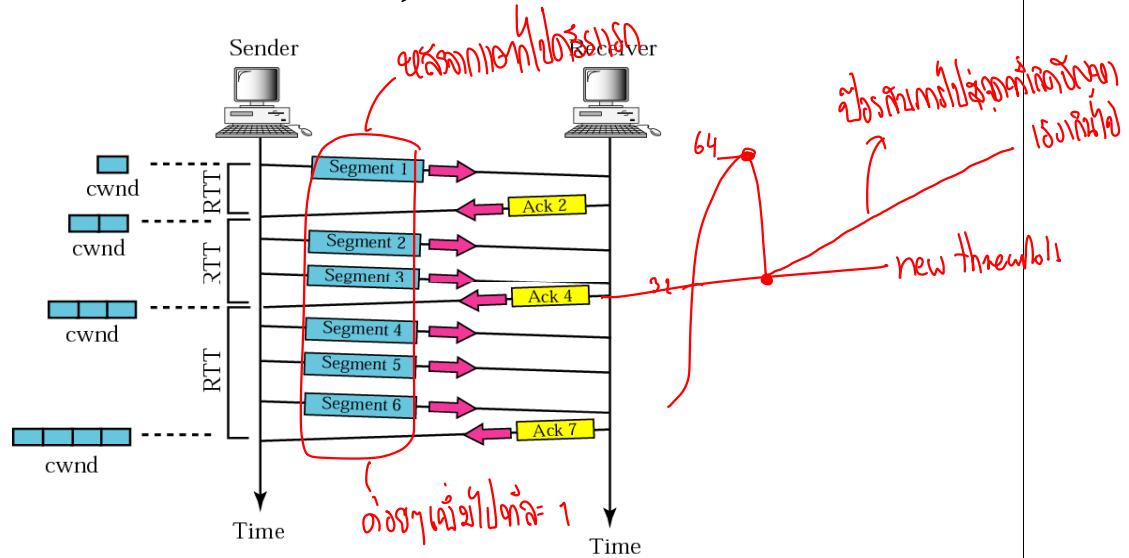
### Slow Start, exponential increase



- When connection begins, **increase rate exponentially** until **first loss event**:
  - double **Congestion Windows** (**cwnd**) every RTT
  - done by **incrementing cwnd** for every **ACK received**
- **Summary:** initial rate is slow but ramps up exponentially fast

110

## phase Congestion Avoidance, additive increase



- In congestion avoidance algorithm size of congestion window (cwnd) increases additively until congestion is detected

111

*Most implementations react differently to congestion detection:*

- If detection is by time-out, a new slow start phase starts
- If detection is by three ACKs, a new congestion avoidance phase starts

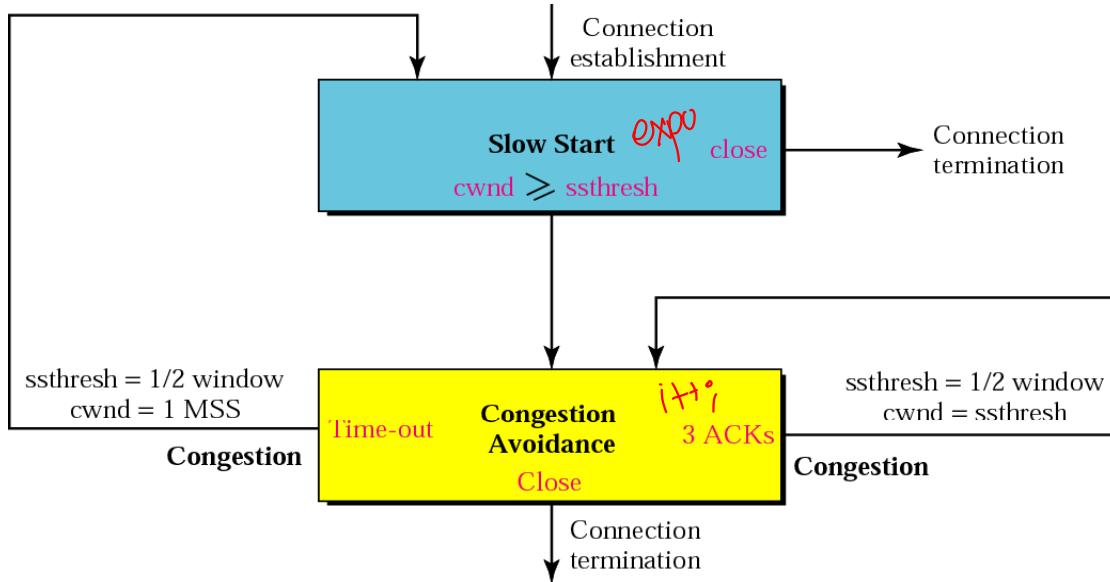
### Mechanism

- AIMD (Additive Increase, Multiplicative Decrease)
- Slow Start (SS)
- Congestion Avoidance

ଯେଉଁପରିବନ୍ଧିତ

112

# TCP congestion policy summary

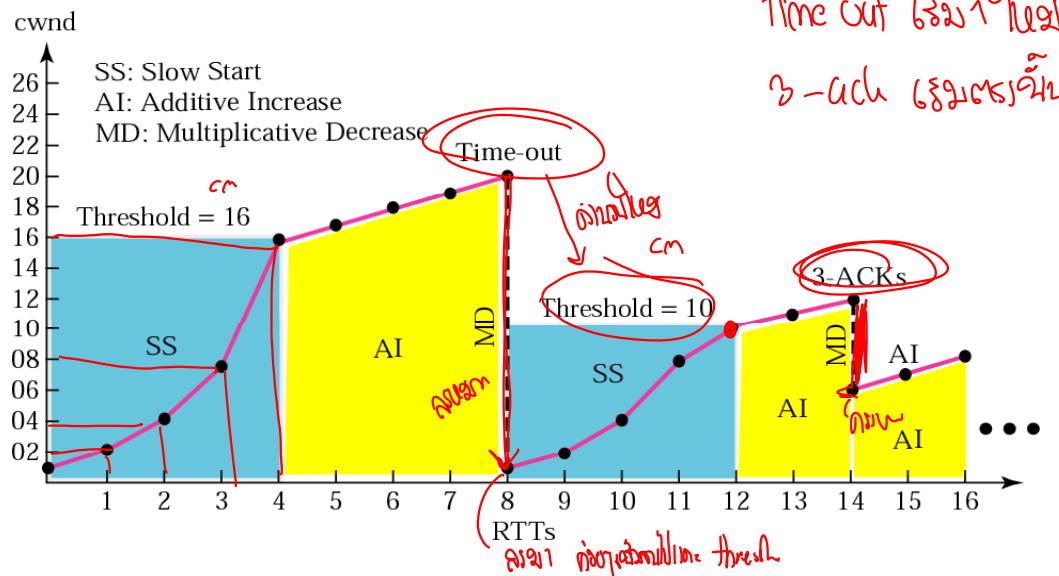


113

## Congestion example

ထုတေသန error threshold  
 $\frac{1}{2}$

time out ပေါ်စွာ အောင် exponentiation  
3-ACK ပေါ်စွာ အောင် exponentiation



114

# Summary: TCP Congestion Control

- When **CongestionWindow(cwnd)** is below **Threshold**, sender in **slow-start** phase, window grows exponentially
- When **cwnd** is above **Threshold**, sender is in **congestion-avoidance** phase, window grows linearly
- When **triple duplicate ACK** occurs, **Threshold** set to **cwnd/2** and **cwnd** set to **Threshold**
- When **timeout** occurs, **Threshold** set to **cwnd/2** and **cwnd** is set to **1 MSS**

120

new threshold = congestion window size on after congestion

2