

CS3640

---

# Transport Layer (2): Reliable Data Transfer

**Prof. Supreeth Shastri**

*Computer Science*

*The University of Iowa*

# Lecture goals

---

*how to achieve reliable data transport over an unreliable network such as the Internet?*

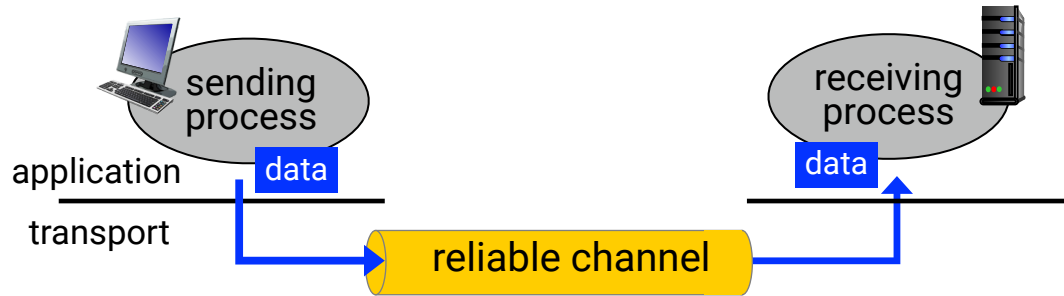
- *Challenges and techniques*
- *Stop-and-Go protocols*
- *Performance analysis*



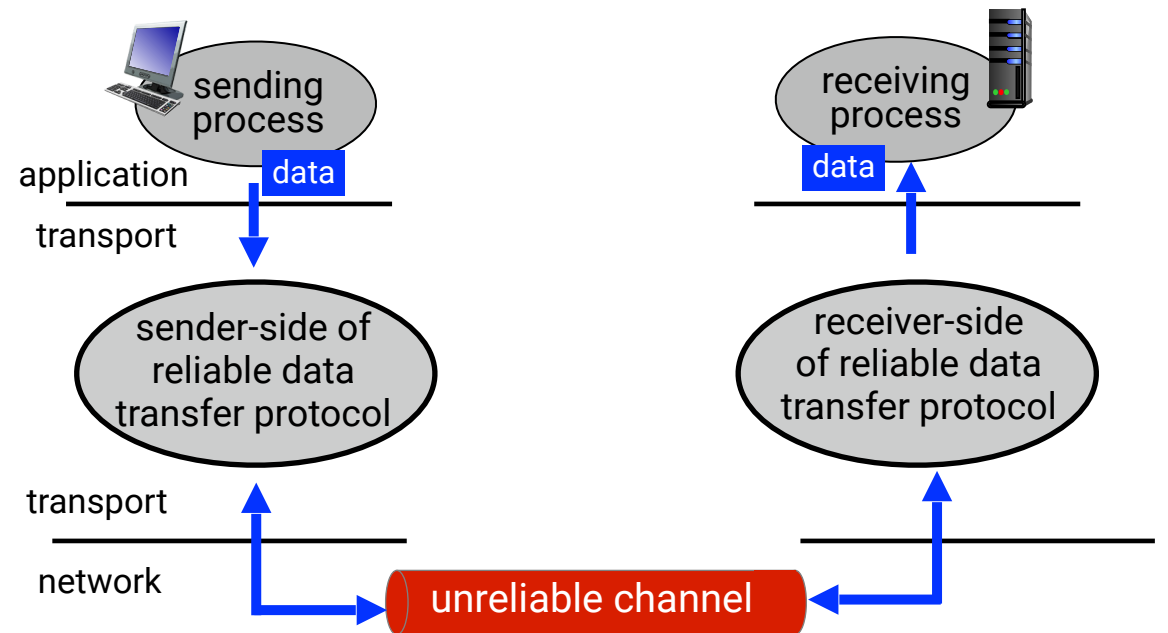
Chapter 3.4

# Principles of reliable data transfer

reliable service abstraction



reliable service implementation



## Key Challenges

Sender, receiver do not know the “state” of each other, or what happens in the channel (unless communicated via a message)

Complexity of reliable data transfer protocol. This depends on the characteristics of unreliable channel (lose, corrupt, reorder data?)

# Designing Reliable Data Transfer (RDT) Protocols

- first, we define characteristics of the underlying unreliable channel
- then, incrementally develop the sender and receiver sides of RDT
- using finite state machines (FSM) to specify the sender and receiver specifications

1

Channel with  
perfect reliability

2

Channel with  
bit errors

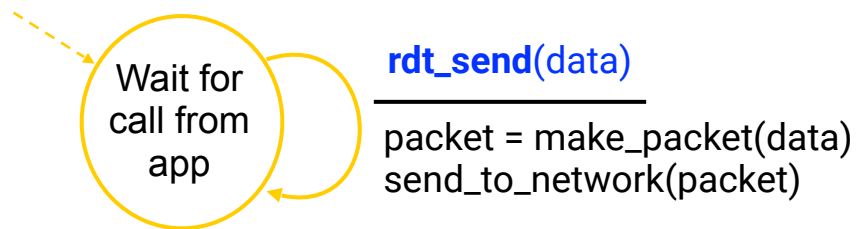
3

Channel with  
bit errors & packet loss

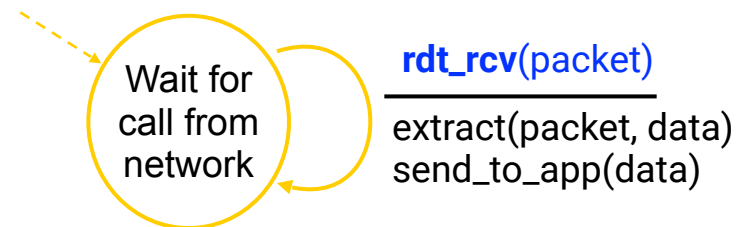
# RDT-1: reliable transport over a perfectly reliable channel

- underlying network channel provides all the required functionality
- So, RDT-1 does not have to handle bit errors, loss of packets, or duplicates
- This renders the sender and receiver FSMs to be simple

## RDT-1 sender



## RDT-1 receiver



# RDT-2: reliable transport over a channel with bit errors

how do we detect bit errors?

- employ **checksums**

how do we recover from errors?

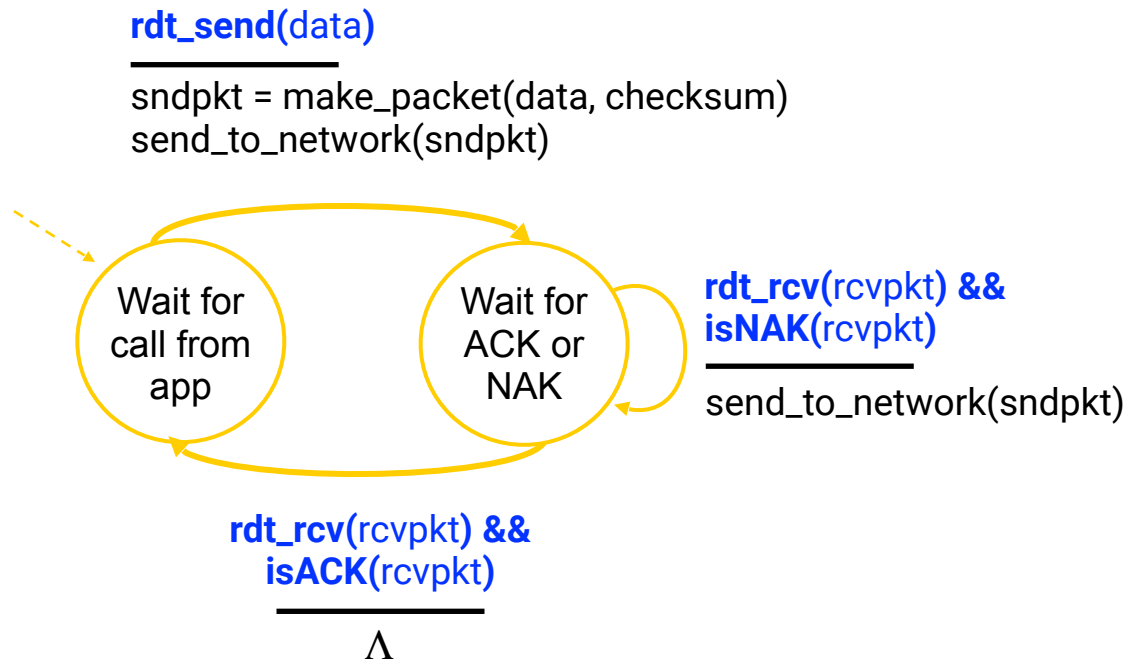
- **acknowledgements (ACKs)**: receiver explicitly tells sender that packet was received OK
- **negative acknowledgements (NAKs)**: receiver explicitly tells sender that packet had errors
- **retransmit** the packet on receipt of NAK

## **stop and wait protocol**

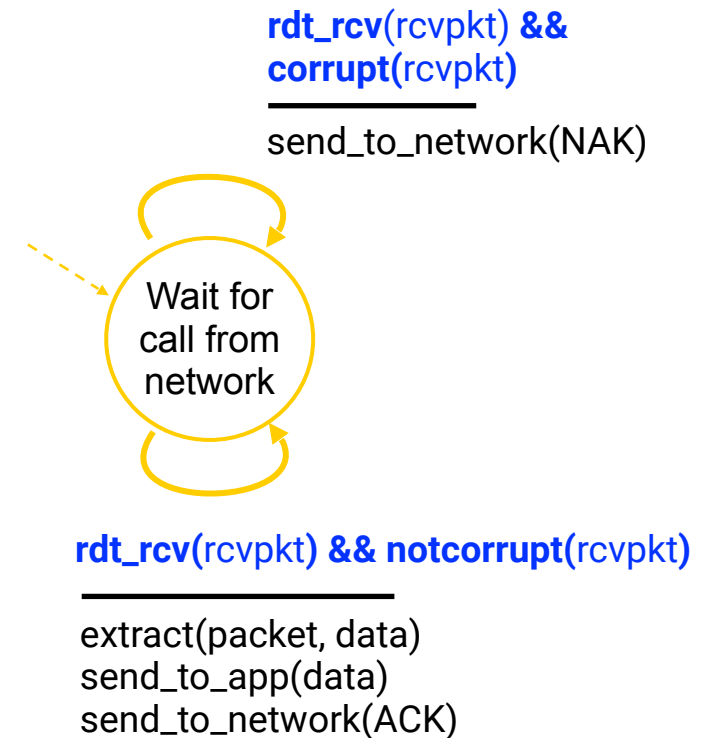
sender sends one packet, then waits for receiver response

# RDT-2: FSM Specifications

## RDT-2 sender



## RDT-2 receiver





# RDT-2 has a serious flaw!

## What happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- sender can't just retransmit as it may result in duplicate packets

## Changes to the Sender

- must check if received ACK/NAK corrupted
- add a sequence number to packets; however, only two seq# (0,1) will suffice. Why?
- results in twice as many states as before

Propose **RDT-2.1** with two changes:

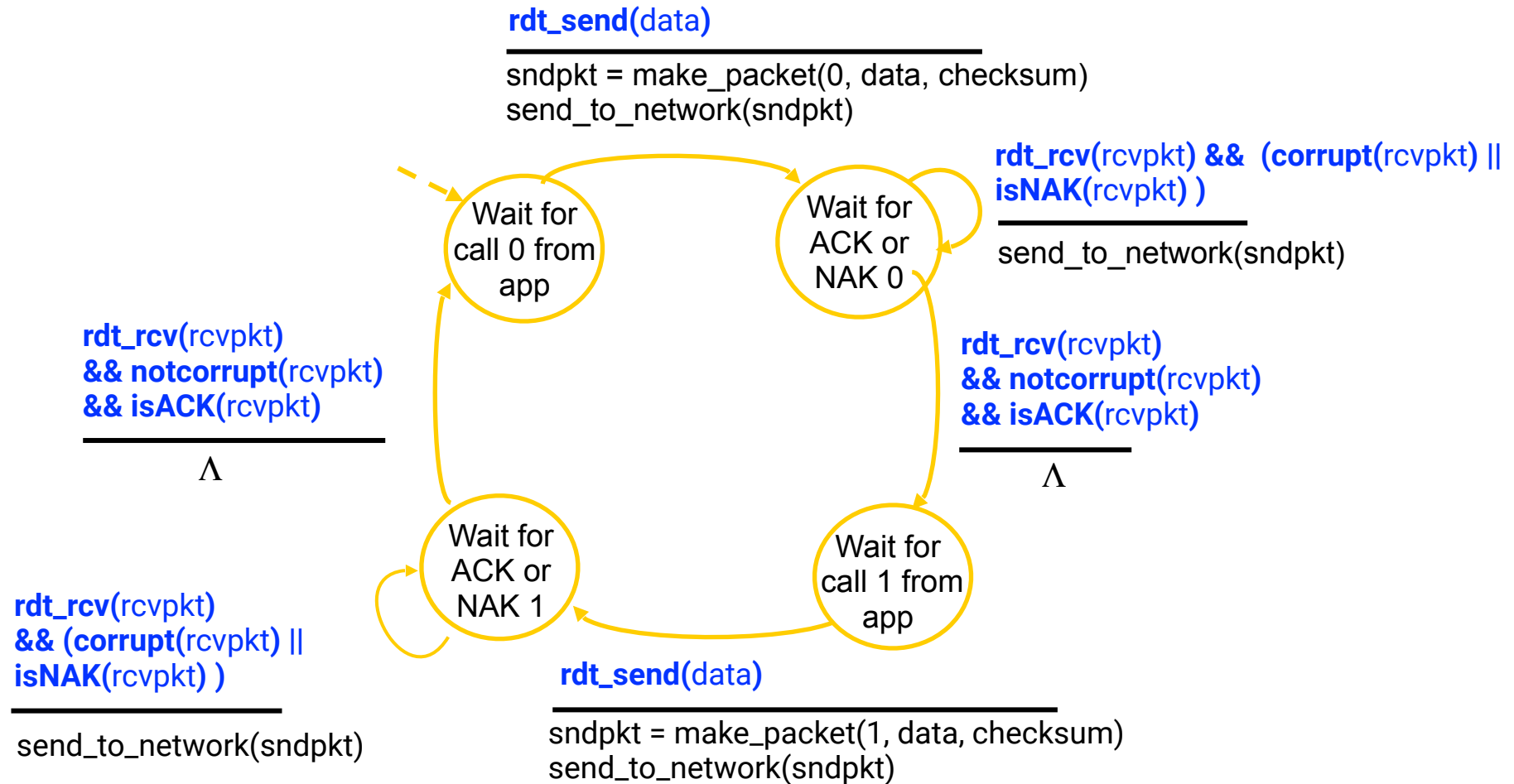
1. *retransmit on corrupted ACK/NAK*
2. *use sequence# to detect duplicates*

## Changes to the Receiver

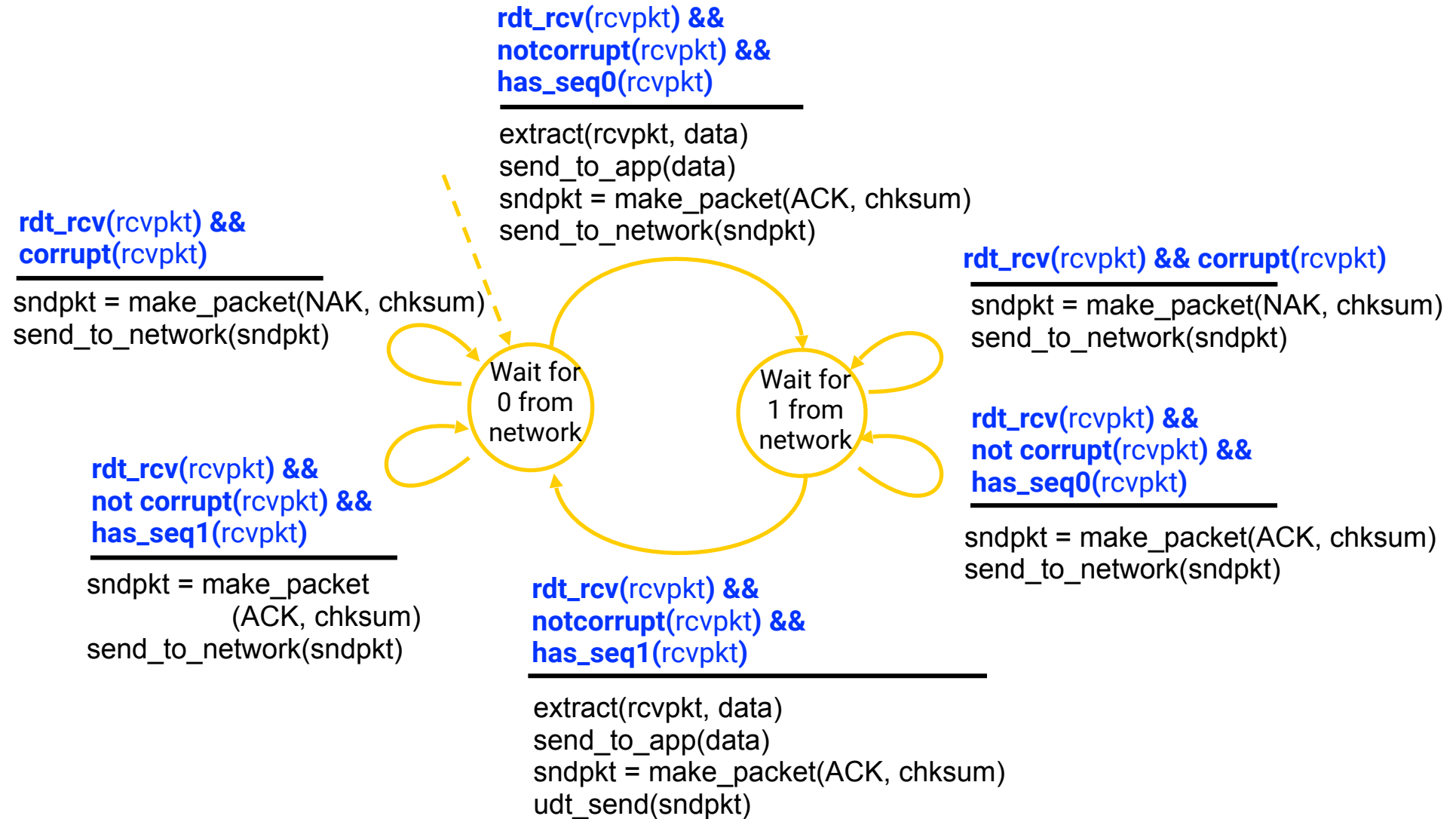
- must check if received packet is a duplicate (using sequence numbers)
- receiver won't know if its last ACK/NAK received OK at sender



# RDT-2.1: sender, handling garbled ACK/NAKs



# RDT-2.1: receiver, handling garbled ACK/NAKs



# RDT-3: reliable transport over a channel with errors and loss

Could sender and receiver recover from packet losses with existing mechanisms?

- sequence#, ACK, and retransmissions will help but not quite enough

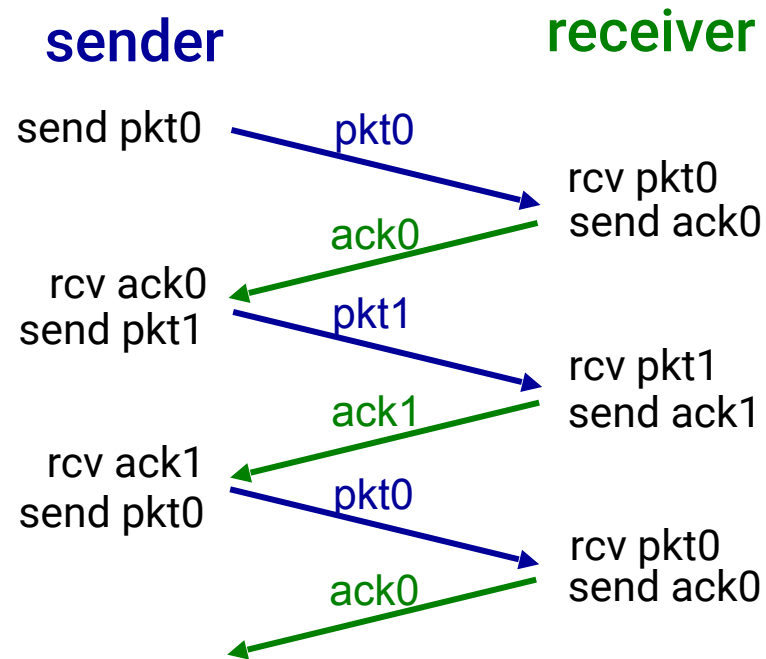
## Approach

- Modeled after the human behavior... the notion of *timeout and retry*
- sender waits a *reasonable* amount of time for ACK; if no ACK is received in that time, *retransmits* the packet again
- receiver already has the mechanism to check for duplicates (via sequence#); however, since multiple ACK can be on the wire now, they must carry the *sequence# of the packet being ACKed*

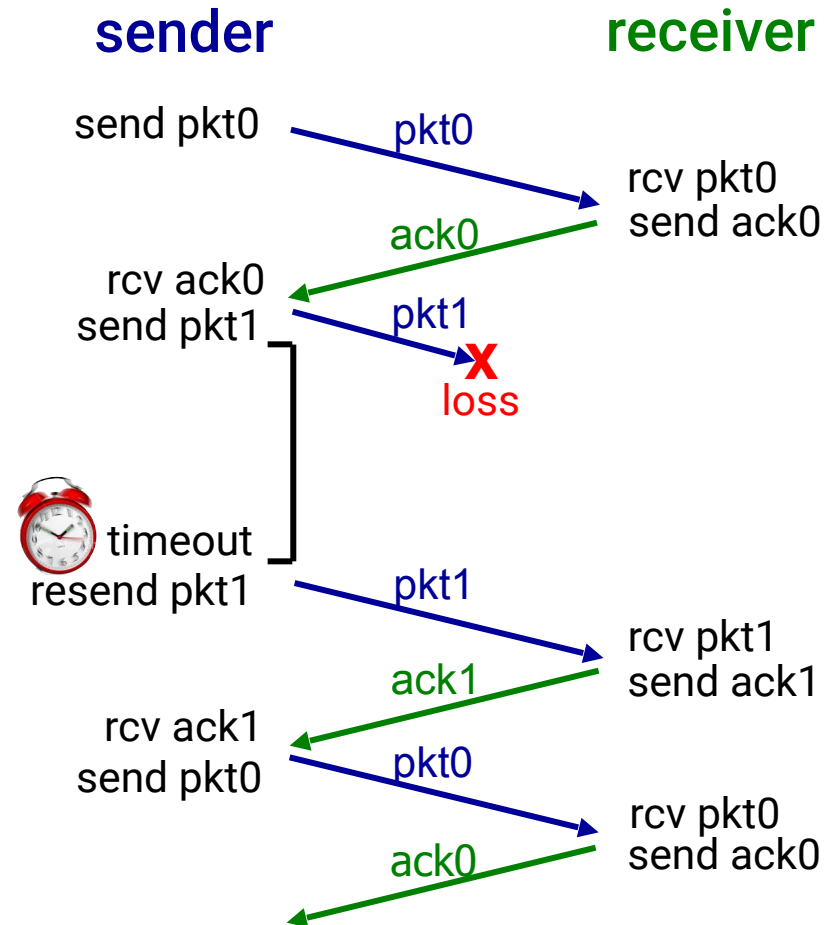


# RDT-3 in action

## Case-1: No loss

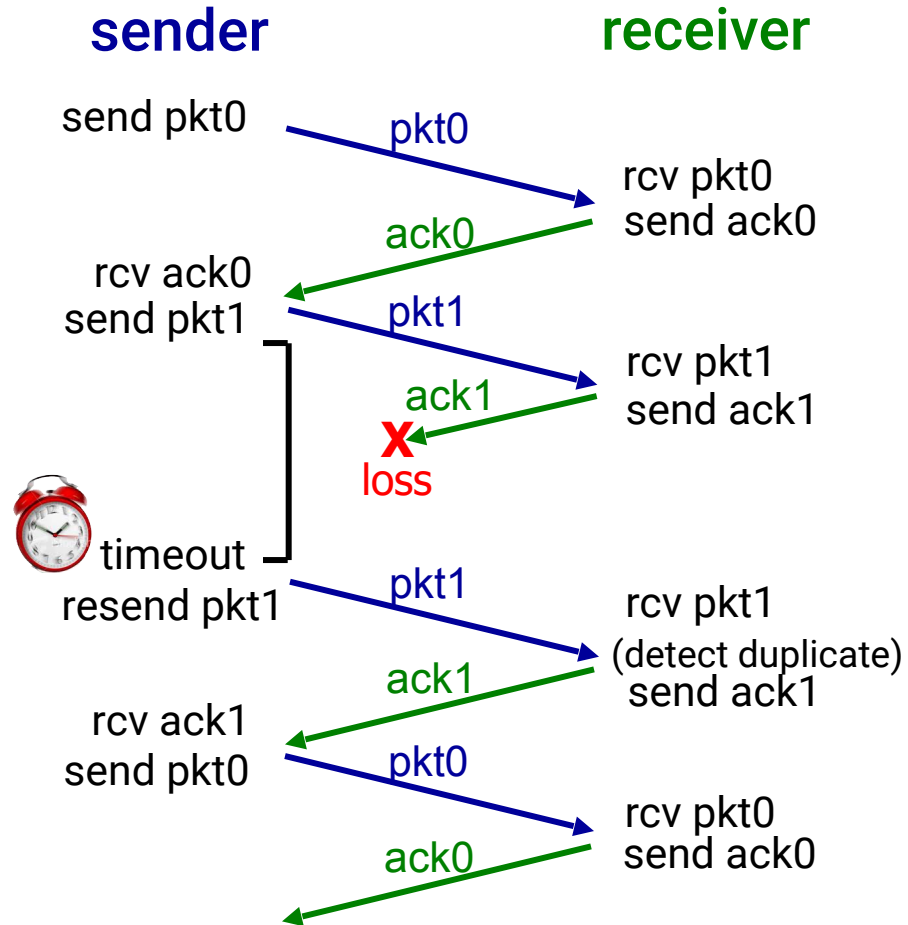


## Case-2: Packet loss

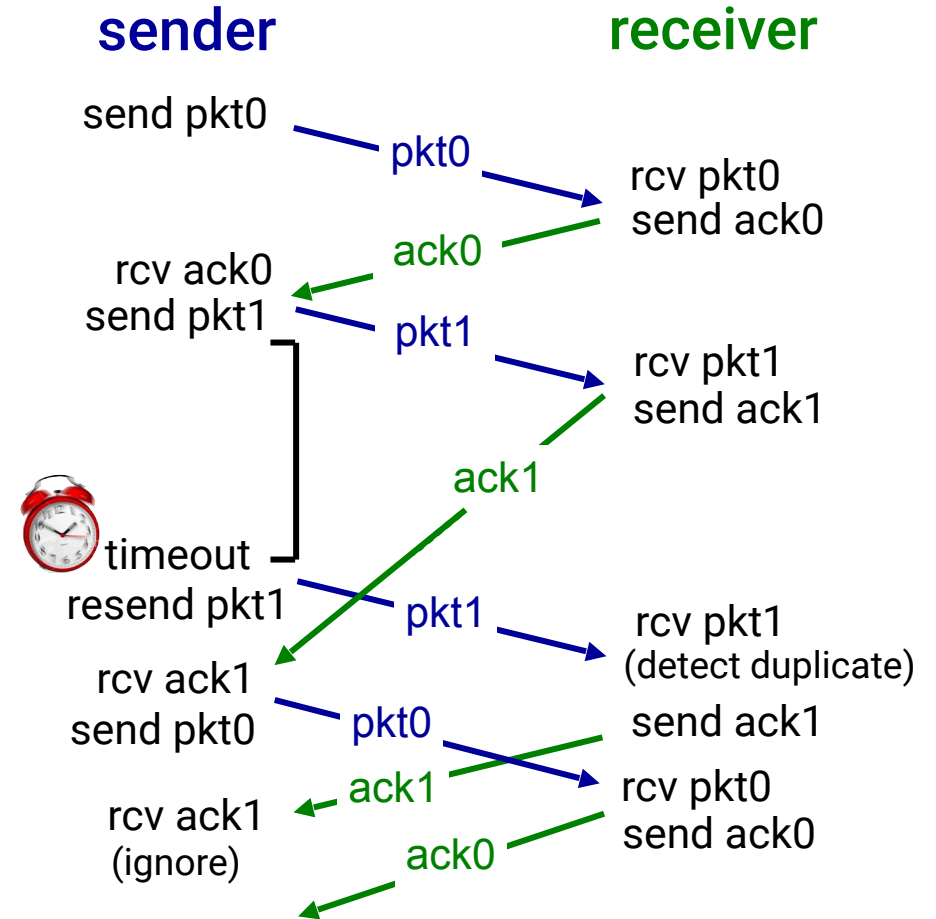


# RDT-3 in action

## Case-3: ACK loss

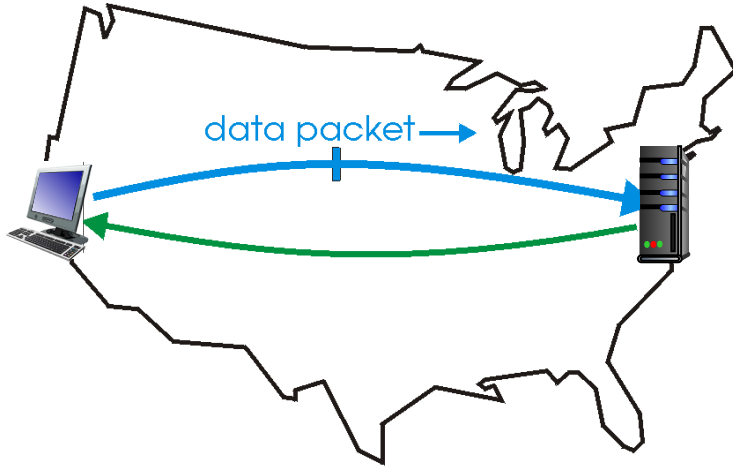


## Case-4: Delayed ACK



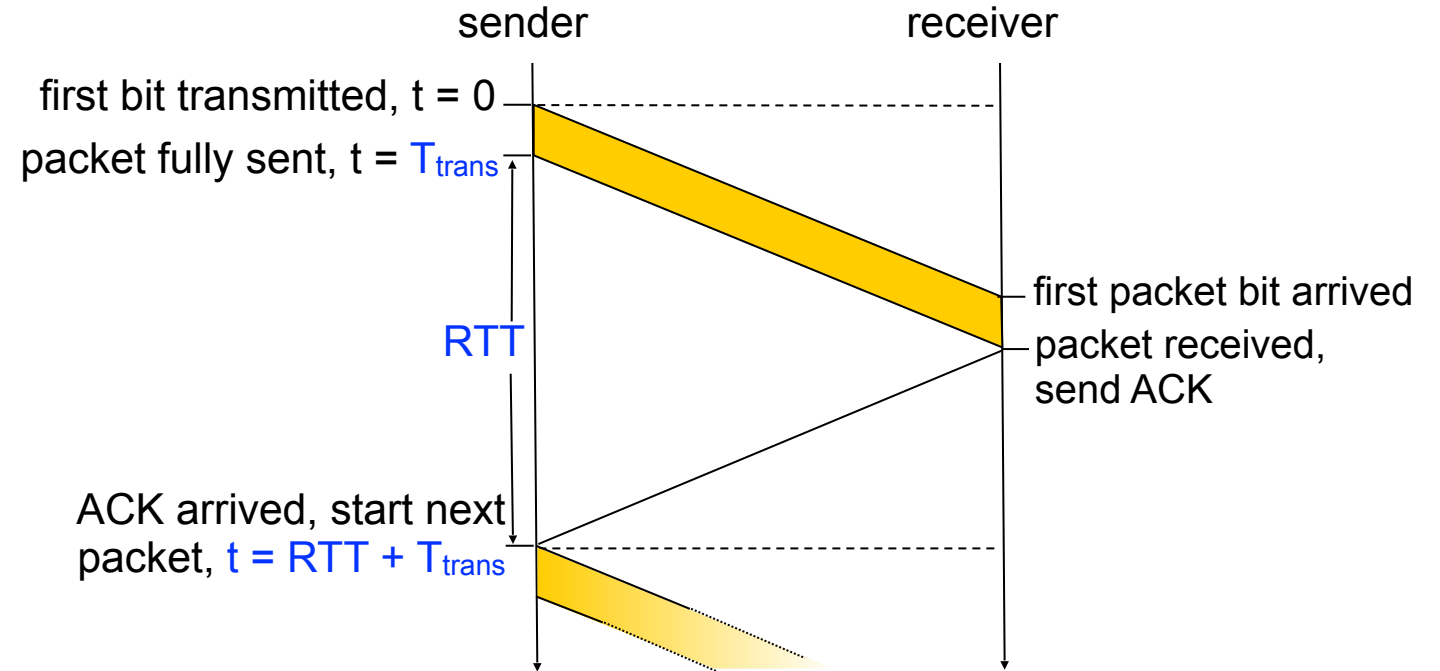
# Performance of **Stop-and-Wait**

# Visualizing stop-and-wait in operation



## An example

- NY-SF propagation,  $D_{\text{prop}} = 15 \text{ ms}$
- Packet length,  $L = 8000 \text{ bits}$
- Transmission rate,  $R = 1 \text{ Gbps}$
- Transmission time,  $T_{\text{trans}} = L/R = 8 \text{ } \mu\text{s}$

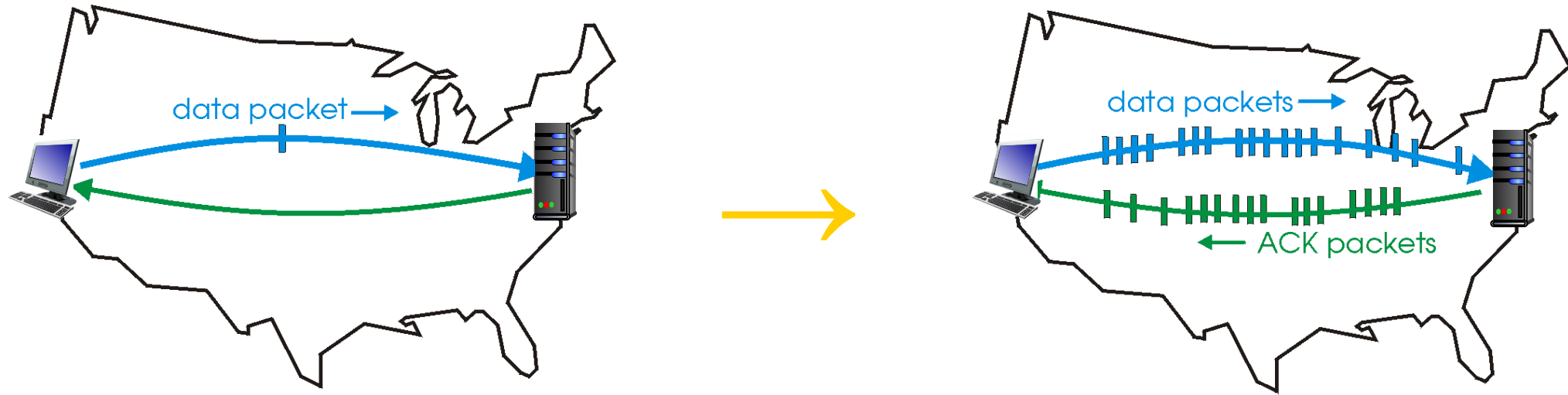


**Sender Channel Utilization:** *fraction of the time bits are sent into the channel*

- Utilization =  $T_{\text{trans}} / (\text{RTT} + T_{\text{trans}})$
- Utilization =  $8 \text{ } \mu\text{s} / 30.008 \text{ ms} = 0.00027 = \mathbf{0.027\%}$



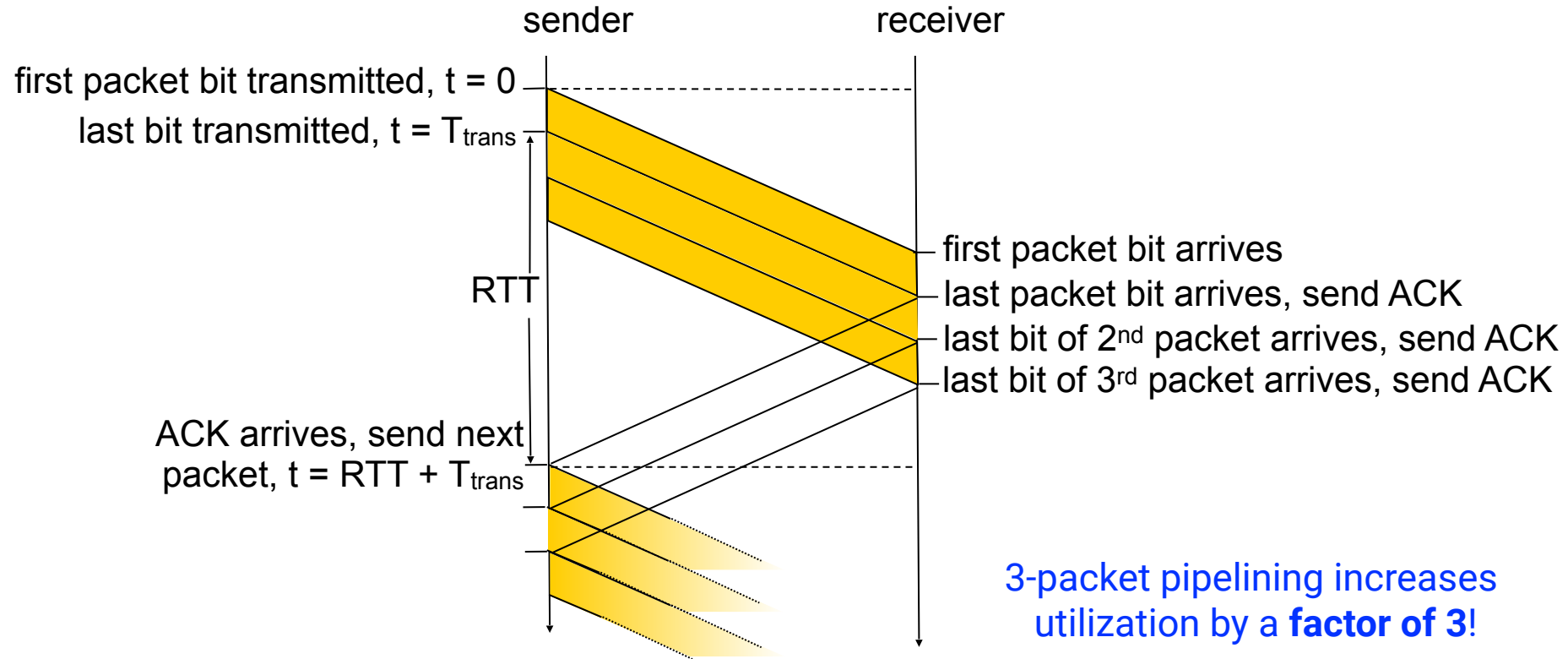
# Increased Utilization w/ Pipelining



## How does pipelining affect RDT-3 protocol?

- range of sequence# must be increased since there may be multiple, in-transit, unACKed packets
- sender and receiver sides need to buffer more than one packet

# Increased Utilization w/ Pipelining



**Sender Channel Utilization:** *fraction of the time bits are sent into the channel*

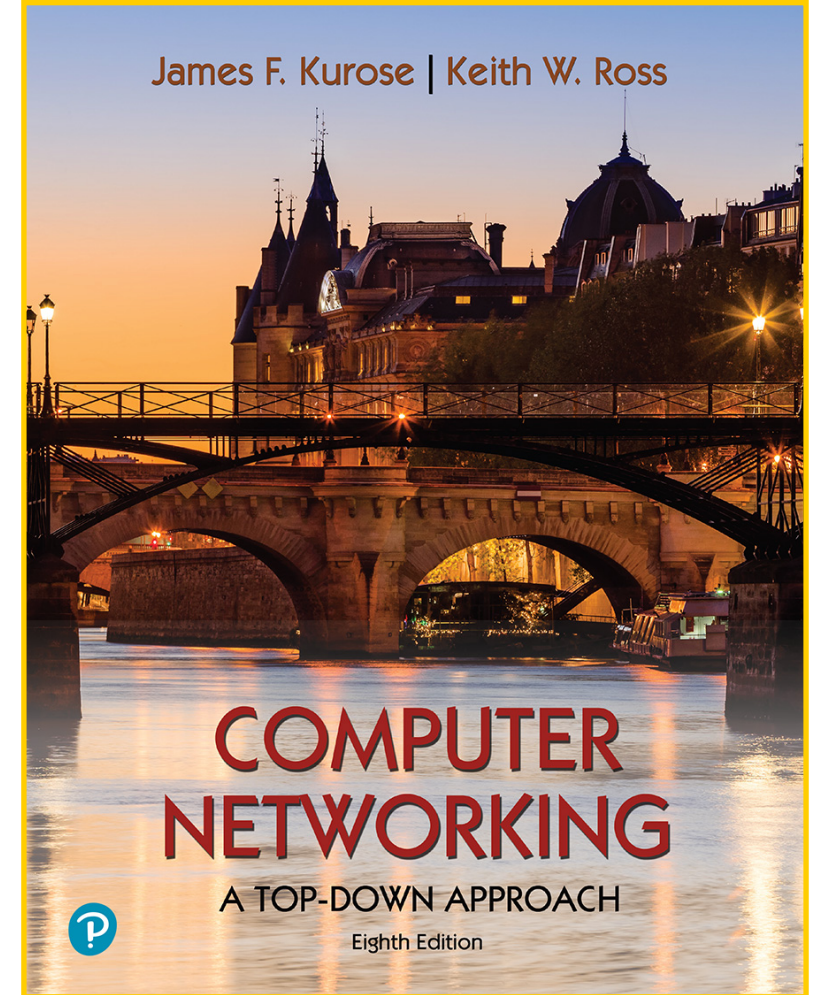
- Utilization =  $3 * T_{\text{trans}} / (\text{RTT} + T_{\text{trans}})$
- Utilization =  $24 \mu\text{s} / 30.008 \text{ ms} = 0.00081 = 0.081\%$

# Next lecture

---

*continued discussion of reliable data transfer, followed by congestion control*

- *Two pipelined RDT protocols*
- *Congestion control*



Chapter 3.4, 3.6

# **Spot Quiz (ICON)**