

CSCI 466: Networks

Reliable Data Transfer (RDT)

Reese Pearsall
Fall 2023

PA1 Due **tonight**

Quiz 3 on Friday

- DNS
- SMTP
- DTP
- P2P
- Transport Layer Intro
- Reliable Data Transfer

Announcements

Tuesday, September 26, 2023 at 3:00:02 PM

[Learn more](#)



Overview

Share

Outputs

Quiz Results

Streams

References

Search

Captions

Audio

Descriptions

Manage

Log



People and groups

0 added | 1 inherited from **My Folder**

Add people and groups

Viewer



Reese Pearsall

unified\reese.pearsall | reesepearsall@montana.edu

Creator



Who can access this video

Anyone at your org who has the link



Link



Embed



Facebook



Twitter



Start at 0:00



https://m

Who can access this video



Restricted

Only specific people and groups



Your Organization (unlisted)

Anyone at your org who has the link



Public (unlisted)

Anyone who has the link

- Quiz 3 (
- DNS
 - SMTF
 - DTP
 - P2P
 - Trans
 - Reliak

OSI Model

Application Layer

Presentation Layer *

Session Layer *

Transport Layer

Network Layer

Data Link Layer

Physical Layer

Application Layer

Messages from Network Applications



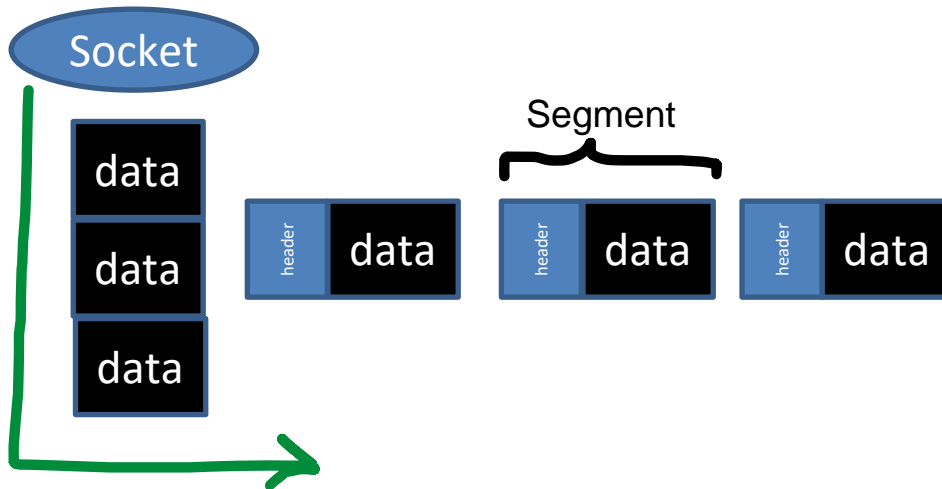
Physical Layer

Bits being transmitted over some medium

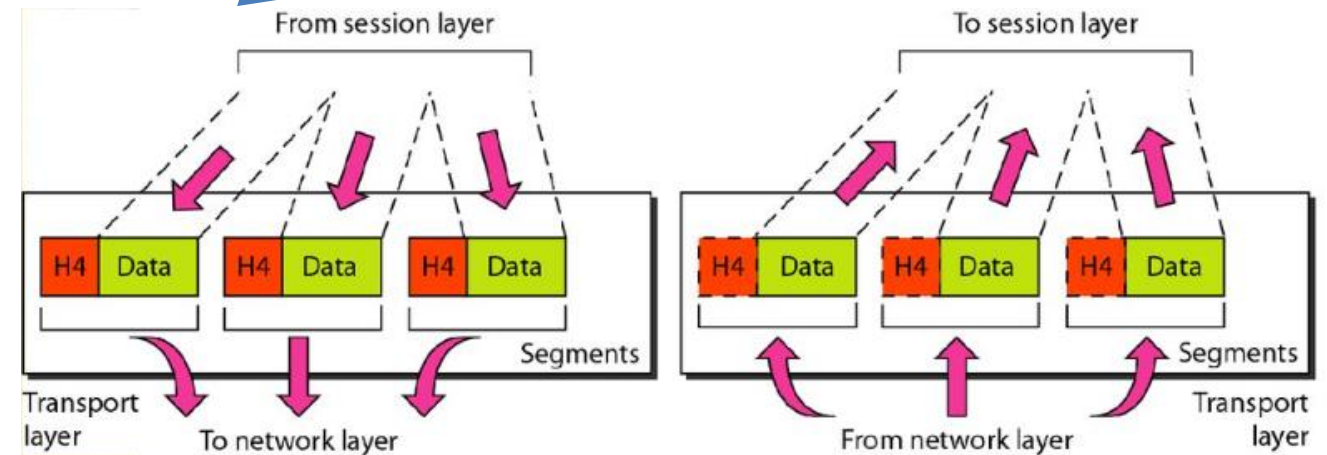
**In the textbook, they condense it to a 5-layer model, but 7 layers is what is most used*

Transport Layer

Multiplexing is the process of gathering chunks from sockets, encapsulating chunks with header information, and passing the segment into the network layer



Demultiplexing is the receiving segments from the transport layer and delivering the segment to the correct socket.

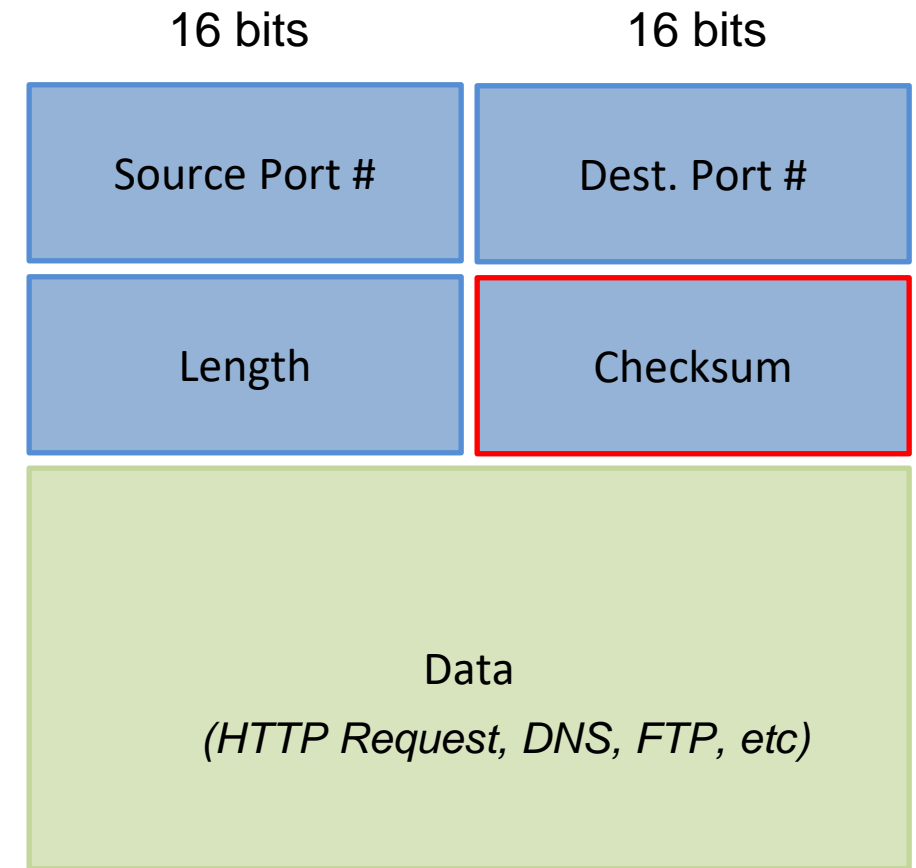


Transport Layer

Service	TCP/UDP Port Number
HTTP	80
DNS	53
HTTPS	443
FTP	20 & 21
SMTP	25
POP3	110
SSH	22
Runescape	43594, 43595

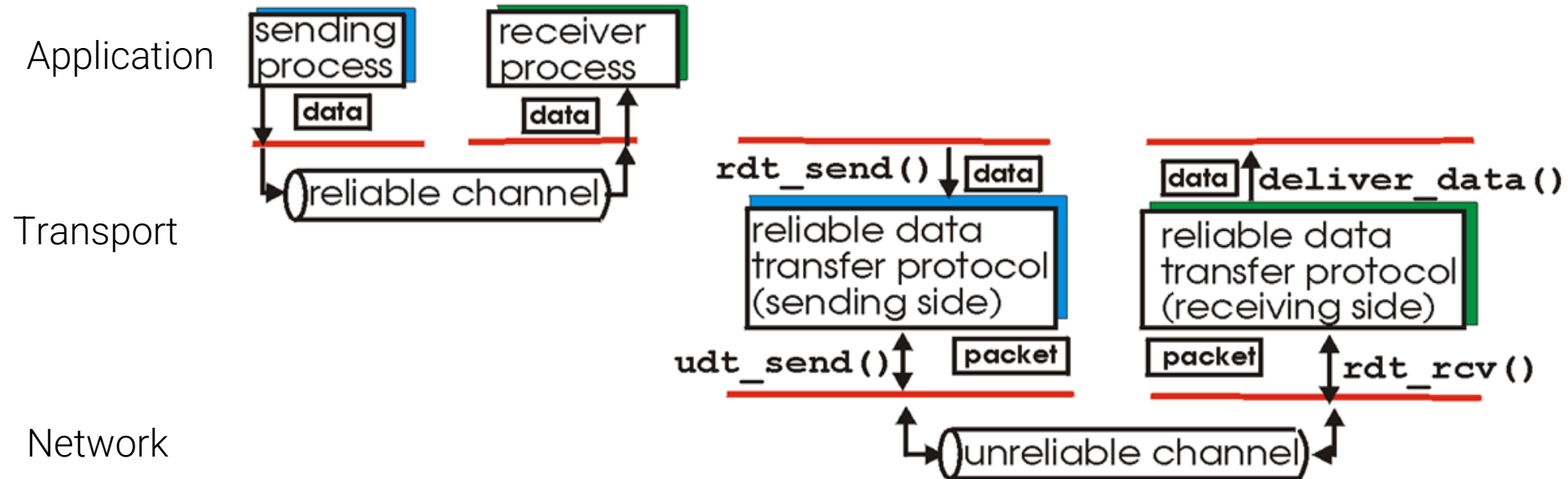
Transport Layer

Transport Layer provides a **checksum** that is used to determine whether bits within a segment have been altered



Transport Layer

Reliable Data Transfer



Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

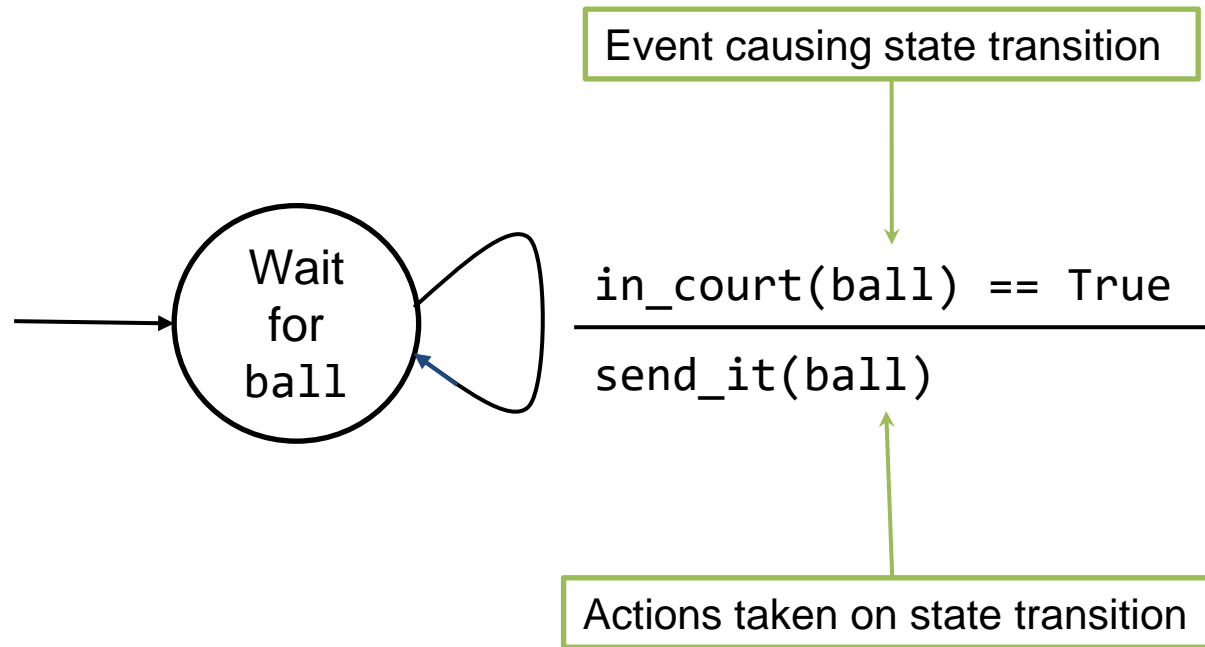
What are some ways in which the network channel can be unreliable?

Things to consider:

- Corruption
- Loss of packets
- Duplicate delivery



Bruce Lee FSM



Transport Layer

Reliable Data Transfer 1.0

RDT 1.0

Assumptions:

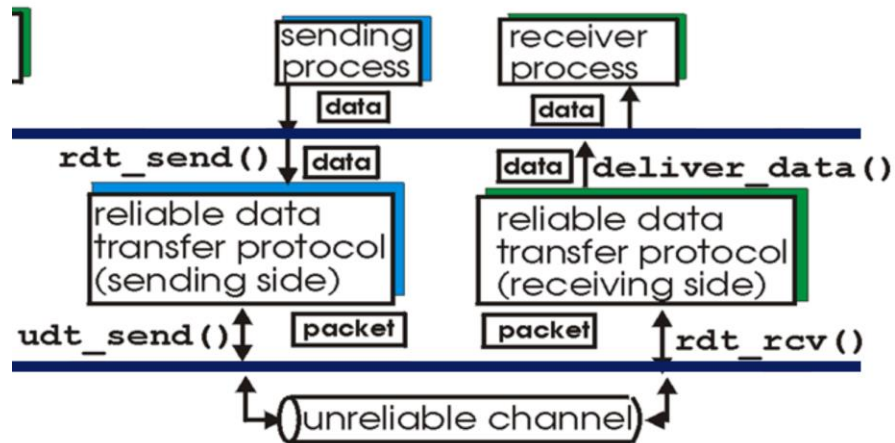
- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering



RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering



Transport Layer

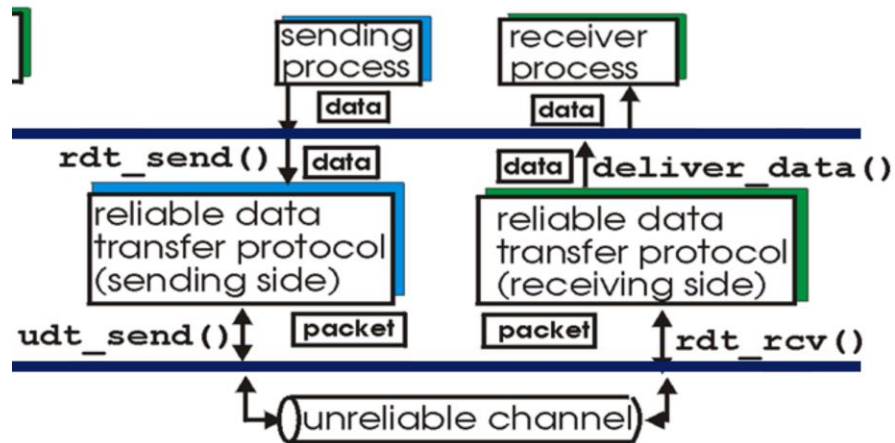
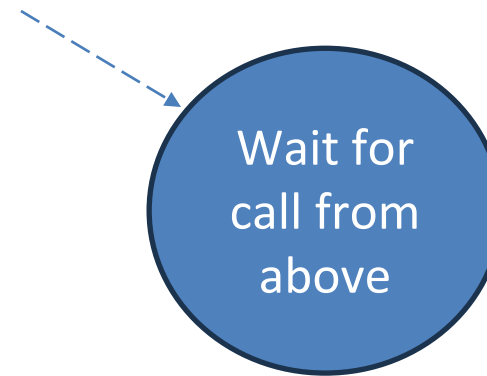
Reliable Data Transfer 1.0

RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

Sender



Transport Layer

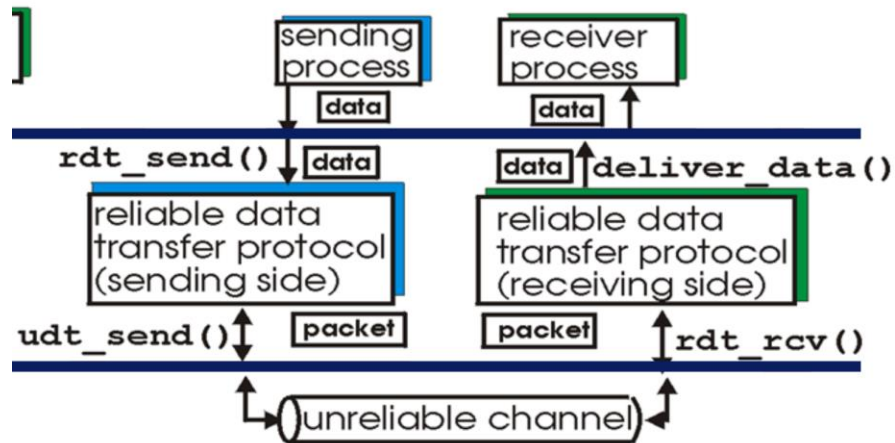
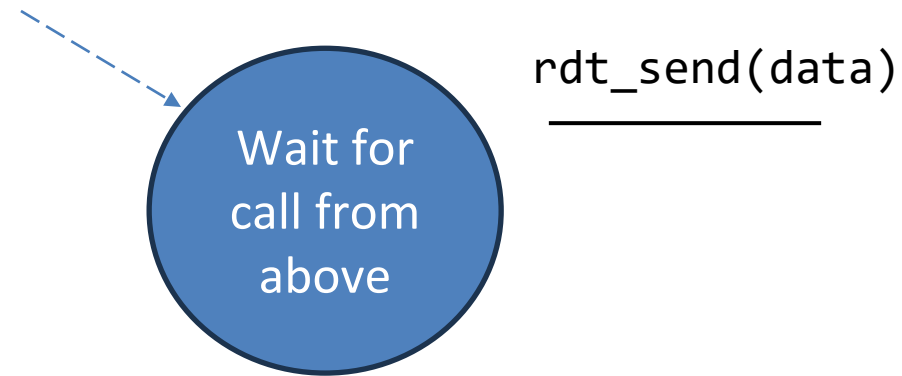
Reliable Data Transfer 1.0

RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

Sender



Transport Layer

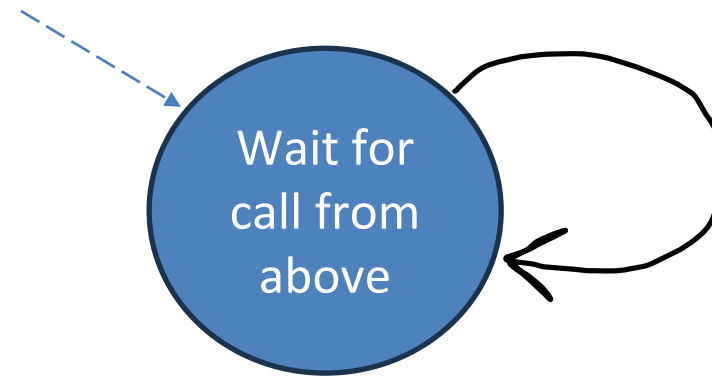
Reliable Data Transfer 1.0

RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

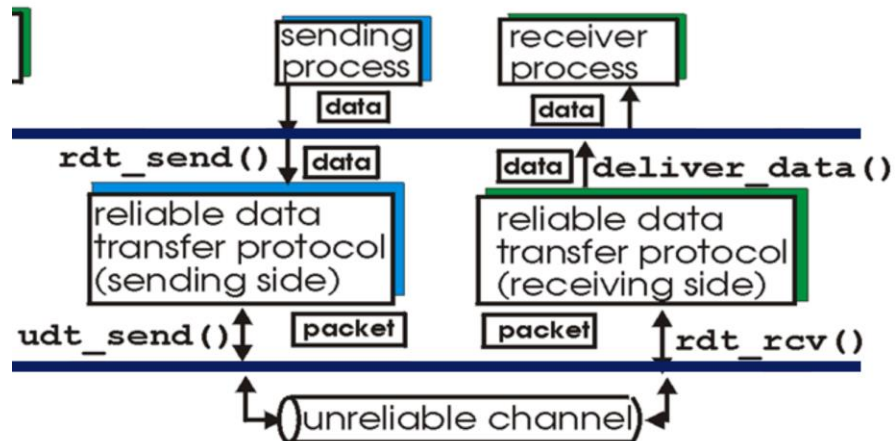
Sender



rdt_send(data)

packet = make_pkt(data)

udt_send(packet)

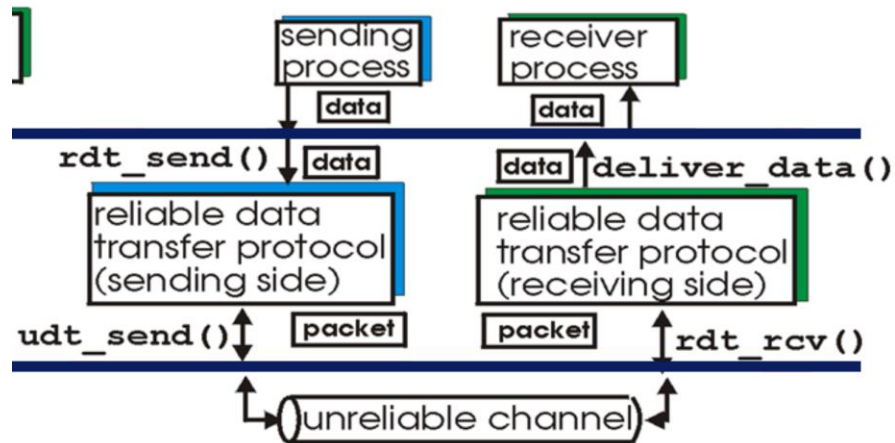


Transport Layer

RDT 1.0

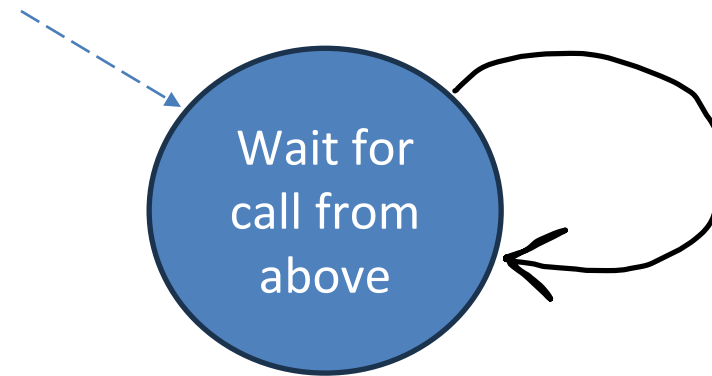
Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering



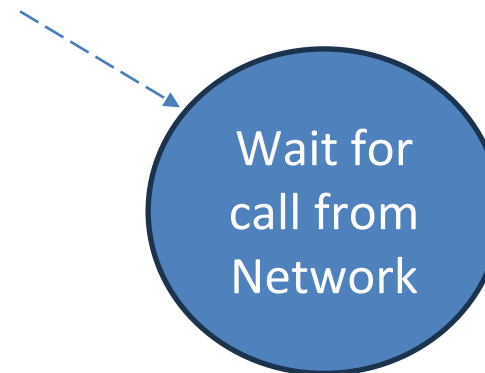
Reliable Data Transfer 1.0

Sender



```
rdt_send(data)
    packet = make_pkt(data)
    udt_send(packet)
```

Receiver

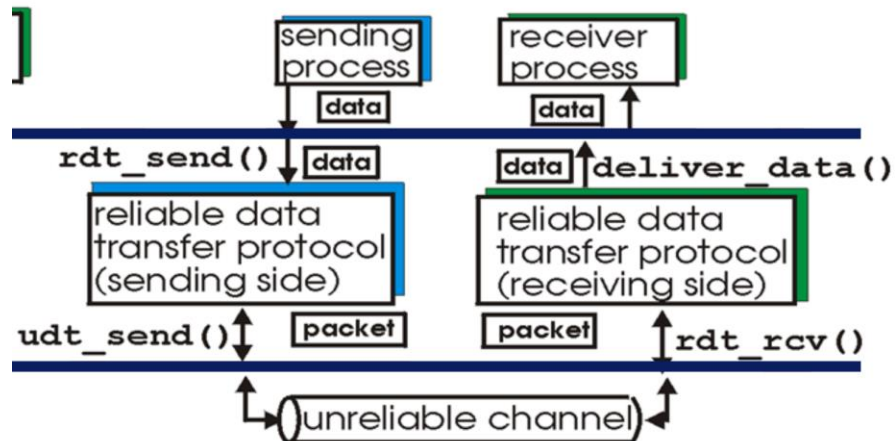


Transport Layer

RDT 1.0

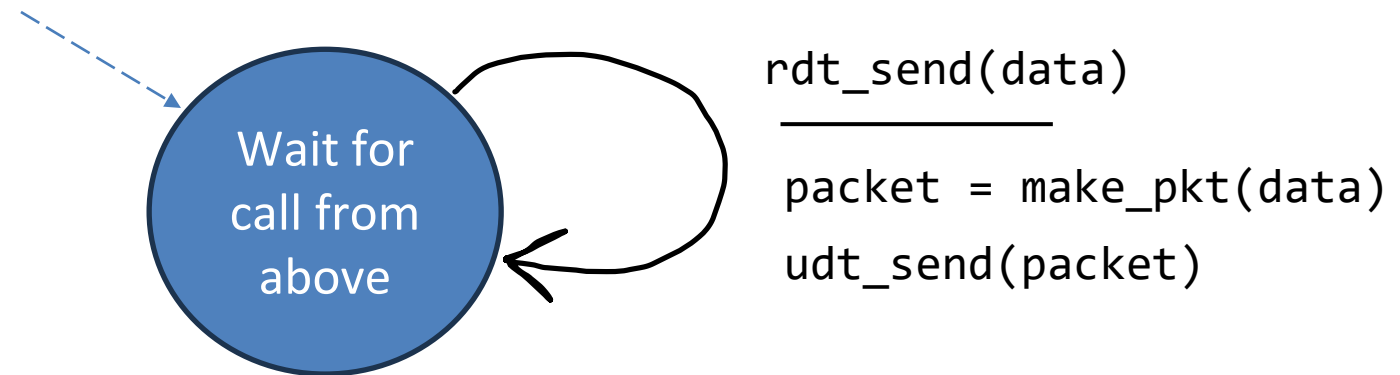
Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

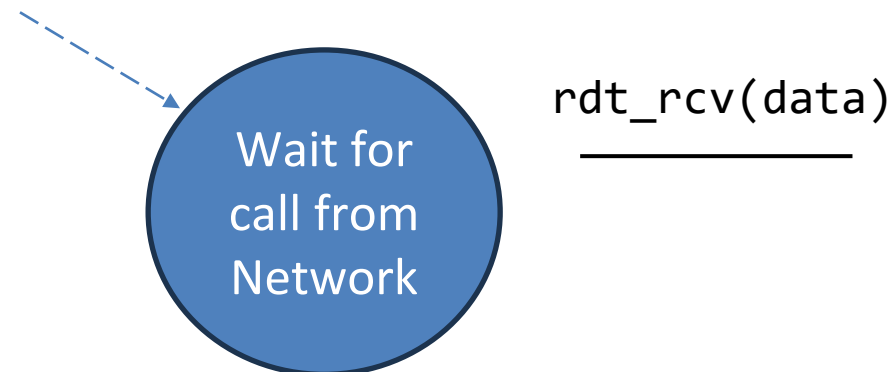


Reliable Data Transfer 1.0

Sender



Receiver



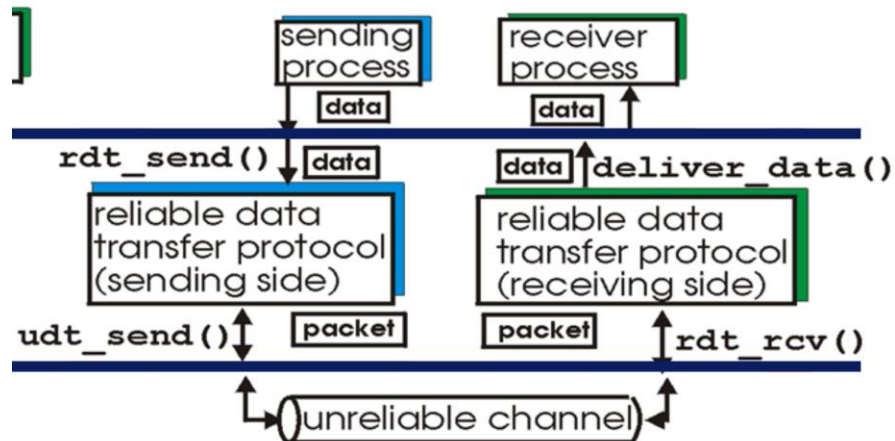
Transport Layer

Reliable Data Transfer 1.0

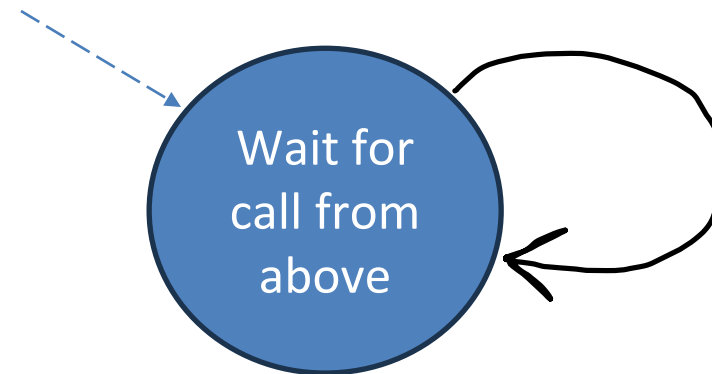
RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

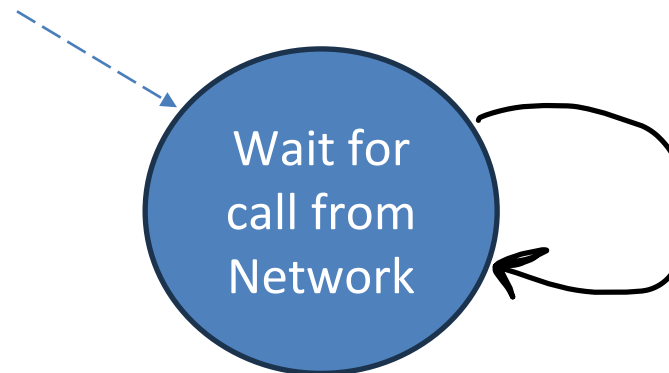


Sender



```
rdt_send(data)
packet = make_pkt(data)
udt_send(packet)
```

Receiver



```
rdt_rcv(data)
data = extract(packet)
deliver_data(data)
```

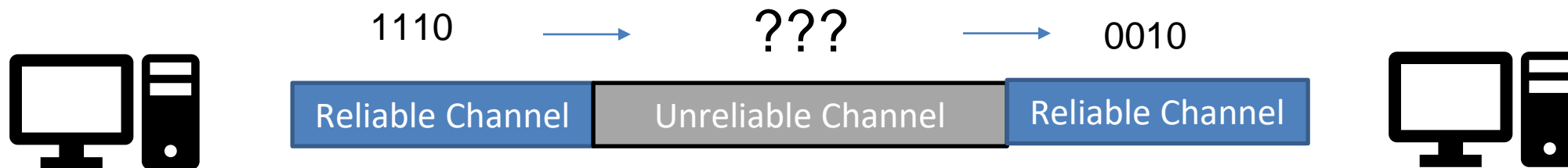
Transport Layer

Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?



Transport Layer

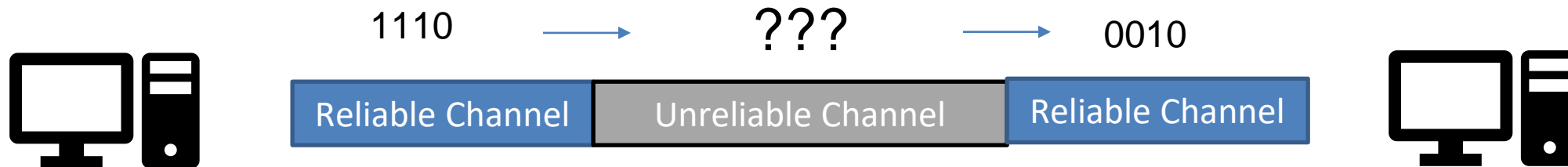
Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?

-Checksum



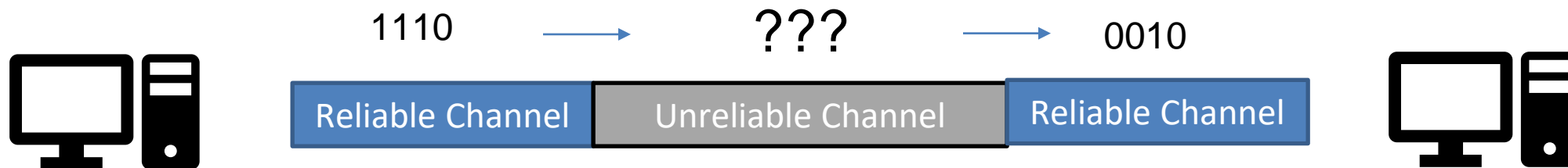
RDT 2.0

Potential for bit errors

How can we detect errors?

-Checksum

What is a good way to handle and prevent errors?



RDT 2.0

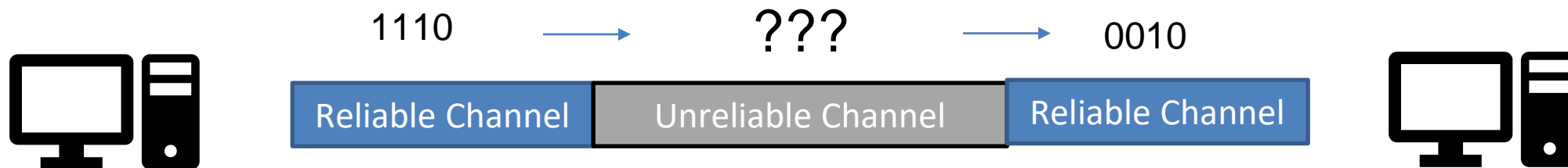
Potential for bit errors

How can we detect errors?

- Checksum

What is a good way to handle and prevent errors?

- Acknowledged packet, Ask for retransmit if needed



Transport Layer

Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?

-Checksum

What is a good way to handle and prevent errors?
- Acknowledged packet, Ask for retransmit if needed

Stop-and-wait: sender sends one packet, then waits for receiver response

Sender

`rdt_send(data)`

`sndpkt = make_pkt(data, checksum)`

`udt_send(sndpkt)`

Wait for call
from appl



Transport Layer

Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?

-Checksum

What is a good way to handle and prevent errors?
- Acknowledged packet, Ask for retransmit if needed

Stop-and-wait: sender sends one packet, then waits for receiver response

Sender

`rdt_send(data)`

`sndpkt = make_pkt(data, checksum)`

`udt_send(sndpkt)`

Wait for call
from appl



Wait for ACK
or NAK

Sender

rdt_send(data)

sndpkt = make_pkt(data, checksum)

udt_send(sndpkt)



Receiver

Sender

rdt_send(data)

sndpkt = make_pkt(data, checksum)

udt_send(sndpkt)

Wait for call
from appl

Wait for ACK
or NAK

Receiver

Wait for call
from below

Sender

rdt_send(data)

sndpkt = make_pkt(data, checksum)

udt_send(sndpkt)

Wait for call
from appl

Wait for ACK
or NAK

Receiver

Wait for call
from below

rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)

udt_send(NAK)

Sender

rdt_send(data)

sndpkt = make_pkt(data, checksum)

udt_send(sndpkt)



Receiver

Wait for call from below

rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)

udt_send(NAK)

rdt_rcv(rcvpkt) &&
!corrupt(rcvpkt)
data = extract(packet)
deliver_data(data)
udt_send(ACK)

Transport Layer

Reliable Data Transfer 2.0

Sender

rdt_send(data)

sndpkt = make_pkt(data, checksum)

udt_send(sndpkt)

Wait for call
from appl

Wait for ACK
or NAK

Receiver

rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)

udt_send(NAK)

rdt_rcv(rcvpkt) &&
!corrupt(rcvpkt)

data = extract(packet)
deliver_data(data)
udt_send(ACK)

Wait for call
from below

Transport Layer

Reliable Data Transfer 2.0

Sender

rdt_send(data)

sndpkt = make_pkt(data, checksum)

udt_send(sndpkt)

rdt_rcv(rcvpkt) &&

isNAK(rcvpkt)

udt_send(sndpkt)

Wait for call
from appl

Wait for ACK
or NAK

Receiver

rdt_rcv(rcvpkt) &&

corrupt(rcvpkt)

udt_send(NAK)

rdt_rcv(rcvpkt) &&

!corrupt(rcvpkt)

data = extract(packet)

deliver_data(data)

udt_send(ACK)

Wait for call
from below

Sender

rdt_send(data)

sndpkt = make_pkt(data, checksum)

udt_send(sndpkt)

rdt_rcv(rcvpkt) &&

isNAK(rcvpkt)

udt_send(sndpkt)

Wait for call
from appl

Wait for ACK
or NAK

rdt_rcv(rcvpkt) && isACK(rcvpkt)

Λ

Receiver

rdt_rcv(rcvpkt) &&

corrupt(rcvpkt)

udt_send(NAK)

rdt_rcv(rcvpkt) &&

!corrupt(rcvpkt)

data = extract(packet)

deliver_data(data)

udt_send(ACK)

Wait for call
from below

Λ = do no action, follow arrow to next state

Sender

What happens if ACK/NAK
Corrupted?

→ Duplicate delivery, or no
retransmission

Solution?

→ Retransmit if CORRUPT packet
received

How to deal we duplicate Packets?

→ ???

Wait for
from ap

Λ = do no a

Receiver

```
rdt_rcv(rcvpkt) &&  
corrupt(rcvpkt)
```

```
udt_send(NAK)
```

```
rdt_rcv(rcvpkt) &&  
!corrupt(rcvpkt)
```

```
data = extract(packet)  
deliver_data(data)  
udt_send(ACK)
```

call
flow

Sender

What happens if ACK/NAK
Corrupted?

→ Duplicate delivery, or no
retransmission

Solution?

→ Retransmit if CORRUPT packet
received

How to deal we duplicate Packets?

→ **Sequence Numbers**

Wait for
from ap

Λ = do no a

Receiver

```
rdt_rcv(rcvpkt) &&  
corrupt(rcvpkt)
```

```
udt_send(NAK)
```

```
rdt_rcv(rcvpkt) &&  
!corrupt(rcvpkt)
```

```
data = extract(packet)  
deliver_data(data)  
udt_send(ACK)
```

call
flow

Transport Layer

Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

(We only need to use 0 or 1 for the sequence number)

Sender

Wait for
call 0 from
above

Transport Layer

Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

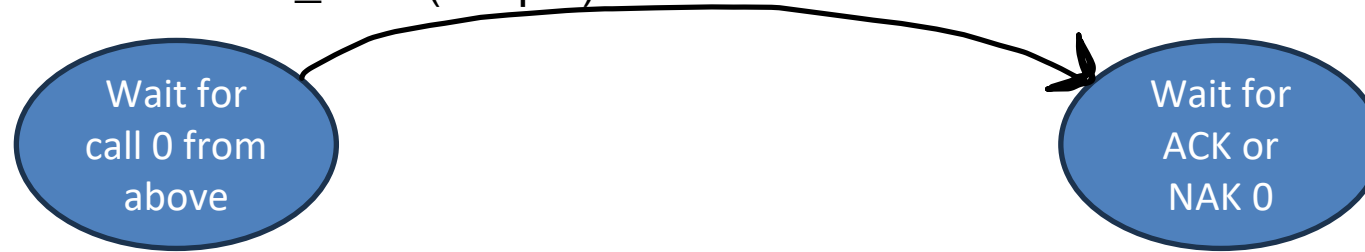
(We only need to use 0 or 1 for the sequence number)

Sender

rdt_send(data)

sndpkt = make_pkt(0, data, checksum)

udt_send(sndpkt)



Transport Layer

Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

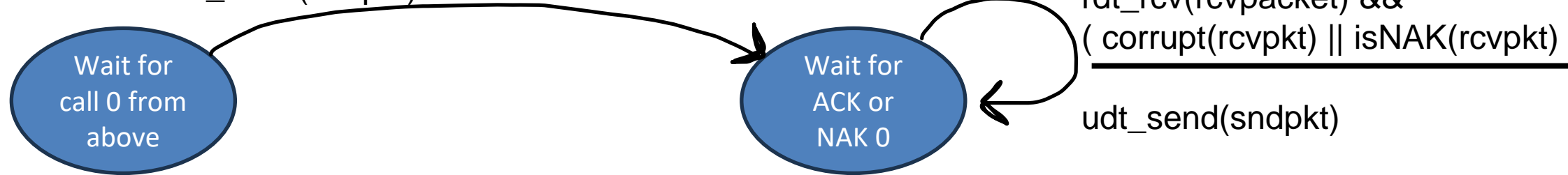
(We only need to use 0 or 1 for the sequence number)

Sender

rdt_send(data)

sndpkt = make_pkt(0, data, checksum)

udt_send(sndpkt)



Transport Layer

Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

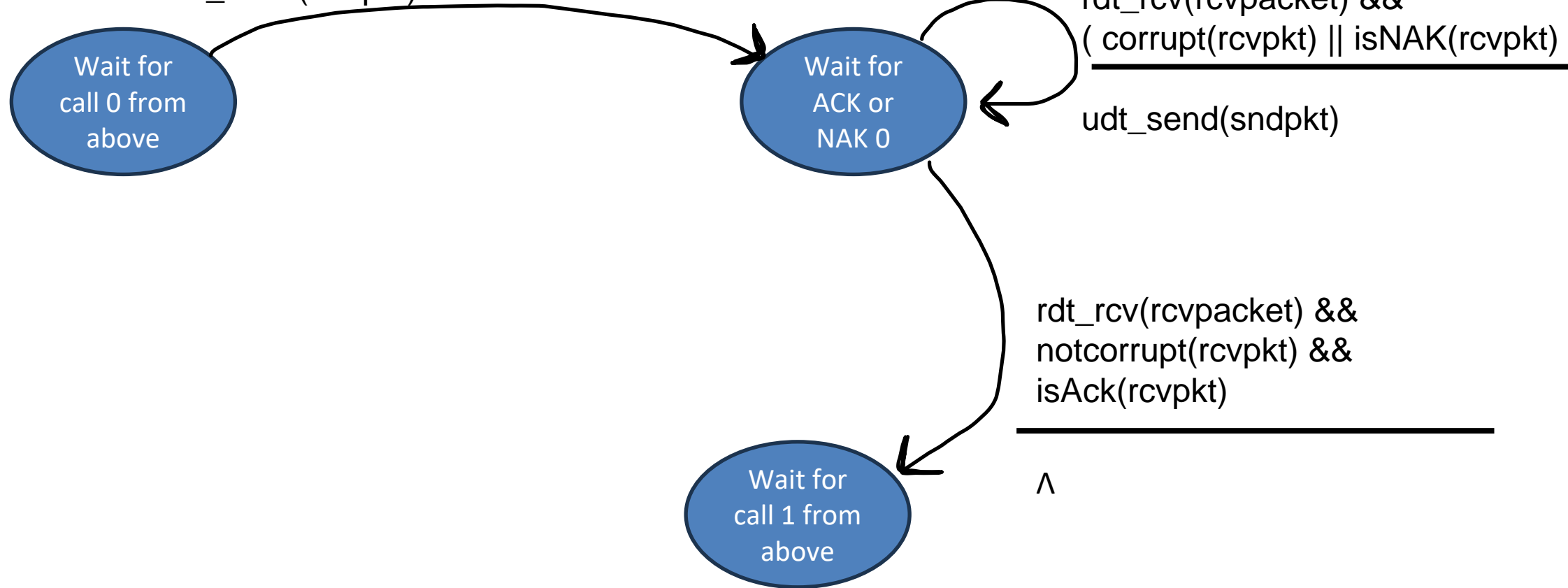
(We only need to use 0 or 1 for the sequence number)

Sender

rdt_send(data)

sndpkt = make_pkt(0, data, checksum)

udt_send(sndpkt)



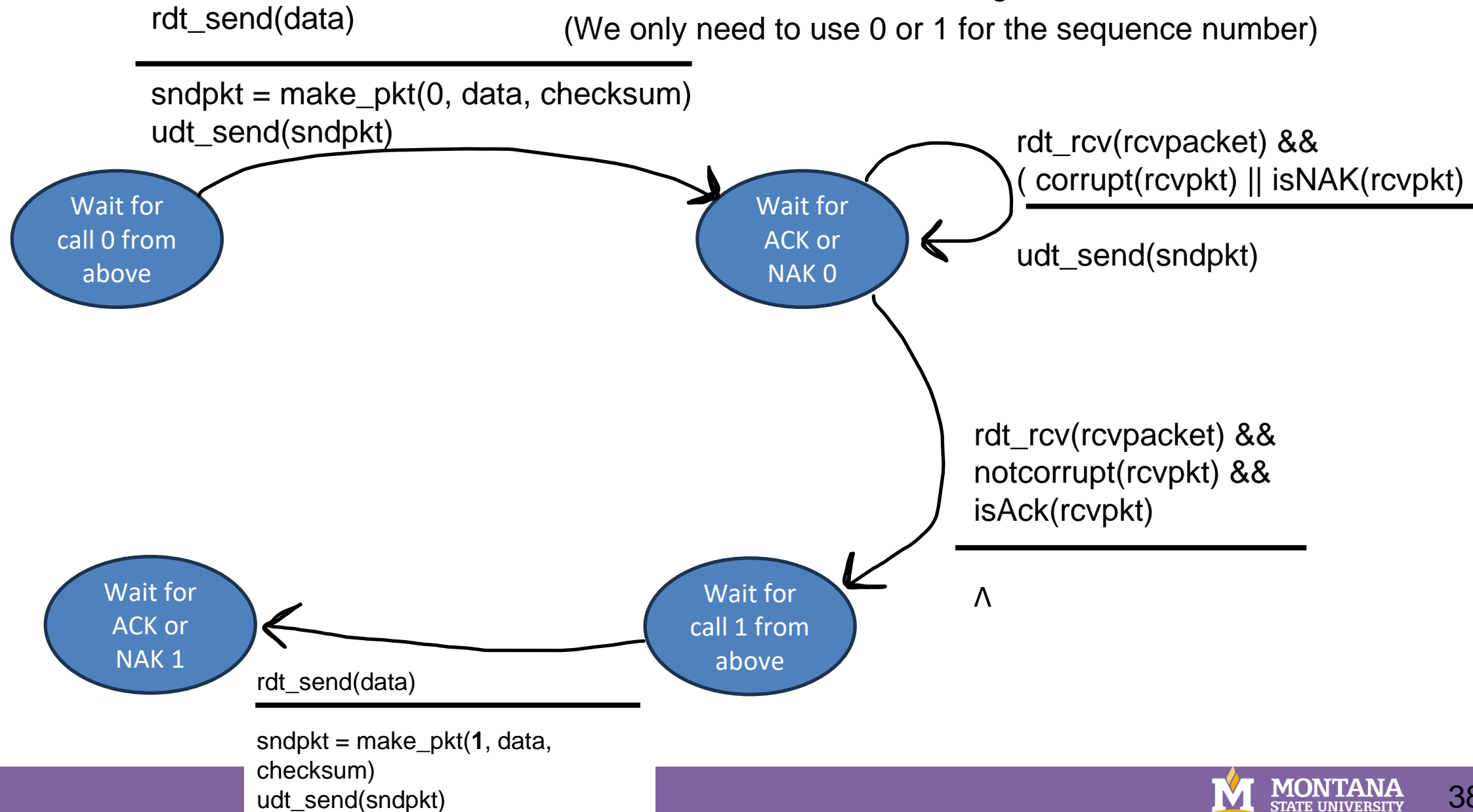
Transport Layer

Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

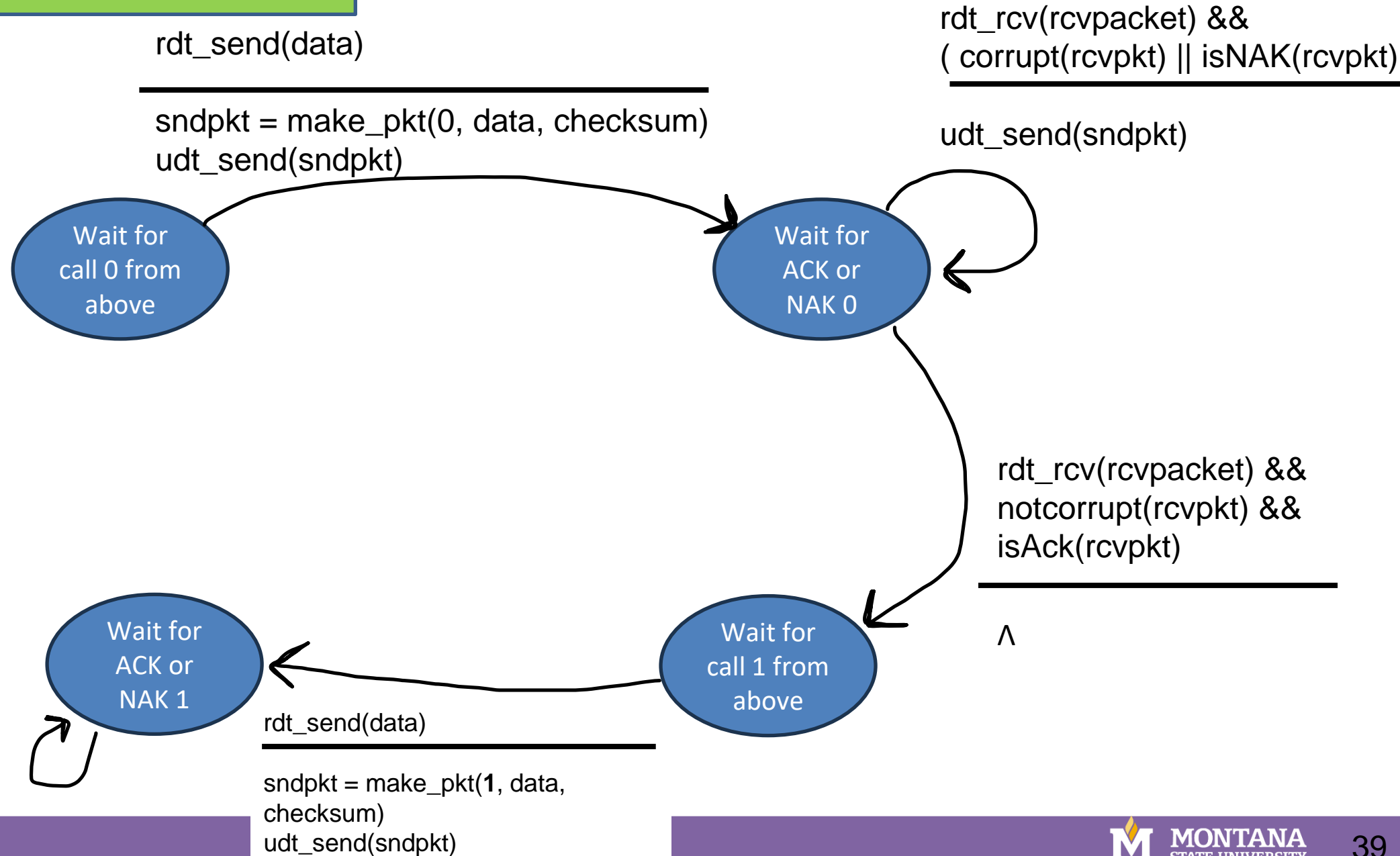
(We only need to use 0 or 1 for the sequence number)

Sender



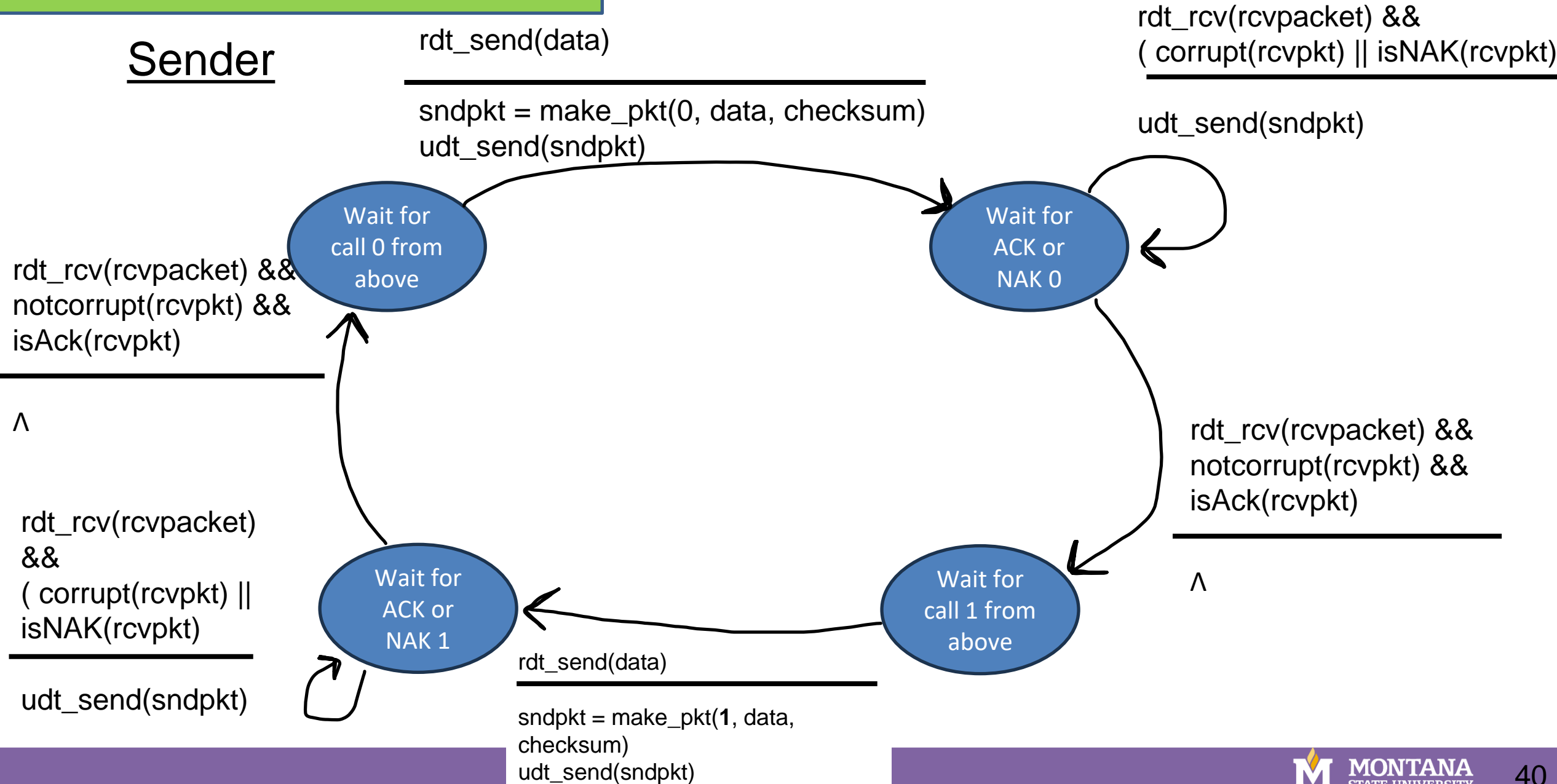
Transport Layer

Sender



Transport Layer

Sender



Transport Layer

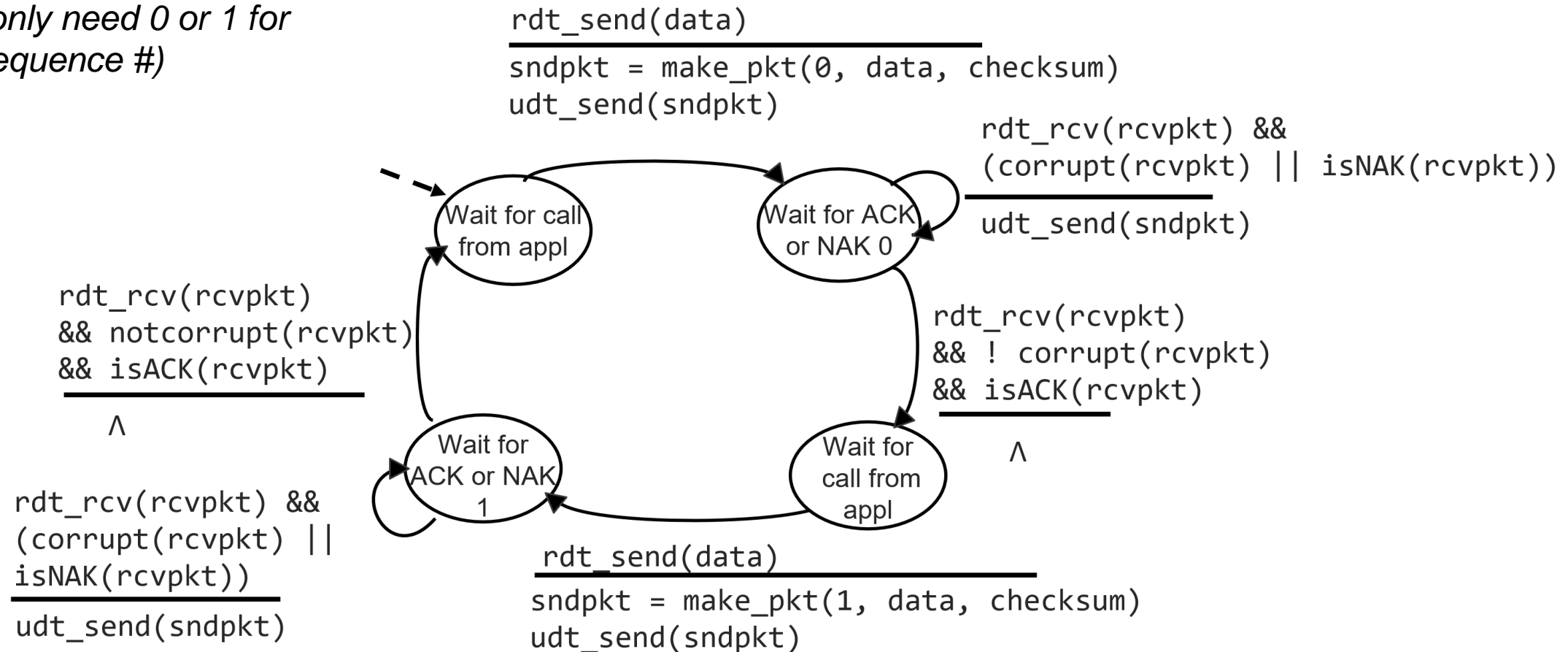
Reliable Data Transfer 2.1

RDT 2.1

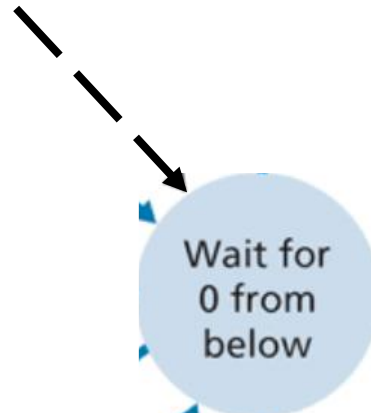
Potential for bit errors and garbled ACKs

Stop-and-wait: sender sends one packet, then waits for receiver response

(We only need 0 or 1 for the sequence #)

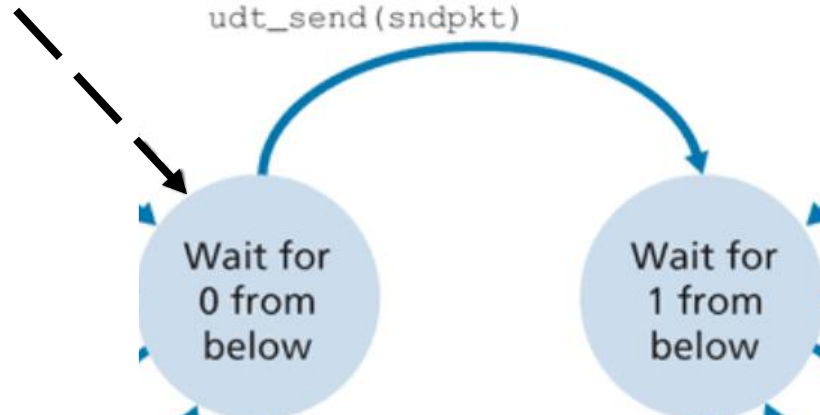


Receiver

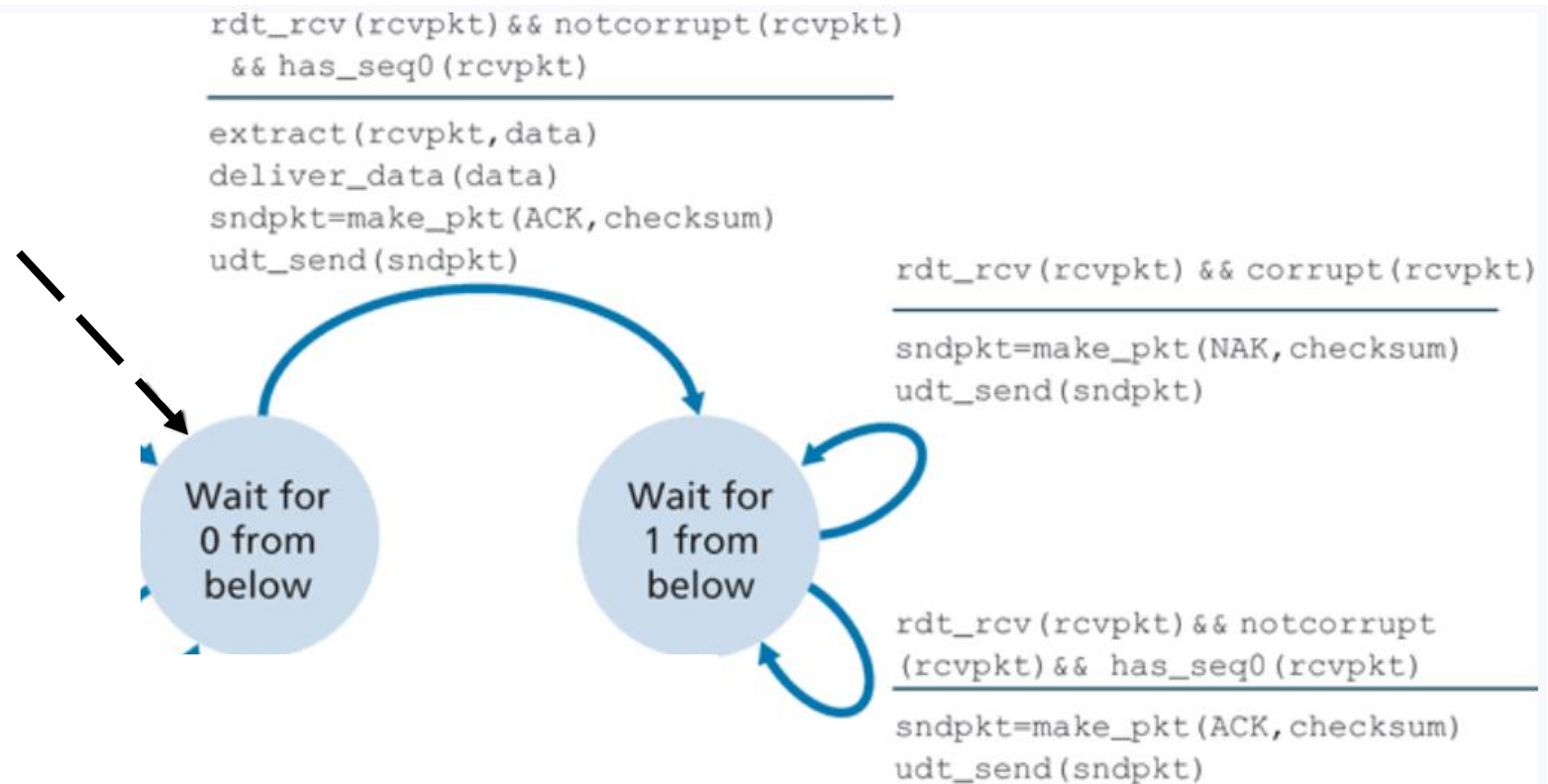


Receiver

```
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
  && has_seq0(rcvpkt)
  extract(rcvpkt, data)
  deliver_data(data)
  sndpkt=make_pkt(ACK, checksum)
  udt_send(sndpkt)
```



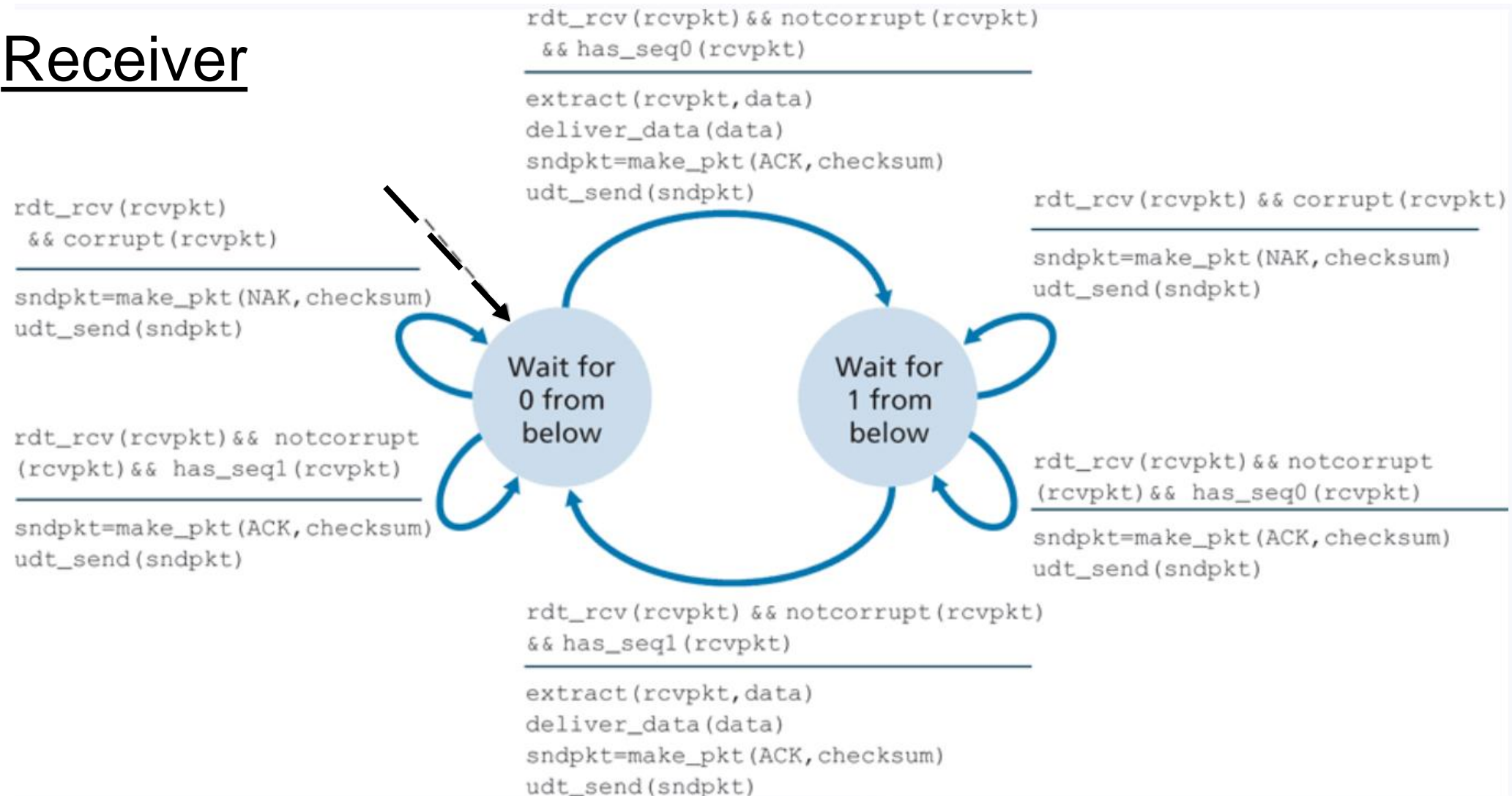
Receiver



Receiver



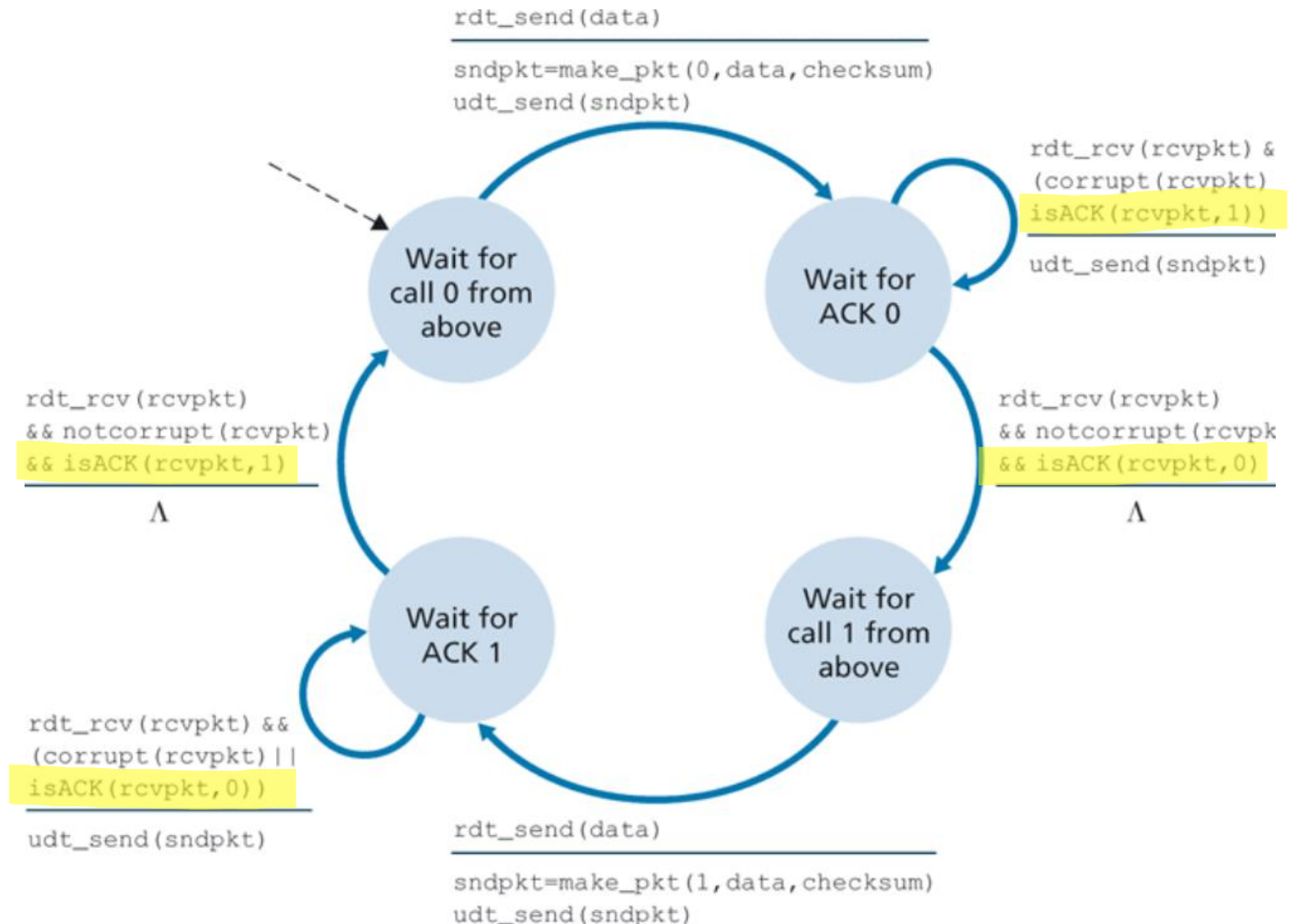
Receiver



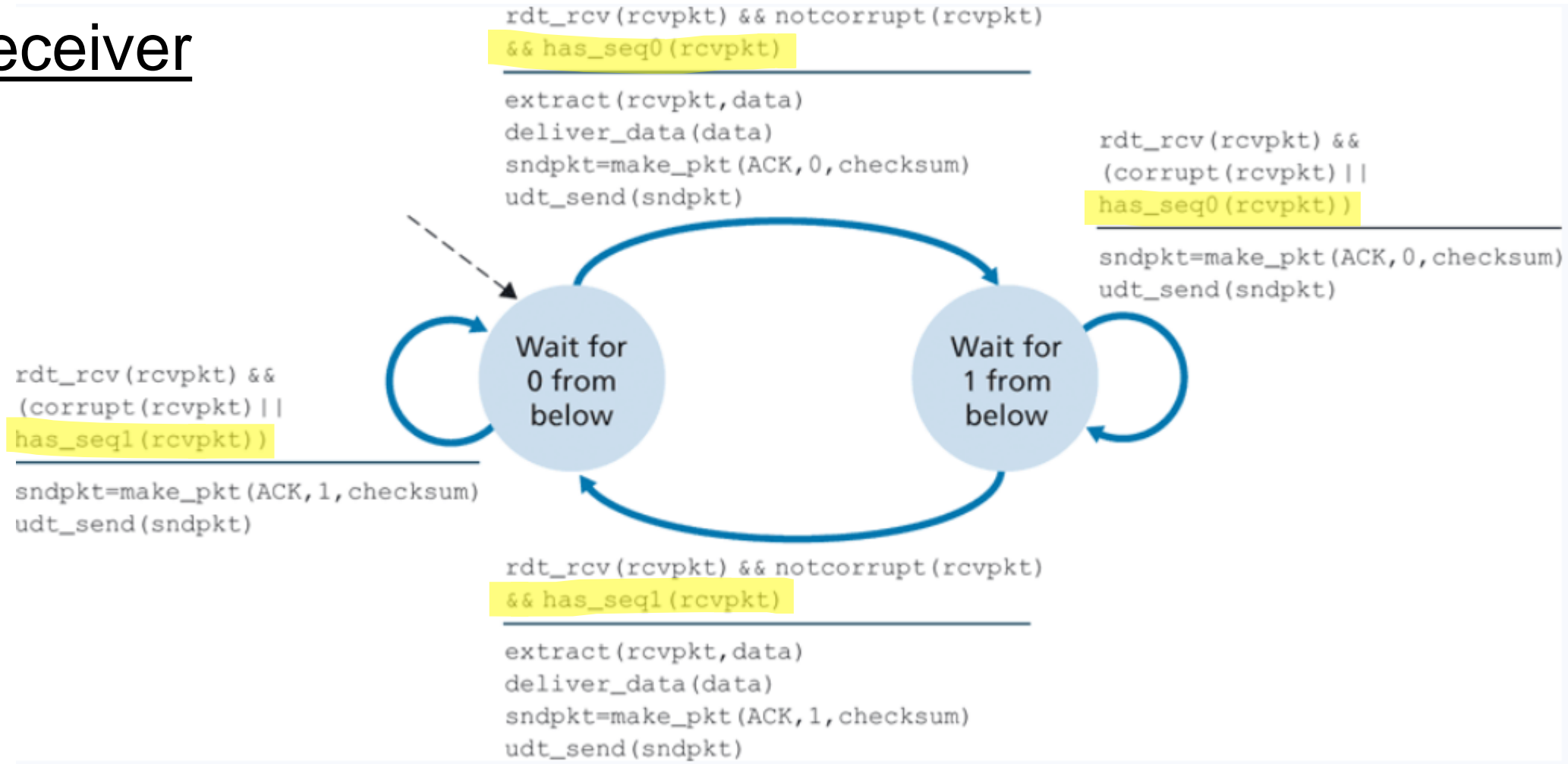
Sender

Instead of NAKs, we provide the sequence number of the packet for our ACK (ACK1, ACK0)

Duplicate ACK at sender results in same action as NAK: retransmit current pkt



Receiver



Transport Layer



A quick meatball break...

We will still need
checksums, Seq #'s,
ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

We will still need
checksums, Seq #'s,
ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

Solution?

We will still need
checksums, Seq #'s,
ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

A timer

Retransmit if ACK received in X amount of time

What if the ACK is just taking a really long time to arrive?

We will still need
checksums, Seq #'s,
ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

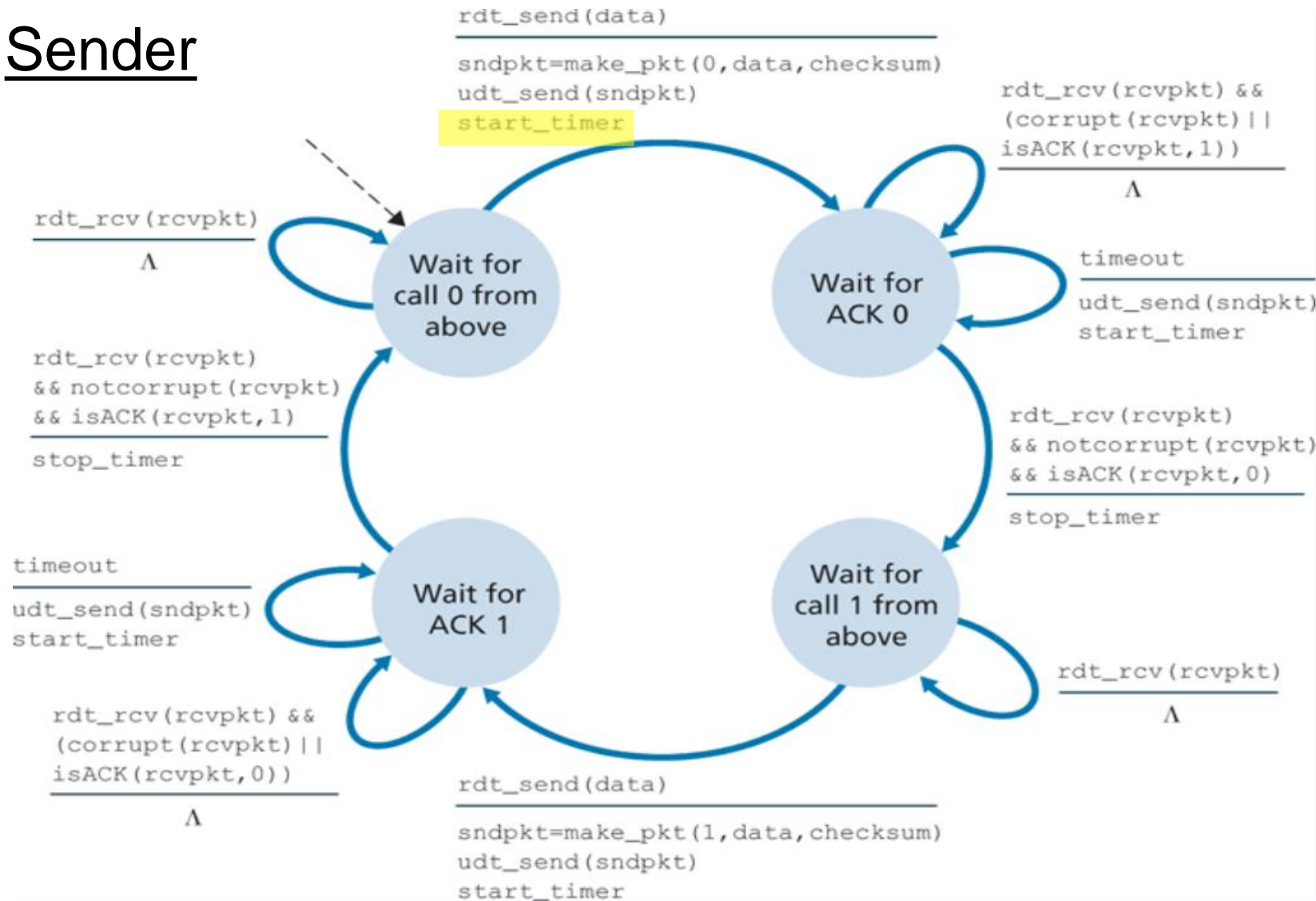
A timer

Retransmit if ACK received in X amount of time

What if the ACK is just taking a really long time to arrive? This will be duplicate data, but we have a Seq # to handle that

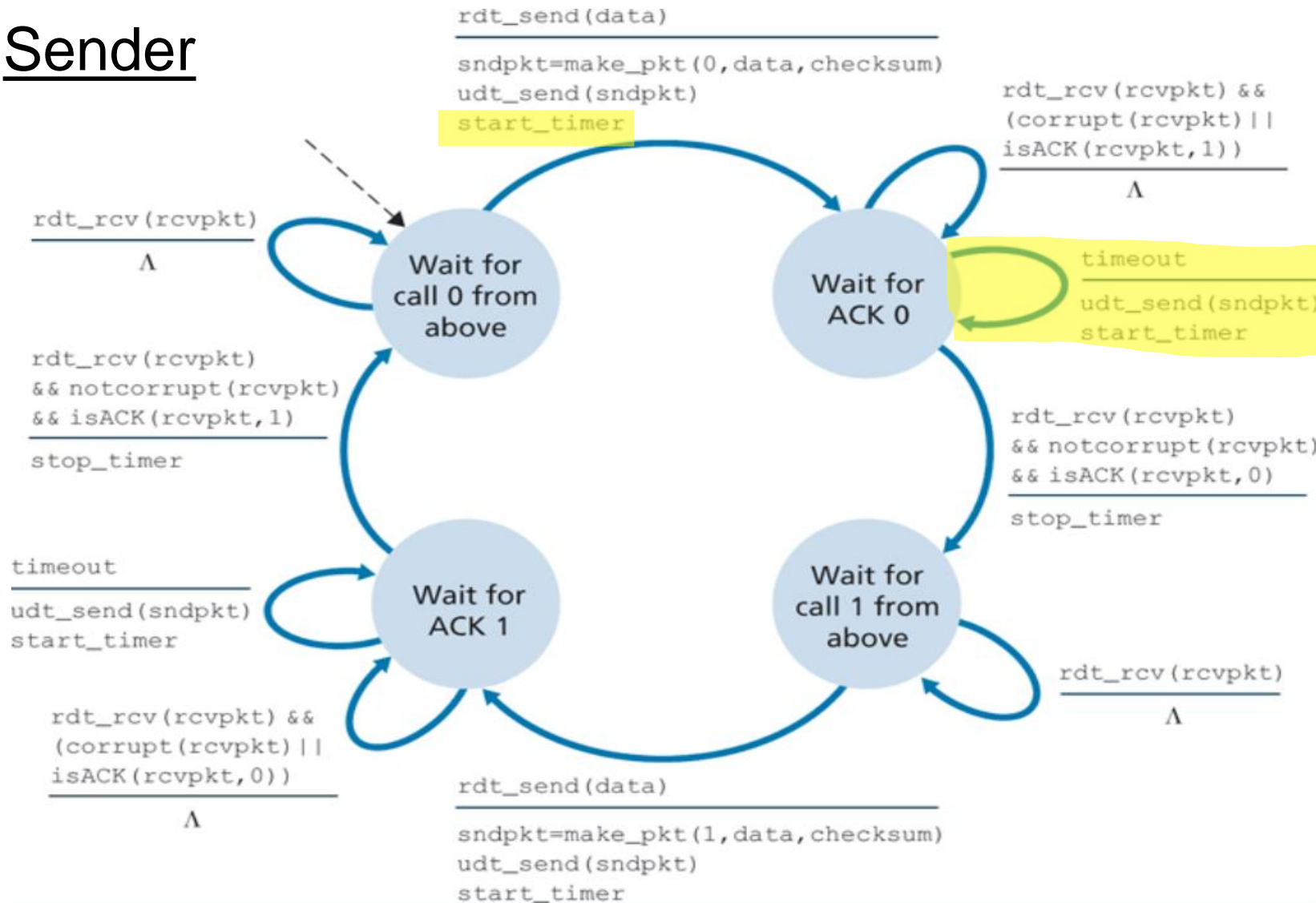
(Packets can get dropped or lost)

Sender



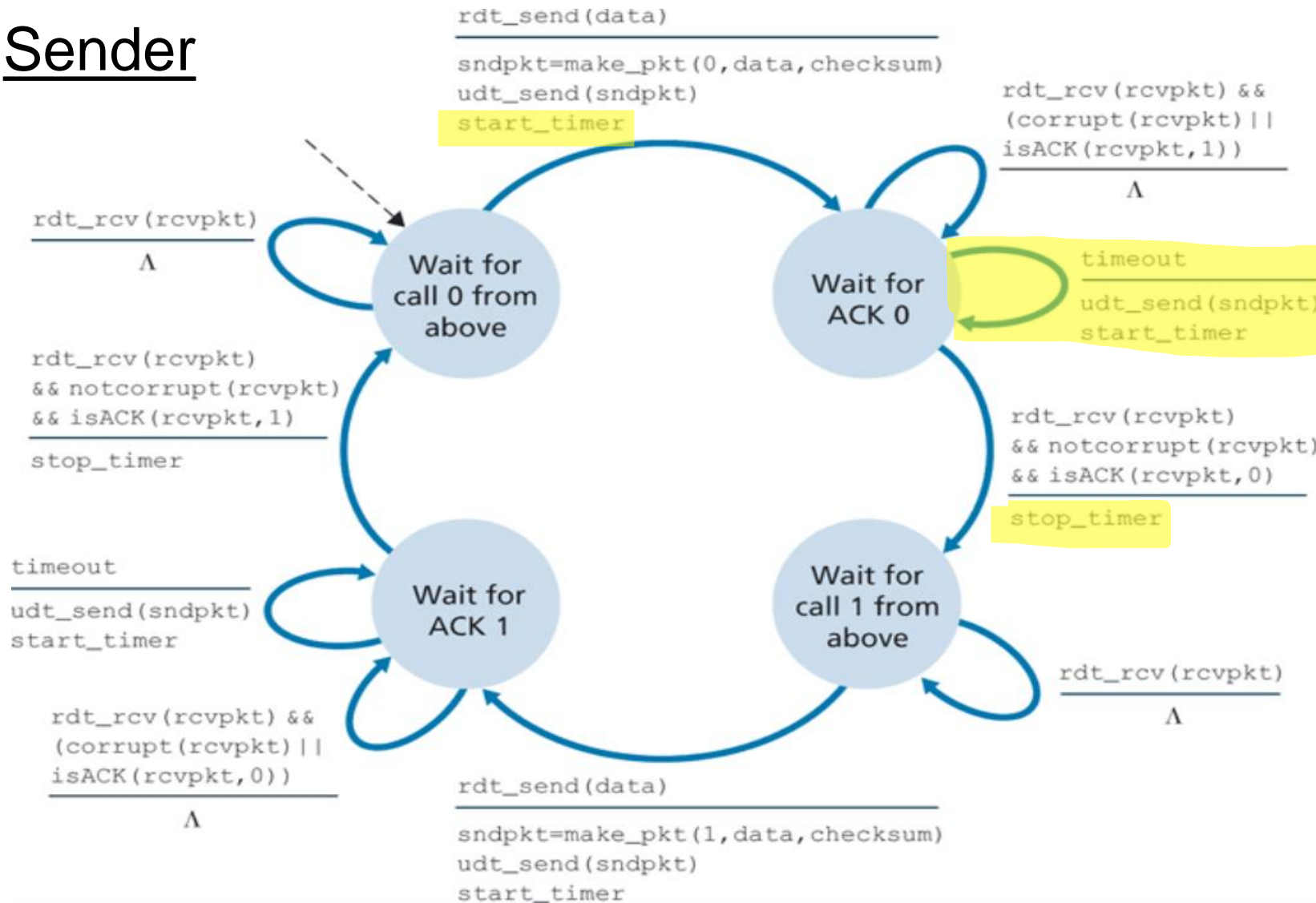
(Packets can get dropped or lost)

Sender



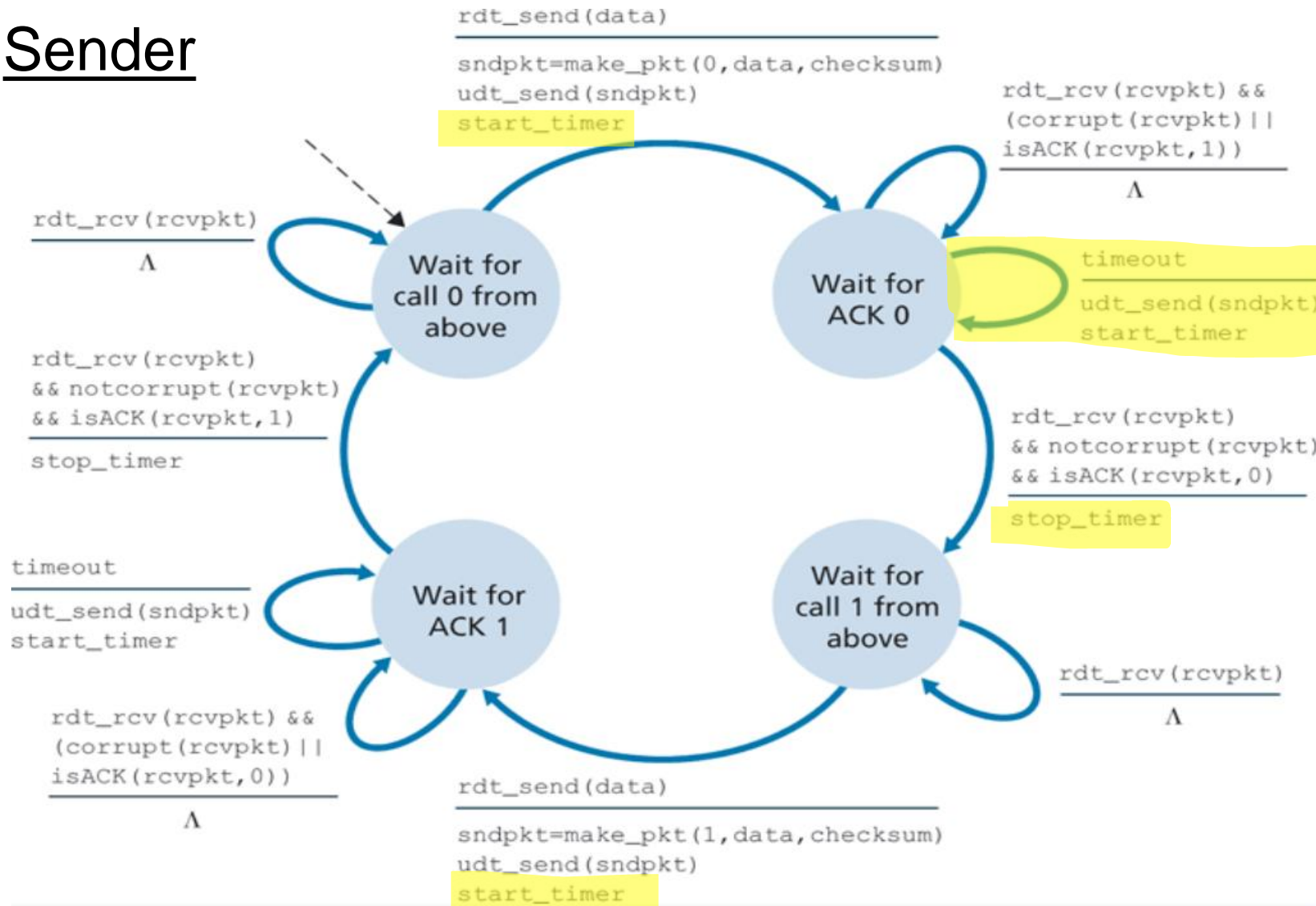
(Packets can get dropped or lost)

Sender



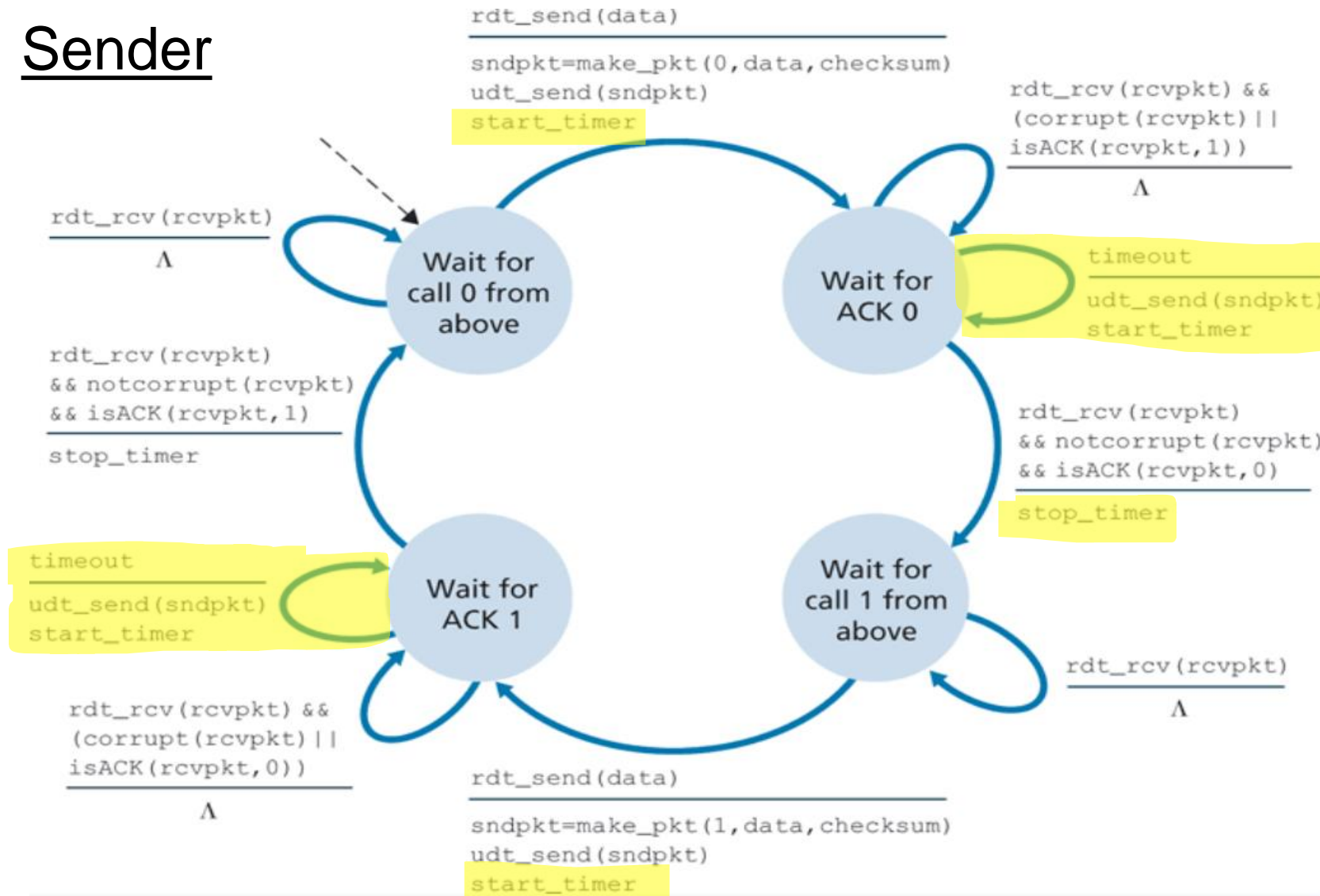
(Packets can get dropped or lost)

Sender



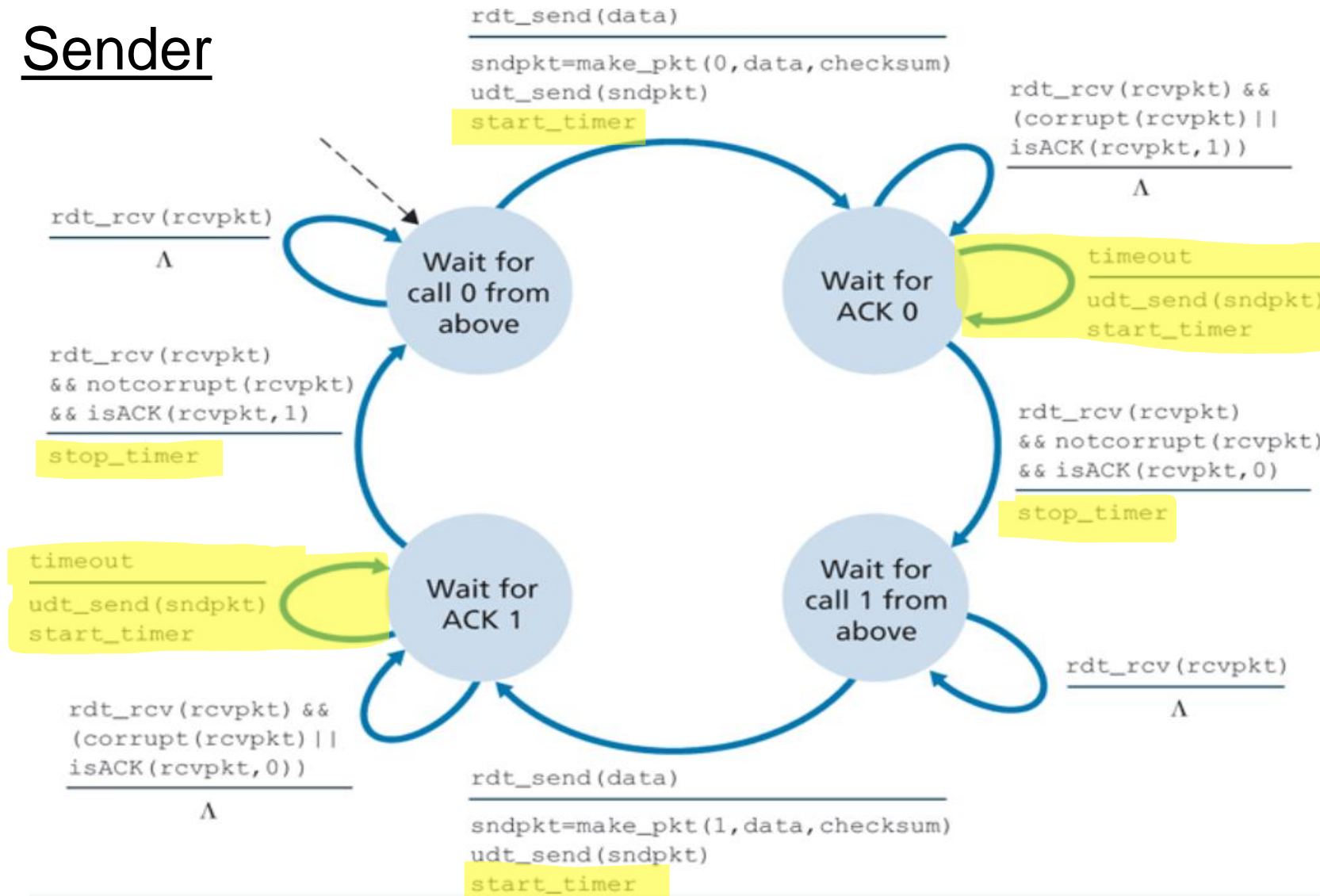
(Packets can get dropped or lost)

Sender



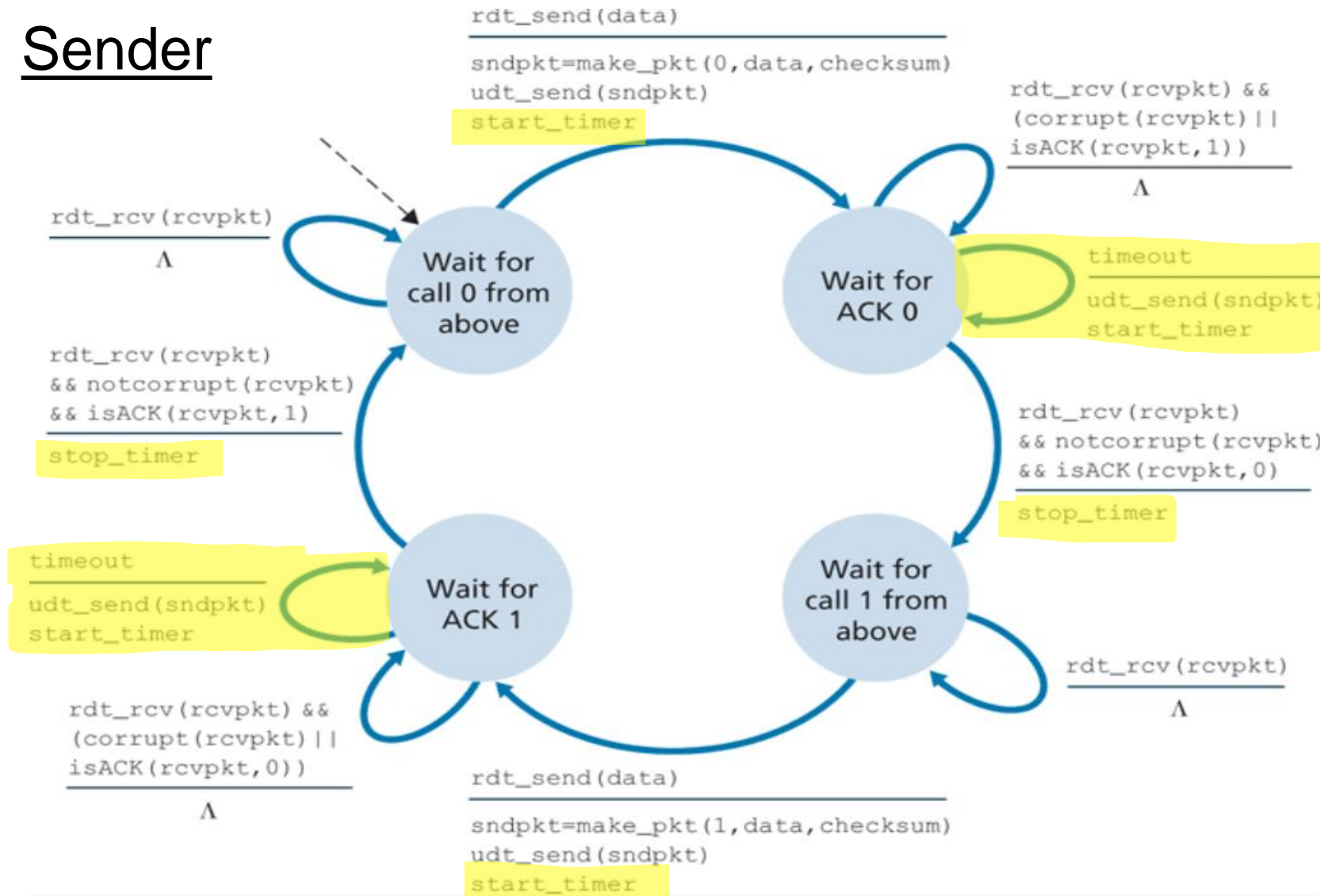
(Packets can get dropped or lost)

Sender



(Packets can get dropped or lost)

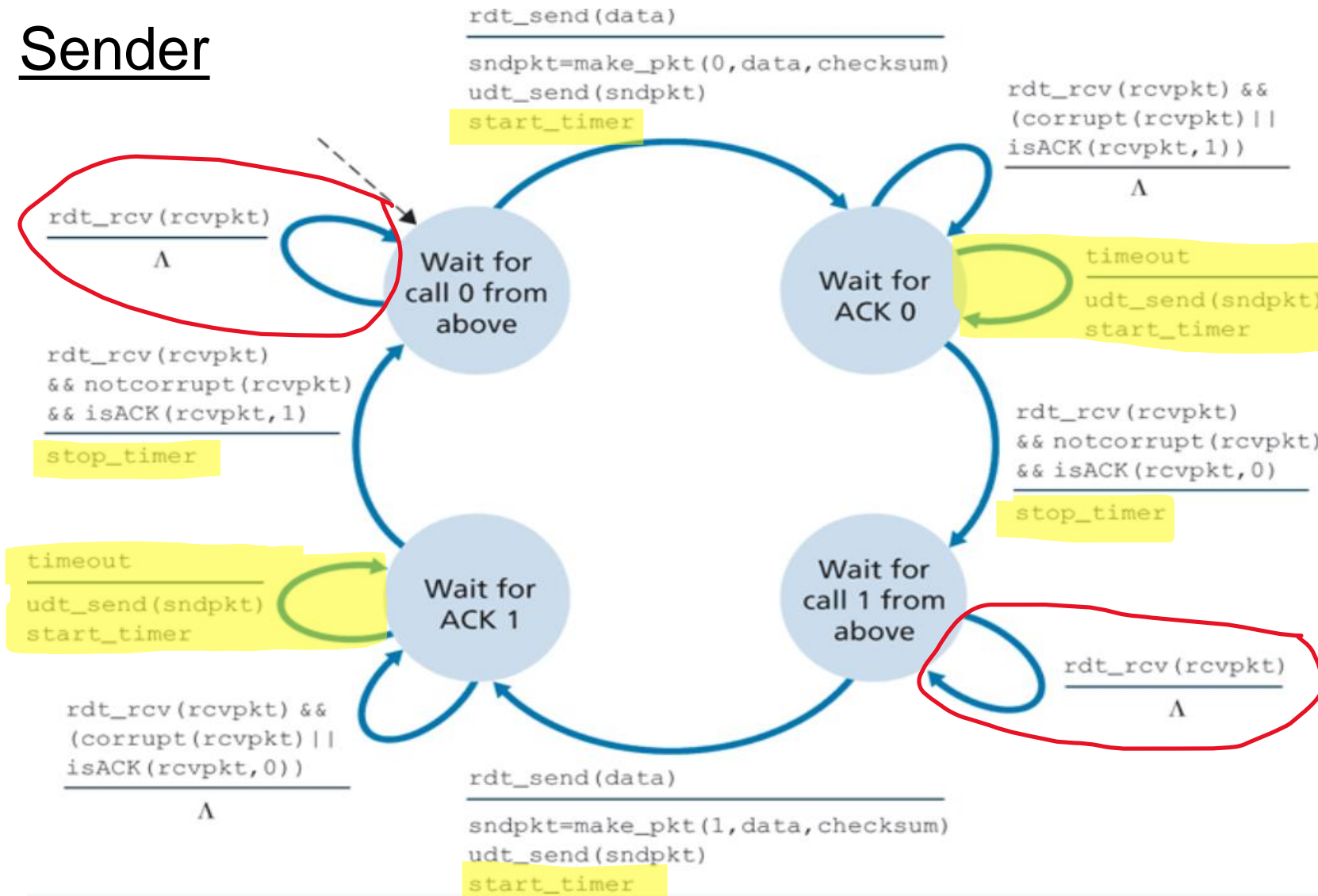
Sender



Receiver will be same as 2.2

(Packets can get dropped or lost)

Sender

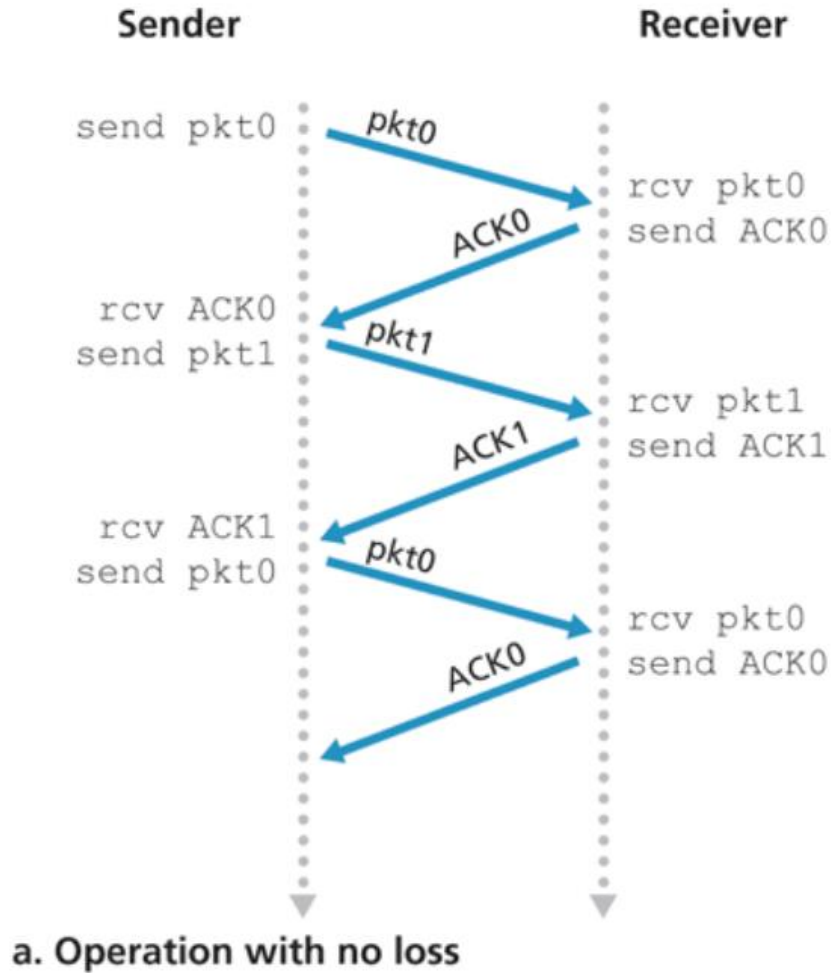


If we are waiting to send packet, but we receive a packet for some reason, this packet must be a duplicate packet, so do nothing

Transport Layer

Reliable Data Transfer 3.0

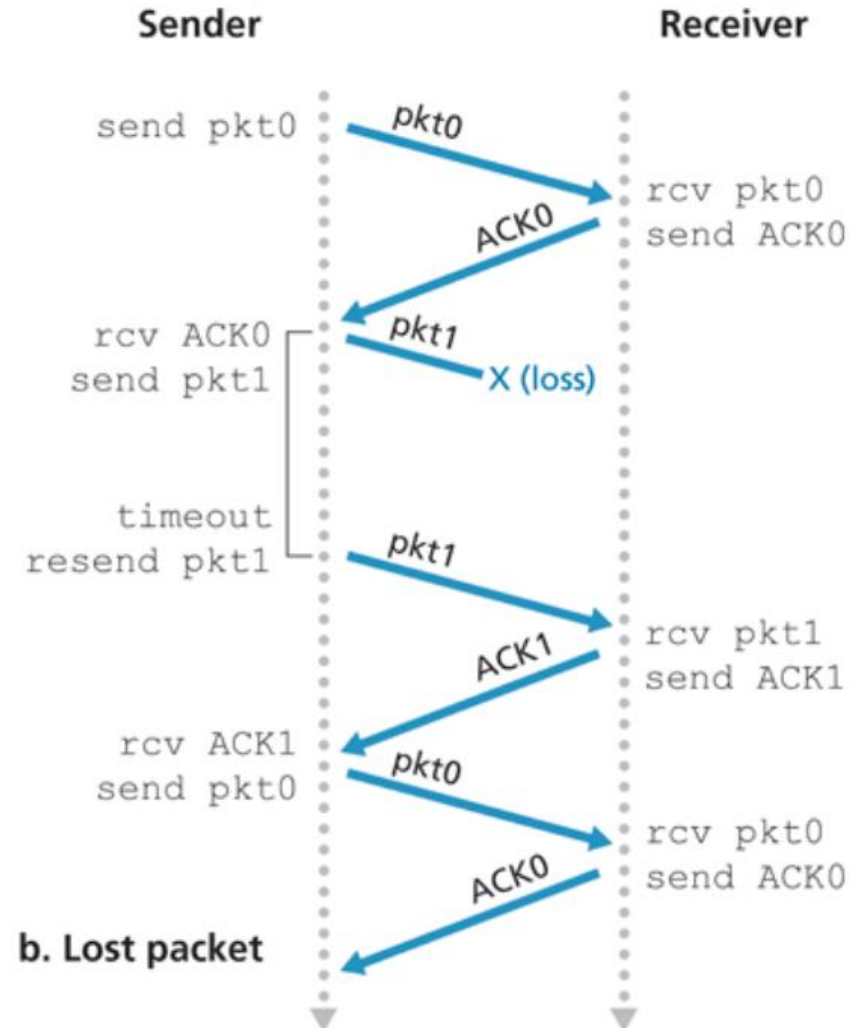
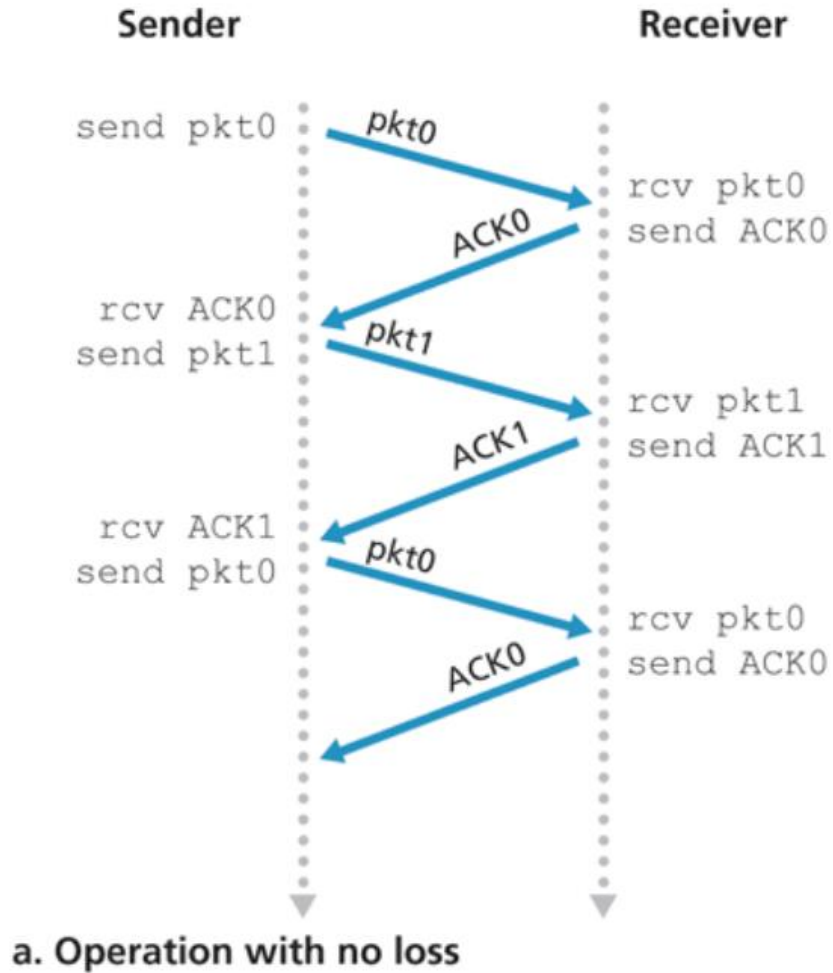
(Packets can get dropped or lost)



Transport Layer

Reliable Data Transfer 3.0

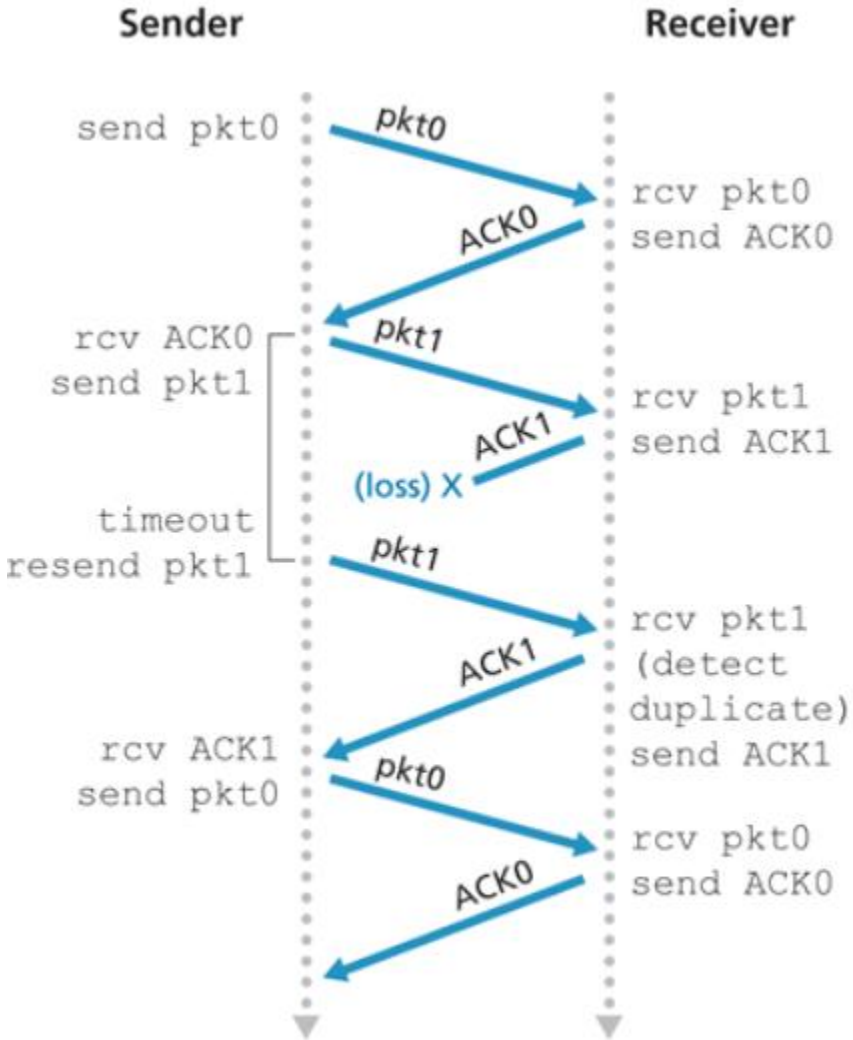
(Packets can get dropped or lost)



Transport Layer

Reliable Data Transfer 3.0

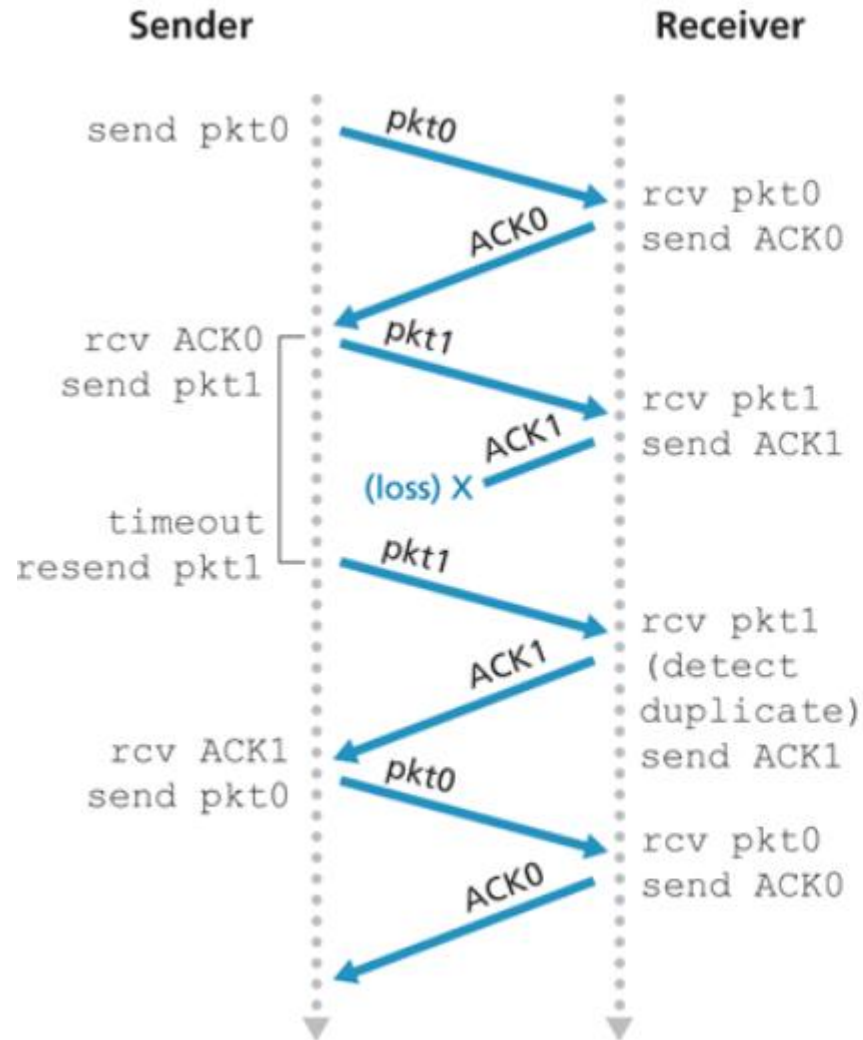
(Packets can get dropped or lost)



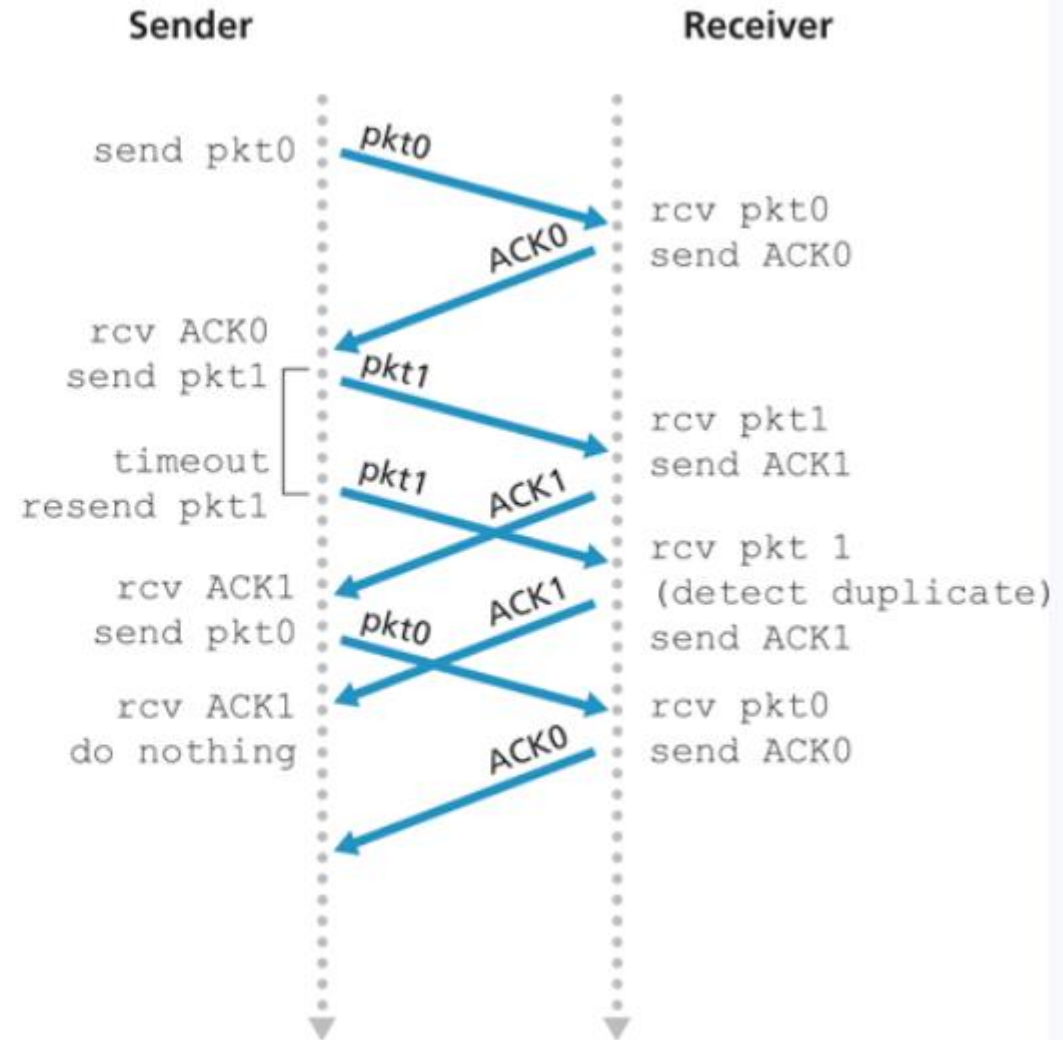
c. Lost ACK

Reliable Data Transfer 3.0

(Packets can get dropped or lost)



c. Lost ACK



d. Premature timeout