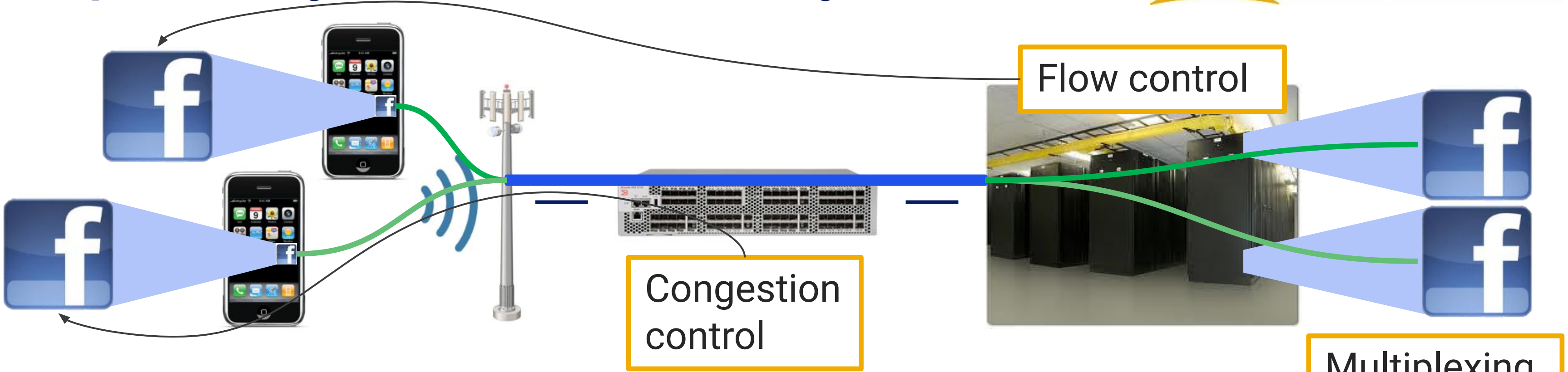
MONTANA STATE UNIVERSITY

Chapter 3

RDT

Transport layer functionality



Flow control

Congestion control

Multiplexing via sockets

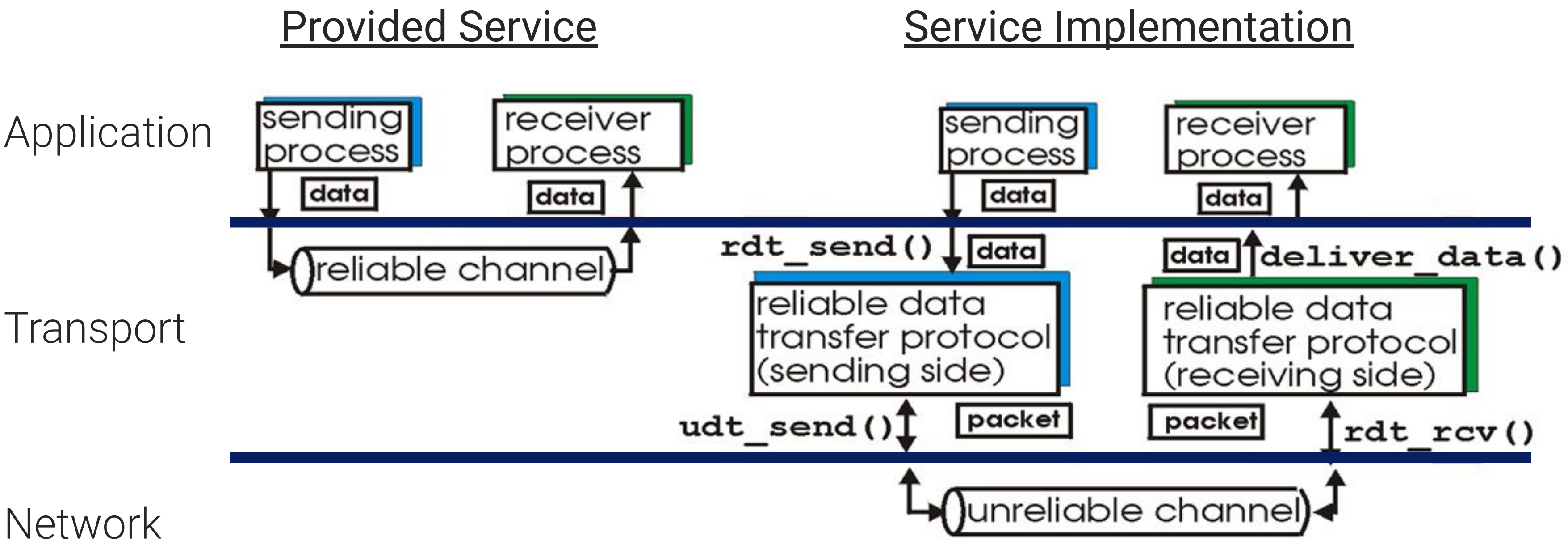
Inter-process comm.

- Segmentation and reassembly
- Error checking

- Reliability
- In-order delivery

7: Application	⟷	Application services	⟷	7: Application
4: Transport	⟷	Logical communications	⟷	4: Transport
3: Network	⟷	3: Network	⟷	3: Network
2: Data link	⟷	2: Data link	⟷	2: Data link
1: Physical	⟷	1: Physical	⟷	1: Physical

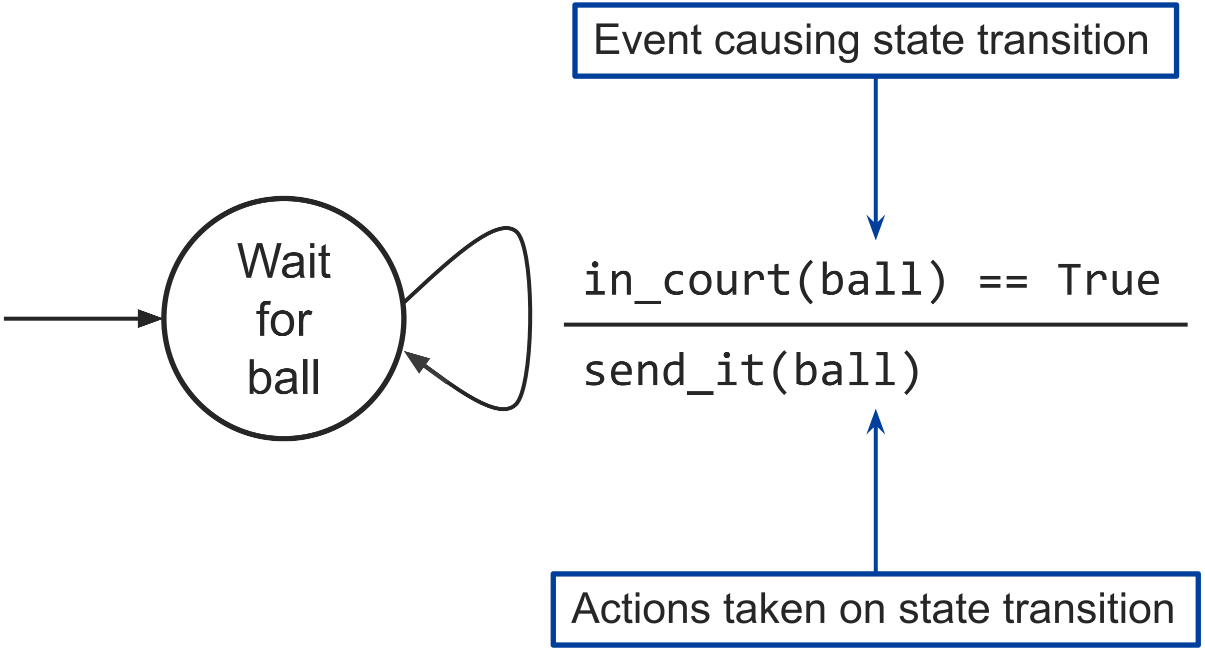
Principles of reliable data transfer



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?

Bruce Lee FSM

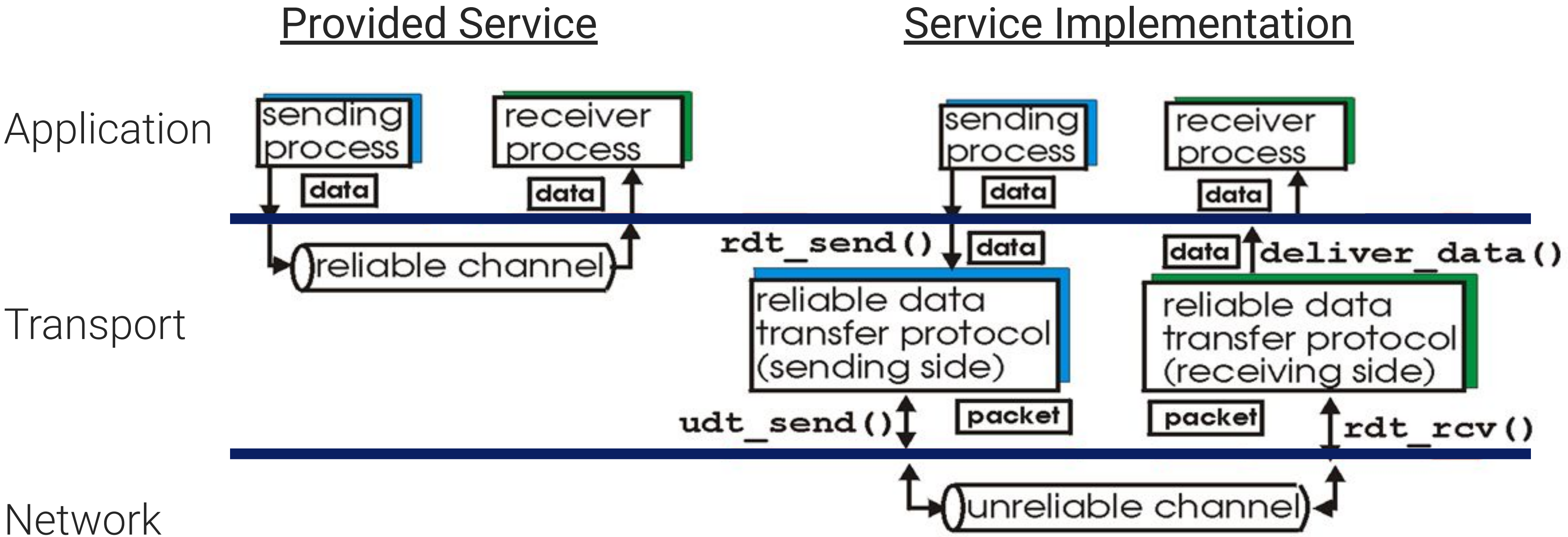


Event causing state transition

```
in_court(ball) == True
──────────────────────
send_it(ball)
```

Actions taken on state transition

Principles of reliable data transfer

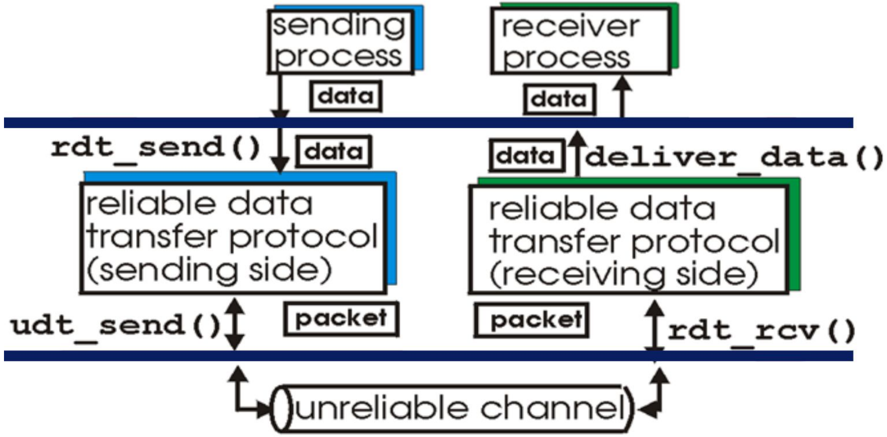


Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

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

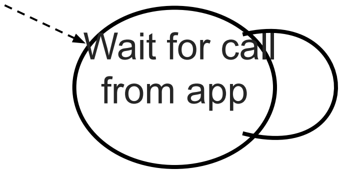
Reliable channel

- rdt1.0: reliable transfer over reliable channel
- Assumptions:
  - Unidirectional, long data flows
  - Perfectly reliable channel:
    - No bit errors
    - No packet loss
    - No packet reordering



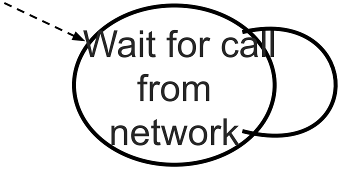
Event causing state transition

Sender



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

Receiver



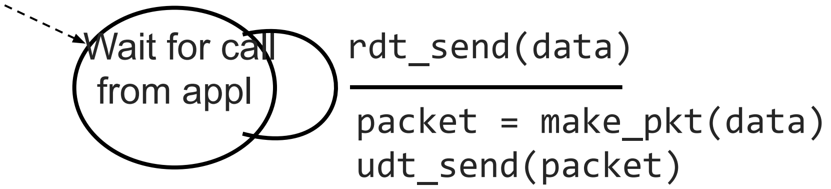
```
rdt_rcv(packet)
_______________
data = extract(packet)
deliver_data(data)
```

Actions taken on state transition

rdt2.0: Channel with bit errors

- How are errors detected?
  - Checksums: `make_pkt(data, checksum)`, `corrupt(rcvpkt)`
- How do humans recover from communication errors?
  - ACKs, NAKs, and retransmissions: `isACK(rcvpkt)`, `isNAK(rcvpkt)`
- Design sender and receiver FSMs for rdt2.0

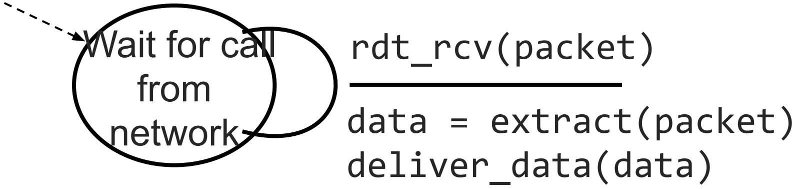
rdt1.0: **Sender** **Receiver**



Wait for call from appl

rdt_send(data)

packet = make_pkt(data)
udt_send(packet)



Wait for call from network

rdt_rcv(packet)

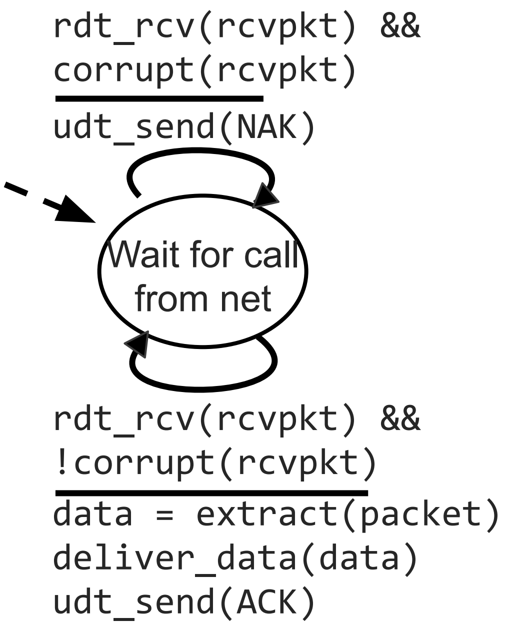
data = extract(packet)
deliver_data(data)

rdt2.0: Channel with bit errors

- How are errors detected?
  - Checksums
- How do humans recover from communication errors?
  - ACKs, NAKs, and retransmissions
- Design sender and receiver FSMs for rdt2.0

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

Receiver

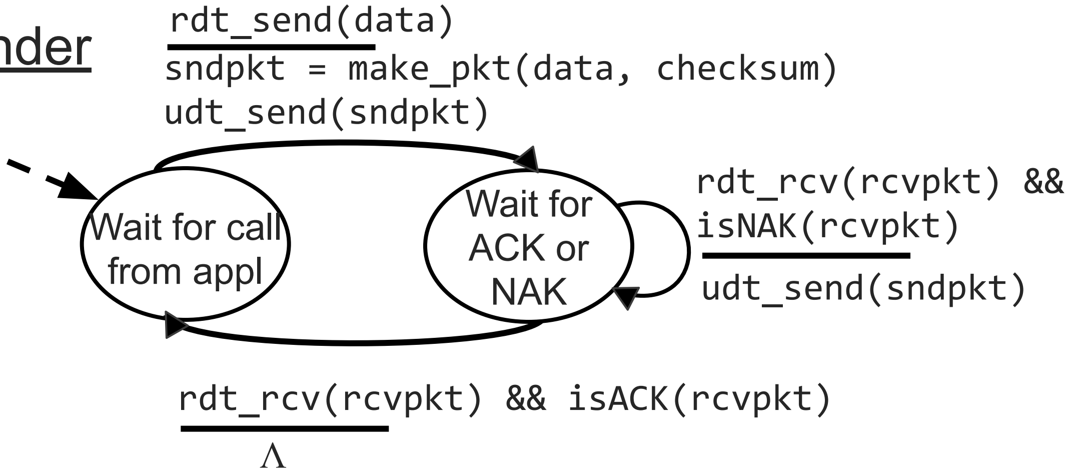


```
rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)
udt_send(NAK)
```

Wait for call from net

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

Sender



```
rdt_send(data)
sndpkt = make_pkt(data, checksum)
udt_send(sndpkt)
```

Wait for call from appl

Wait for ACK or NAK

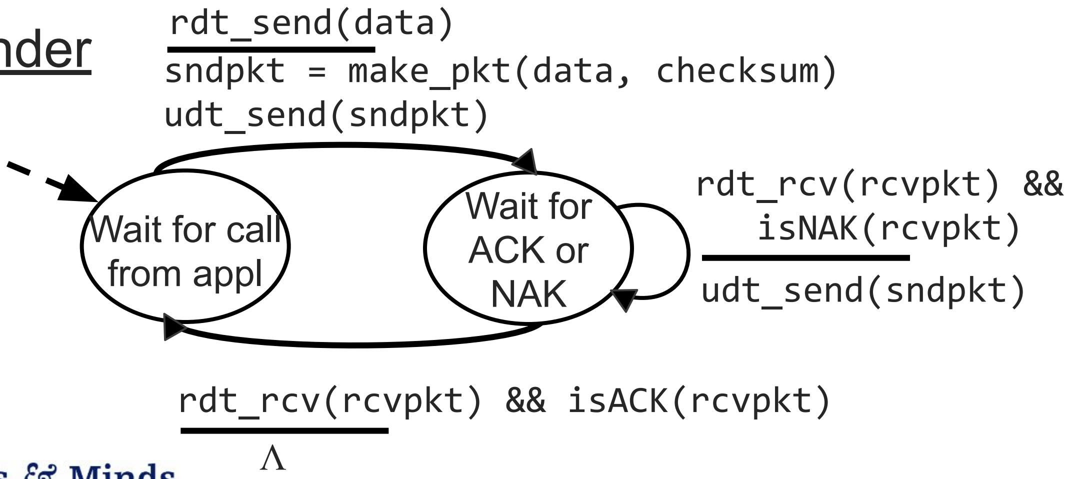
```
rdt_rcv(rcvpkt) &&
isNAK(rcvpkt)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) && isACK(rcvpkt)
Λ
```

rdt2.0: Channel with bit errors

- What happens if ACK/NAK corrupted?
  - Duplicate delivery, or no retransmission when needed
- How can we deal with corrupted ACKs/NAKs?
  - Retransmission, but can get duplicate packets
- How to handle duplicate packets?
  - Embed sequence numbers in packets: `make_pkt(seq_num, data, checksum)`, `get_seq_num(rcvpkt)`
  - Only need `0` and `1` for `seq_num`. Why?
- Come up with sender and receiver FSMs for rdt2.0 with sequence numbers and retransmissions

Sender

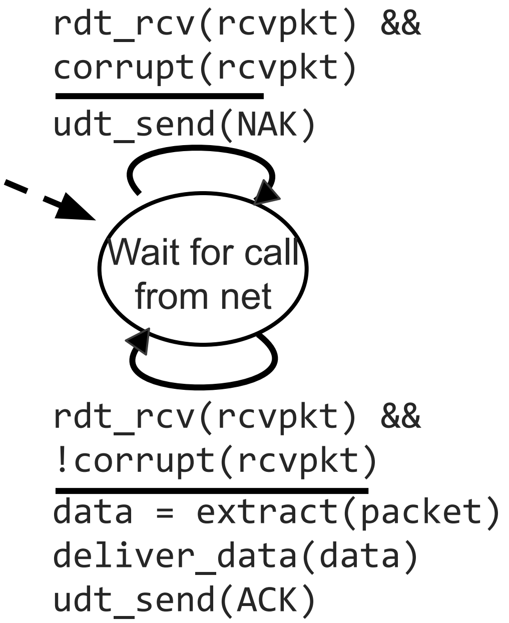


```
rdt_send(data)
sndpkt = make_pkt(data, checksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
  isNAK(rcvpkt)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) && isACK(rcvpkt)
Λ
```

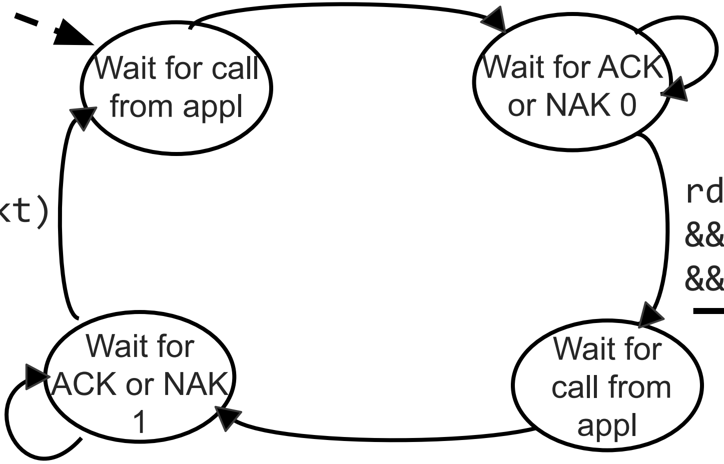
Receiver



```
rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)
udt_send(NAK)
```

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

rdt2.1: sender, handles garbled ACKs



```
rdt_send(data)
________________________________________
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) || isNAK(rcvpkt))
________________________________________
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt)
&& ! corrupt(rcvpkt)
&& isACK(rcvpkt)
________________________________________
```
Λ

```
rdt_send(data)
________________________________________
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
isNAK(rcvpkt))
________________________________________
udt_send(sndpkt)
```

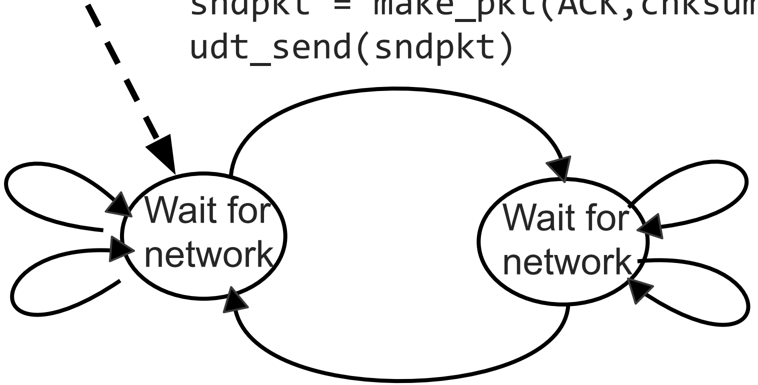
```
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt)
________________________________________
```
Λ

rdt2.1: receiver, handles garbled ACKs

Deliver data only once

```
rdt_rcv(rcvpkt) && !corrupt(rcvpkt)
  && get_seq_num(rcvpkt) == 0
data = extract(packet)
deliver_data(data)
sndpkt = make_pkt(ACK,chksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)
sndpkt =
 make_pkt(NAK,chksum)
udt_send(sndpkt)
```



```
rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)
sndpkt = make_pkt(NAK,chksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
get_seq_num(rcvpkt) == 1
sndpkt =
 make_pkt(ACK,chksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
!corrupt(rcvpkt) &&
get_seq_num(rcvpkt) == 1
data = extract(packet)
deliver_data(data)
sndpkt = make_pkt(ACK,chksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
!corrupt(rcvpkt) &&
get_seq_num(rcvpkt) == 0
sndpkt = make_pkt(ACK,chksum)
udt_send(sndpkt)
```

Loop on duplicates

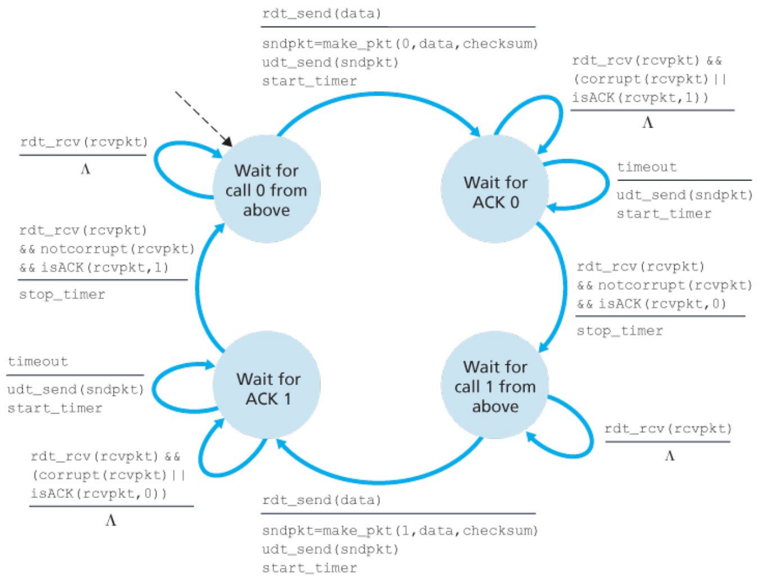
rdt3.0: bit errors and loss

- New assumption:
  - packet loss

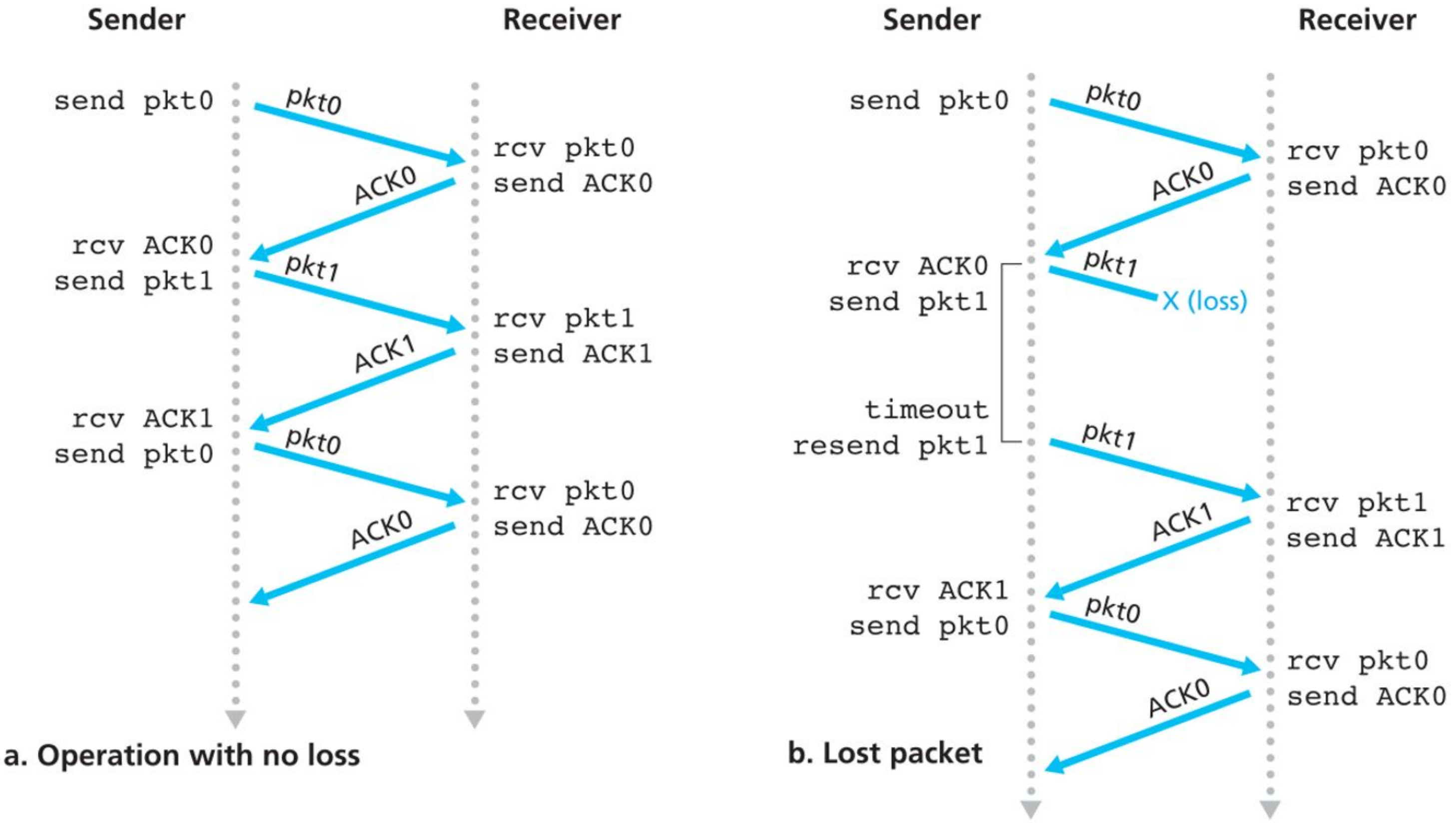
- How do we know a loss has occurred?
- What can we do about it?

- Approach:
  - Sender waits “reasonable” amount of time for ACK
  - Retransmits packet if no ACK received in this time

- What if packet/ACK only delayed?
  - Duplicate packets ignored at the receiver through sequence numbers
  - Receiver specifies sequences number of ACKed packet


rdt_send(data)
sndpkt=make_pkt(0,data,checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
Wait for call 0 from above
Wait for ACK 0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK 1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt=make_pkt(1,data,checksum)
udt_send(sndpkt)
start_timer

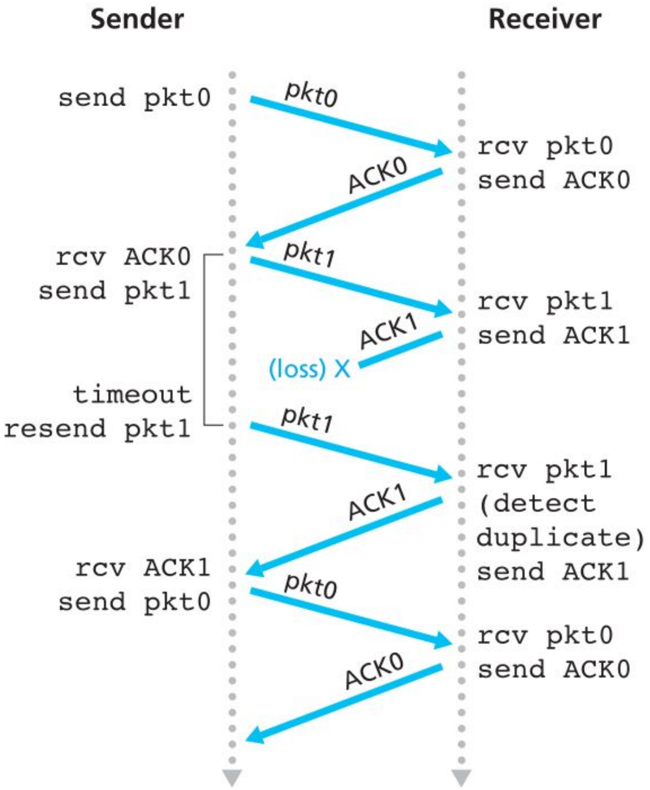
rdt3.0 in action



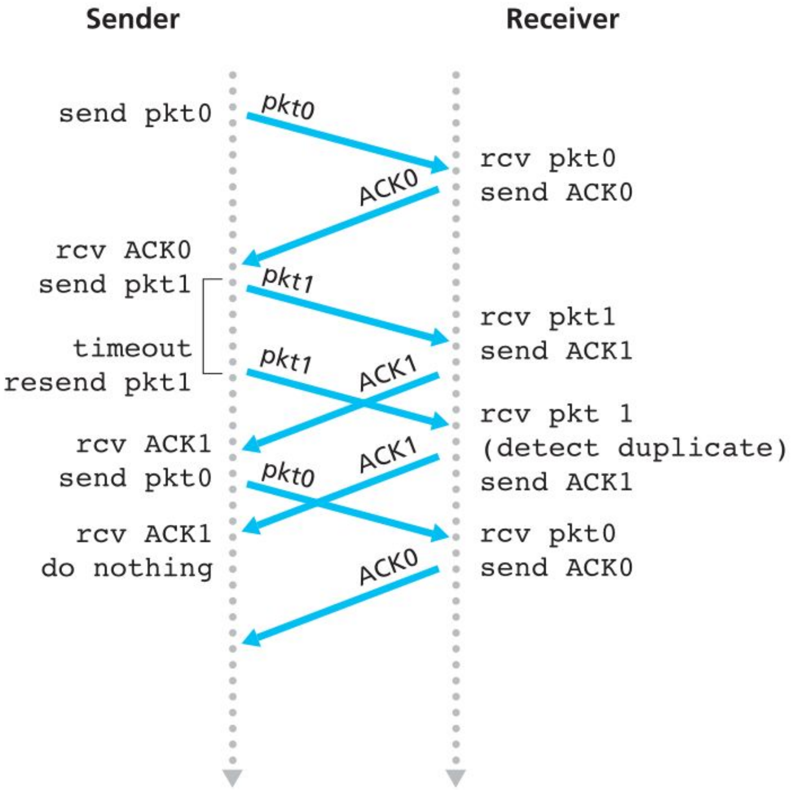
a. Operation with no loss

b. Lost packet

rdt3.0 in action



c. Lost ACK



d. Premature timeout

MONTANA
STATE UNIVERSITY