

# Data Link Layer: Reliable Data Transfer

Kameswari Chebrolu

All the figures used as part of the slides are either self created or from the public domain with either 'creative commons' or 'public domain dedication' licensing. The public sites from which some of the figures have been picked include: <http://commons.wikimedia.org> (Wikipedia, Wikimedia and workbooks); <http://www.sxc.hu> and <http://www.pixabay.com>

# Recap

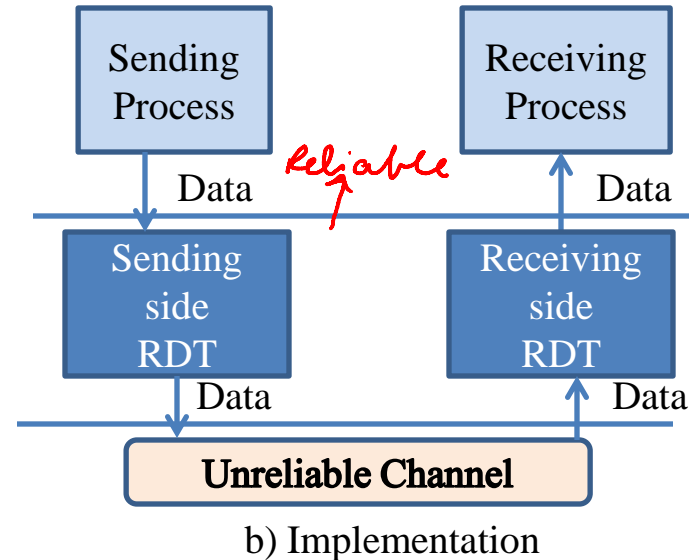
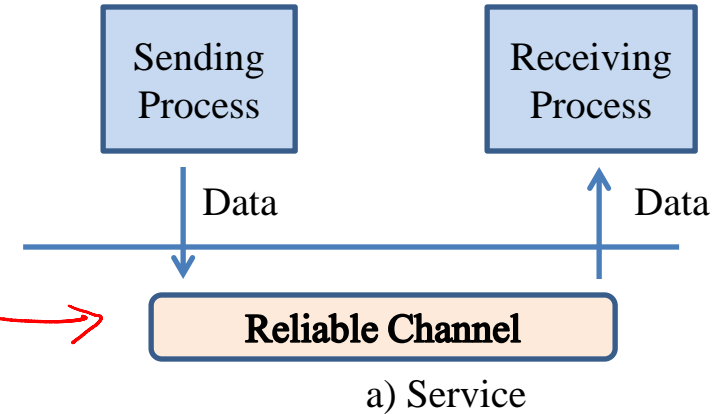
- Frame-by-Frame next-hop delivery
  - Frames can get corrupted or lost
  - Error Detection helps detect corrupted frames
  - What next?
- Recover the corrupted/lost frames → Reliable Data Transfer
  - One of the most researched problem in networking

Link Layer ✓  
Transport Layer

# Outline

- Develop a Reliable Data Transfer protocol (RDT)

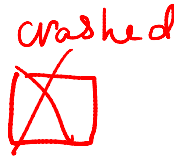
- Unreliable channel with bit errors
- Unreliable channel with bit errors and losses



# RDTv1.0: Channel with bit errors

→ corrupted, not lost

- Telephone Analogy
- Receiver Feedback
  - Positive: aha, ok, hmm → ACK
  - Negative: repeat that, didn't follow, what did you say? → NACK
  - Do we need both?
- Sender retransmits on NACK

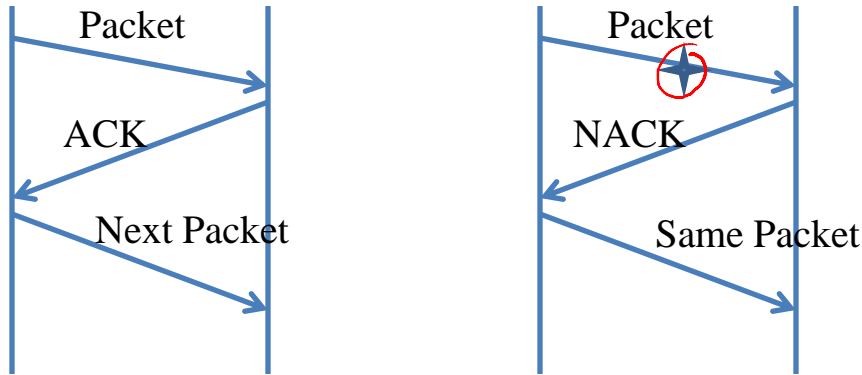


## Required Functionality:

- Error Detection mechanism ✓
  - Checksum, CRC etc
- ?

# Automatic Repeat Request (ARQ)

- Protocols based on Feedback and retransmissions



RDTv1.0

## Required Functionality:

- Error Detection mechanism
  - Checksum, CRC etc
- Receiver Feedback
  - ACK + NACK

# RDTr1.0 has a fatal flaw!

- What if the ACK/NACK got corrupted?
  - What should sender do then?
- Send next packet? If prev. pkt is lost, RDT not reliability ✗
- Send previous packet? If prev. pkt is not lost, creates duplicates ✓

## Required Functionality:

- Error Detection mechanism
  - Checksum, CRC etc
- Receiver Feedback
  - ACK + NACK
  - Data Sequence Numbers ✓

# RDTr2.0

- Receiver gives feedback (ACK, NACK)
- Sender retransmits 'sequenced' packet on NACK, garbled ACK/NACK
- Receiver discards duplicates if any based on sequence number

## Required Functionality:

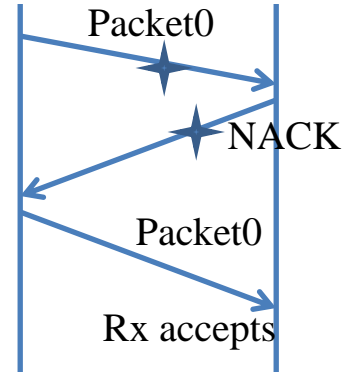
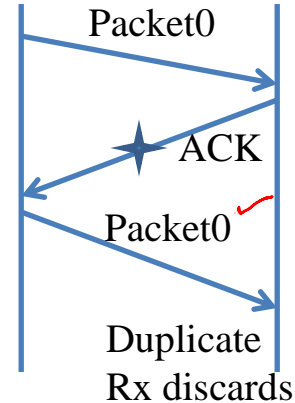
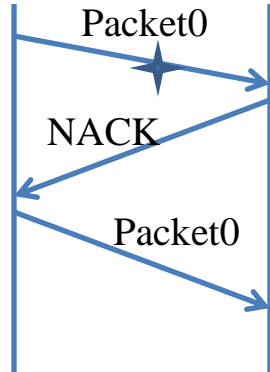
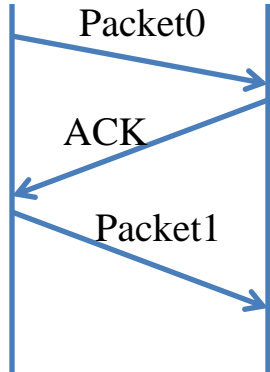
- Error Detection mechanism
  - Checksum, CRC etc
- Receiver Feedback
  - ACK + NACK
  - Data Sequence Numbers

# RDTv2.0

- What is the sequence number space?

– Just two seq #s “0 , 1” will suffice

min. Seq #  $\rightarrow \begin{cases} 0, 1, \dots, \text{infinite} \\ 0, 1, 2, 3 \\ 0, 1 \end{cases}$   
 $\rightarrow$  1-bit Receive Pkt

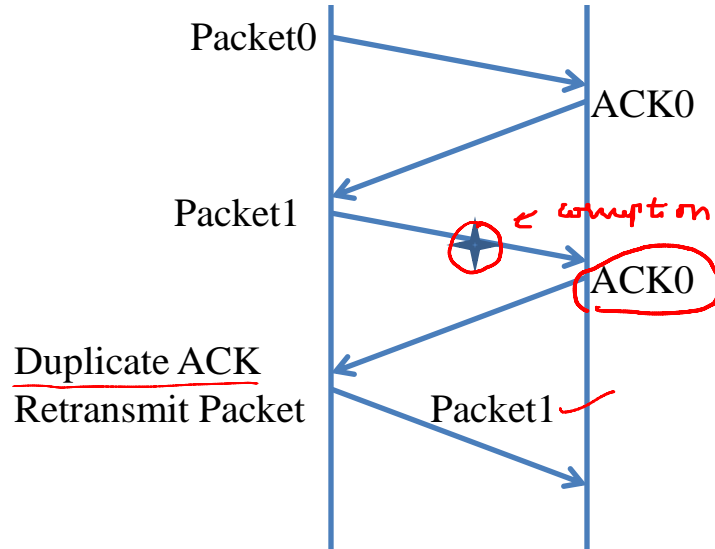




# RDTv2.1


- Optimization: NACK free operation
  - Convey same information as NACK but through ACK.  
How?
- Instead of NACK, receiver sends ACK of last correctly received packet
  - Receiver must explicitly include seq # of packet being ACKed
- Duplicate ACK at sender results in same action as NACK: retransmit current packet

# NACK Free Protocol



# RDTr2.1

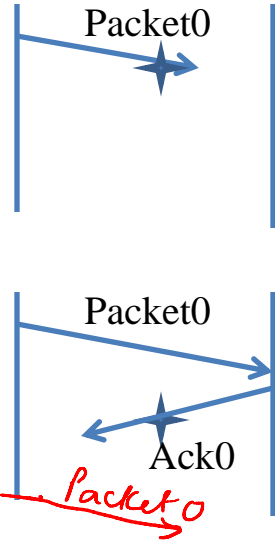
## Required Functionality:

- Error Detection mechanism ✓
    - Checksum, CRC etc
  - Receiver Feedback
    - ACK + ~~NACK~~
    - Data Sequence Numbers ✓
    - ACK carries sequence number of data packets
- 

# RDT: Channel with Errors and Losses

- Will RDTv2.1 work?
- Sender gets no feedback: Need a Timeout mechanism
- How long to wait?

↳ Link, Tx time, Prog time, Processing to ~~Packet0~~



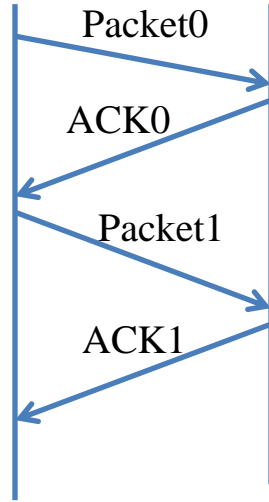
# RDTv3.0

- 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, seq. #’s help resolve this

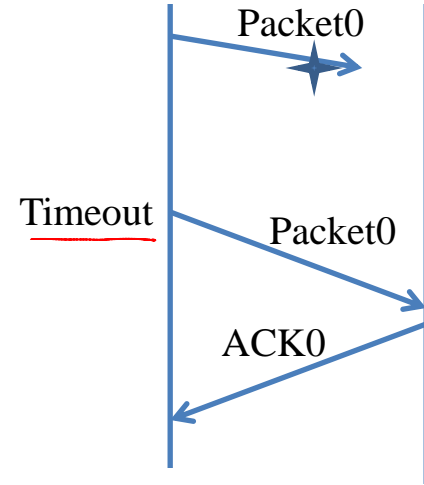
## Required Functionality:

- Error Detection mechanism
  - Checksum, CRC etc
- Receiver Feedback
  - ACK + NACK
  - Data Sequence Numbers
  - ACK carries data seq. No.
  - Timeout Timer

# RDTrv3.0: Stop and Wait Protocol In Action



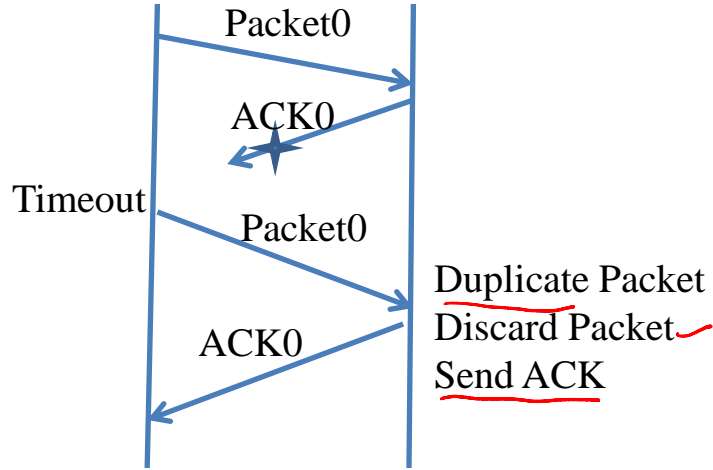
(a) No Loss



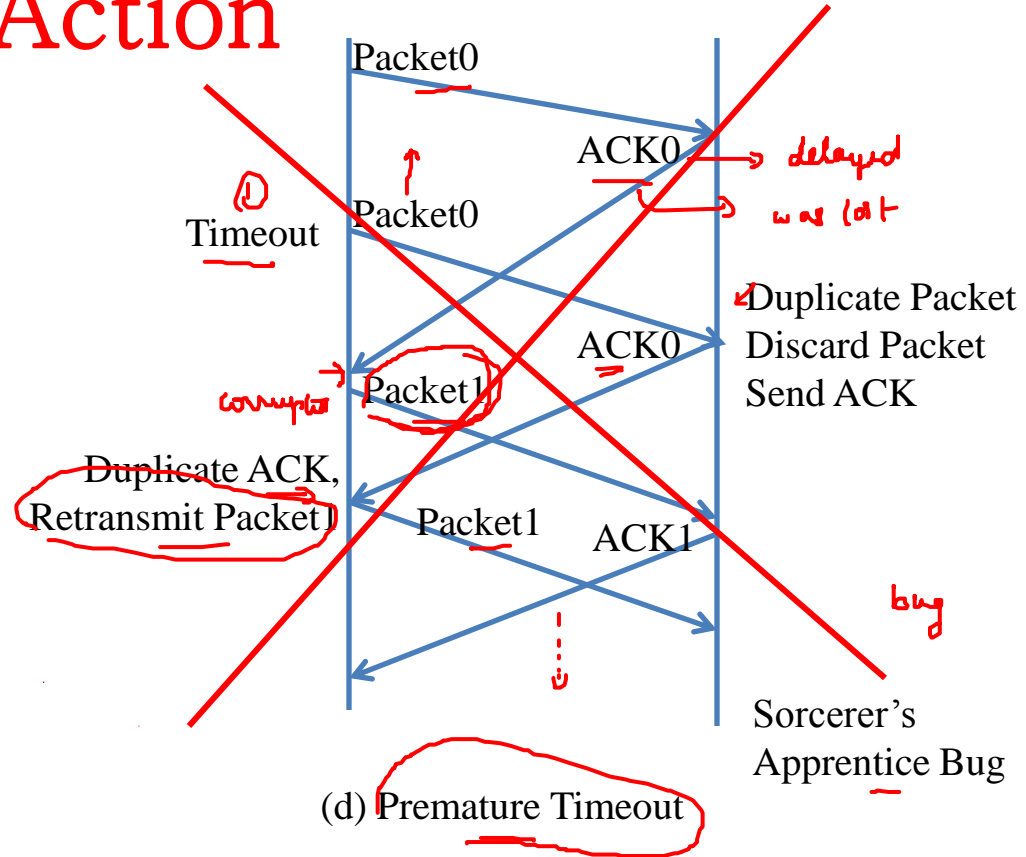
(b) Lost Packet

Also called Alternating Bit Protocol

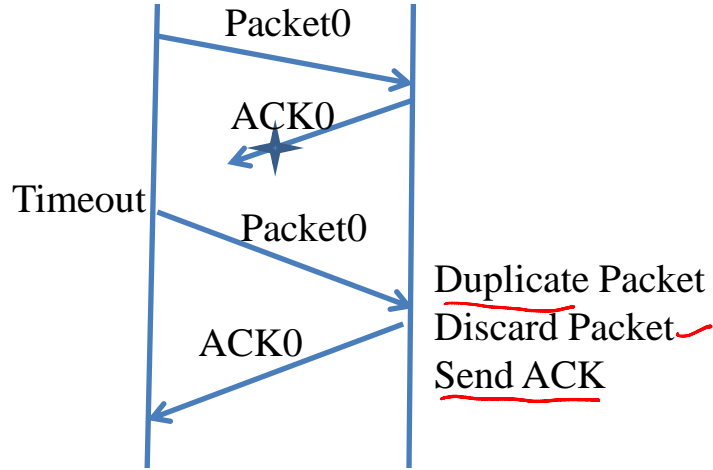
# RDTv3.0: Stop and Wait Protocol In Action



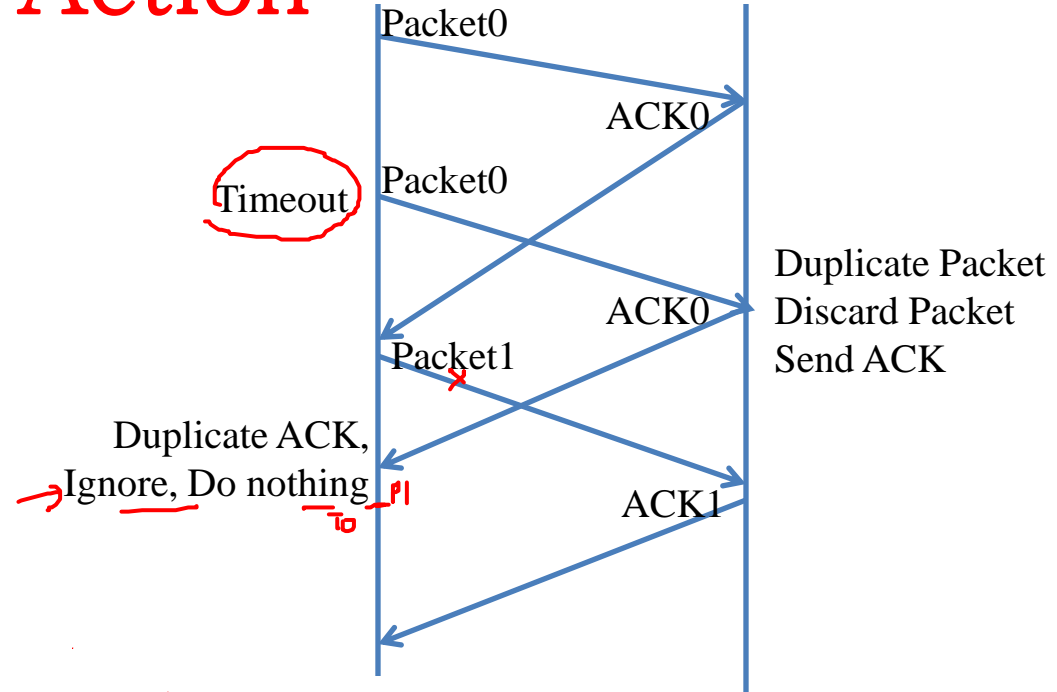
(c) Lost ACK



# RDTrv3.0: Stop and Wait Protocol In Action



(c) Lost ACK



(d) Premature Timeout



# Design of RDT protocols

- Many challenges to handle
  - ACKs/Packet loss, ACKs/Packet delayed,
  - Duplication of packets, Reordering, Incorrect timeout timer settings, Receiver capabilities → Buffer
  - Protocol has to work correctly and efficiently in spite of all this

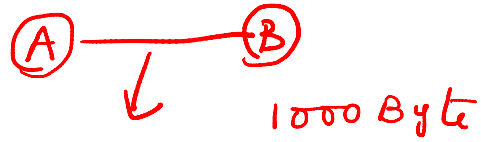
TCP

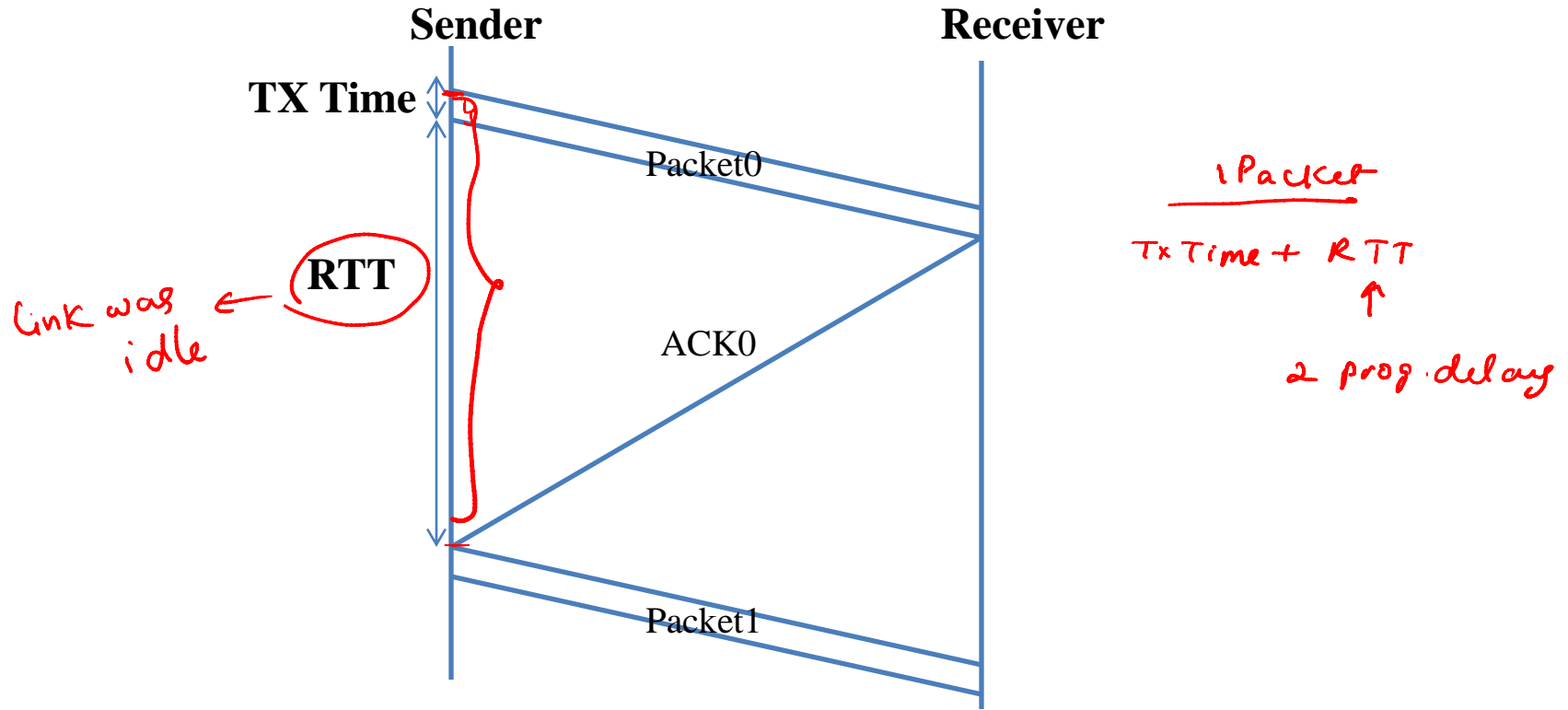
# NACK vs ACK

$x$   $\downarrow x+1$   $\rightarrow$  long time  
 $(x+2)$

- Can conclude packet loss on detecting 'holes'
  - Long delay between some packets can slow down recovery
  - What if the last packet in the flow is lost? *dead lock*
    - Receiver doesn't generate NACK, sender assume 'all is well'
- Advantage of NACK: If errors are infrequent, reduces overhead of feedback *NACK + ACK*

# Performance of Stop and Wait Protocol

- What is the achieved throughput?
  - 10 Mbps link, 10 ms prop. delay, 1KB packet, ACK too small (ignore its Transmission time)
- Throughput: 8000 bits / [(8000/10<sup>7</sup>) + 2\*0.010]  
= 384.6Kbps
- Utilization = 384.6kbps/10000kbps = 3.8%



$$\text{Utilization} = \text{Transmission time} / (\text{Transmission time} + \text{RTT})$$

# Summary

- Reliable data transfer protocols provide 'reliable channel' service abstraction to higher layers
- We incrementally determined the required functionality needed in RDT protocols
  - bit errors
  - losses
- The current protocol designed is inefficient
- Future: Build on this framework to design better protocols
  - functionality