

Principles of Reliable Data Transfer

Study-Ready Notes

Compiled by Andrew Photinakis

October 17, 2025

Contents

1	Introduction to Reliable Data Transfer	2
2	Reliable Data Transfer Service Abstraction	2
2.1	Service Model	2
3	RDT Protocol Interfaces and Mechanisms	2
3.1	Protocol Interface Functions	2
3.2	Key Protocol Mechanisms	3
4	Finite State Machine Specification	3
4.1	FSM Concepts	3
5	RDT1.0: Reliable Transfer over Reliable Channel	3
5.1	Assumptions	3
5.2	FSM Specifications	4
6	RDT2.0: Channel with Bit Errors	4
6.1	New Challenges	4
6.2	Error Recovery Mechanisms	4
7	RDT2.1: Handling Garbled ACK/NAKs	4
7.1	The Fatal Flaw in RDT2.0	4
7.2	Solution: Sequence Numbers	5
8	RDT2.2: NAK-Free Protocol	5
8.1	Approach	5
9	RDT3.0: Channels with Errors and Loss	6
9.1	New Channel Assumptions	6
9.2	Solution: Countdown Timer	6

10 RDT3.0 Operation Scenarios	6
10.1 Four Key Scenarios	6
11 Performance Analysis of RDT3.0	7
11.1 Sender Utilization	7
11.2 Utilization Formula	7
11.3 Example Calculation	7
12 Pipelined Protocols	8
12.1 Limitations of Stop-and-Wait	8
12.2 Pipelining Solution	8
12.3 Utilization with Pipelining	8
13 Go-Back-N Protocol	8
13.1 Sender Operation	8
13.2 Sender Window Structure	9
14 Selective Repeat Protocol	9
14.1 Key Features	9
14.2 Window Size Constraint	9
15 Study Aids and Exam Preparation	10
15.1 Key Concepts to Master	10
15.2 Practice Questions	10
16 Summary	11

1 Introduction to Reliable Data Transfer

- Core component of transport-layer services
- Essential for connection-oriented protocols like TCP
- Builds upon multiplexing/demultiplexing concepts
- Addresses challenges of unreliable network channels

[Summary: Reliable data transfer protocols ensure data delivery despite unreliable network conditions, using mechanisms like acknowledgments, sequence numbers, and retransmissions.]

2 Reliable Data Transfer Service Abstraction

2.1 Service Model

- **Abstraction:** Perfectly reliable channel between sending and receiving processes
- **Reality:** Underlying channel is unreliable (loses, corrupts, reorders data)
- **Implementation:** Transport layer protocols handle reliability

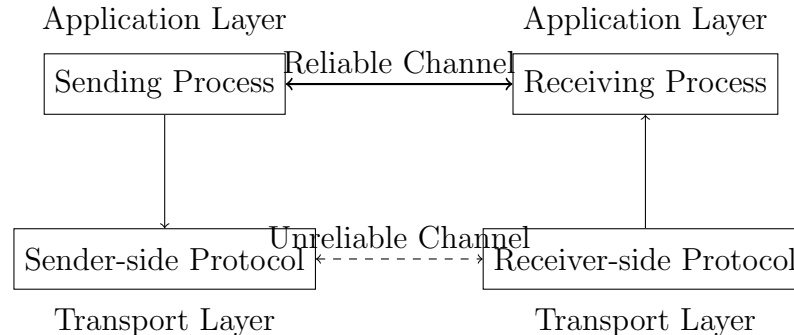


Figure 1: Reliable data transfer: abstraction vs implementation

3 RDT Protocol Interfaces and Mechanisms

3.1 Protocol Interface Functions

- **rdt_send():** Called from above to deliver data to receiver upper layer
- **udt_send():** Called by RDT to transfer packet over unreliable channel
- **rdt_rcv():** Called when packet arrives on receiver side
- **deliver_data():** Called by RDT to deliver data to upper layer

3.2 Key Protocol Mechanisms

- Error detection (checksums)
- Acknowledgments (ACKs) and Negative Acknowledgments (NAKs)
- Retransmission
- Sequence numbers for duplicate detection
- Timers for loss detection

[Mnemonic: "ACE RT" - Acknowledgments, Checksums, Error detection, Retransmission, Timers - key RDT mechanisms.]

4 Finite State Machine Specification

4.1 FSM Concepts

- Used to specify sender and receiver behavior
- **State:** Current condition of protocol entity
- **Event:** Trigger that may cause state transition
- **Actions:** Operations performed during transition

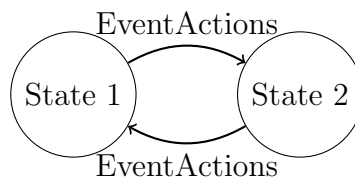


Figure 2: Finite State Machine basic structure

5 RDT1.0: Reliable Transfer over Reliable Channel

5.1 Assumptions

- Underlying channel is perfectly reliable
- No bit errors
- No loss of packets

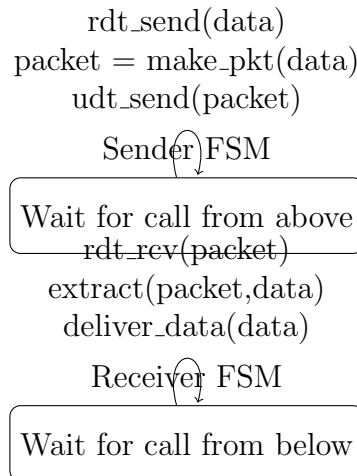


Figure 3: RDT1.0: Simple FSMs for perfectly reliable channel

5.2 FSM Specifications

[Summary: RDT1.0 assumes perfect channel conditions - sender simply sends packets, receiver simply receives and delivers them without error handling.]

6 RDT2.0: Channel with Bit Errors

6.1 New Challenges

- Underlying channel may flip bits in packets
- Use checksums to detect bit errors
- Recovery mechanisms needed

6.2 Error Recovery Mechanisms

- **Acknowledgments (ACKs)**: Receiver tells sender packet received OK
- **Negative Acknowledgments (NAKs)**: Receiver tells sender packet had errors
- **Stop-and-wait**: Sender sends one packet, waits for receiver response

7 RDT2.1: Handling Garbled ACK/NAKs

7.1 The Fatal Flaw in RDT2.0

- What if ACK/NAK becomes corrupted?
- Sender doesn't know what happened at receiver

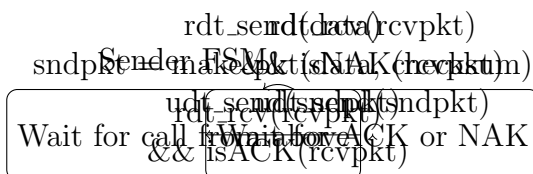


Figure 4: RDT2.0 sender FSM handling bit errors

- Can't just retransmit: possible duplicate

7.2 Solution: Sequence Numbers

- Add sequence number to each packet
- Two sequence numbers (0,1) suffice for stop-and-wait
- Receiver discards duplicate packets

[Concept Map: RDT Evolution → RDT1.0 (perfect channel) → RDT2.0 (bit errors, ACK/NAK) → RDT2.1 (garbled ACKs, sequence numbers) → Each version adds mechanisms to handle more realistic channel impairments.]

8 RDT2.2: NAK-Free Protocol

8.1 Approach

- Same functionality as RDT2.1 but using ACKs only
- Instead of NAK, receiver sends ACK for last packet received OK
- Receiver explicitly includes sequence number of packet being ACKed
- Duplicate ACK at sender triggers retransmission (same as NAK)

RDT2.1 (with NAKs)	RDT2.2 (NAK-free)
Uses ACK and NAK messages	Uses only ACK messages
Receiver sends NAK for corrupted packets	Receiver sends ACK for last good packet
Simpler logic	Reduced message types
Two response types	Unified response mechanism
Used in: Educational examples	Used in: TCP (real-world)

Table 1: Comparison of RDT2.1 vs RDT2.2

9.1 New Channel Assumptions

- ## 9.2 Solution: Countdown Timer

- ```

 rdt_send(data)
 sndpkt = make_pkt(0, data, checksum)
 RDT3.0 Sender rdt_send(sndpkt)
 FSM (partial)
 start timer
 timeoutudt_wait(sndpkt) start timer for ACK0
 if rdt_rcv(rcvpkt)
 && notcorrupt(rcvpkt)
 && isACK(rcvpkt,1)
 stop timer
 timeoutudt_wait(sndpkt) start timer for ACK1

```

## 10 RDT3.0 Operation Scenarios

## 10.1 Four Key Scenarios

- [Summary: RDT3.0 handles both bit errors and packet loss using checksums, sequence numbers, acknowledgments, and countdown timers, providing complete reliability over unreliable channels.]

| Scenario          | Problem                     | Solution                          |
|-------------------|-----------------------------|-----------------------------------|
| No loss           | None                        | Normal operation                  |
| Packet loss       | Data packet lost in transit | Timeout and retransmission        |
| ACK loss          | Acknowledgment lost         | Timeout and retransmission        |
| Premature timeout | ACK delayed, not lost       | Sequence numbers handle duplicate |

Table 2: RDT3.0 operation scenarios and solutions

## 11 Performance Analysis of RDT3.0

### 11.1 Sender Utilization

- **U<sub>sender</sub>**: Fraction of time sender is busy sending
- For stop-and-wait protocol: very low utilization

### 11.2 Utilization Formula

$$U_{sender} = \frac{L/R}{RTT + L/R}$$

Where:

- $L$  = packet length (bits)
- $R$  = transmission rate (bps)
- $RTT$  = round-trip time

### 11.3 Example Calculation

- 1 Gbps link, 15 ms propagation delay, 8000 bit packet
- Transmission time:  $D_{trans} = \frac{L}{R} = \frac{8000}{10^9} = 8$  microseconds
- $RTT = 2 \times 15$  ms = 30 ms
- $U_{sender} = \frac{0.008}{30.008} = 0.00027$

[Mnemonic: "Low Utilization Long Wait" - Stop-and-wait protocols have low utilization due to waiting for ACKs.]



## 12 Pipelined Protocols

### 12.1 Limitations of Stop-and-Wait

- Very low utilization (0.027% in example)
- Protocol limits performance of underlying infrastructure
- Most of the time, sender is idle waiting for ACK

### 12.2 Pipelining Solution

- Allow multiple "in-flight" packets
- Increase range of sequence numbers
- Require buffering at sender and/or receiver
- Dramatically increases utilization

### 12.3 Utilization with Pipelining

$$U_{sender} = \frac{N \times L/R}{RTT + L/R}$$

Where  $N$  is the number of pipelined packets.

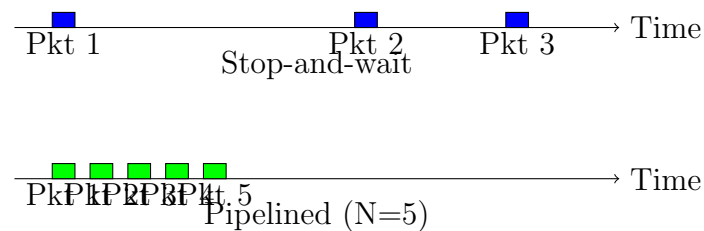


Figure 6: Comparison of stop-and-wait vs pipelined transmission

## 13 Go-Back-N Protocol

### 13.1 Sender Operation

- Window of up to  $N$  consecutive transmitted but unACKed packets
- $k$ -bit sequence number in packet header
- **Cumulative ACK:** ACK( $n$ ) acknowledges all packets up to and including sequence number  $n$

- Single timer for oldest in-flight packet
- Timeout(n): retransmit packet n and all higher sequence number packets in window

## 13.2 Sender Window Structure

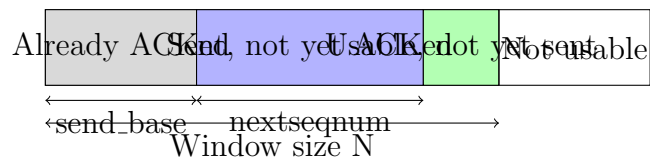


Figure 7: Go-Back-N sender window structure

## 14 Selective Repeat Protocol

### 14.1 Key Features

- Receiver individually acknowledges all correctly received packets
- Buffers out-of-order packets for eventual in-order delivery
- Sender maintains timer for each unACKed packet
- More efficient but more complex than Go-Back-N

### 14.2 Window Size Constraint

- Critical relationship between sequence number space and window size
- For sequence numbers 0 to  $2^k - 1$ , window size must satisfy:

$$N \leq 2^{k-1}$$

- Prevents ambiguity between old and new packets

[Concept Map: Pipelined Protocols → Go-Back-N (cumulative ACKs, simple) vs Selective Repeat (individual ACKs, efficient) → Trade-off between complexity and performance → Window size constraints critical for correctness.]

| Go-Back-N                     | Selective Repeat             |
|-------------------------------|------------------------------|
| Cumulative acknowledgments    | Individual acknowledgments   |
| Discards out-of-order packets | Buffers out-of-order packets |
| Single timer for window       | Timer per packet             |
| Simpler implementation        | More complex implementation  |
| Less efficient with loss      | More efficient with loss     |
| Window: $N \leq 2^k - 1$      | Window: $N \leq 2^{k-1}$     |

Table 3: Comparison of Go-Back-N vs Selective Repeat

## 15 Study Aids and Exam Preparation

### 15.1 Key Concepts to Master

- Understand the evolution of RDT protocols and what problem each version solves
- Be able to draw and interpret FSM diagrams for each protocol version
- Calculate sender utilization for stop-and-wait and pipelined protocols
- Explain the window size constraints for Go-Back-N and Selective Repeat
- Compare and contrast different reliability mechanisms

### 15.2 Practice Questions

1. **Trace the evolution** of RDT protocols from 1.0 to 3.0. What specific problem does each new version address, and what mechanism does it introduce?
2. Calculate the **sender utilization** for a stop-and-wait protocol with the following parameters: packet size = 1500 bytes, transmission rate = 100 Mbps, RTT = 50 ms. How does this change if we use pipelining with a window size of 10?
3. Explain why **Selective Repeat** requires that the window size be at most half the sequence number space, while **Go-Back-N** can use almost the entire sequence number space.
4. Draw the **FSM for RDT3.0 sender** and explain how it handles packet loss, ACK loss, and premature timeouts.
5. Compare the **performance characteristics** of Go-Back-N vs Selective Repeat in a network with high packet loss rates. Which would you choose and why?

[Mnemonic: "RDT 1-2-3" - RDT1.0 (perfect), RDT2.0 (errors), RDT3.0 (loss) - each adds one more real-world problem to solve.]

## 16 Summary

- Reliable data transfer protocols build reliability over unreliable channels
- Key mechanisms: error detection, acknowledgments, sequence numbers, timers, re-transmissions
- RDT protocol evolution addresses increasingly realistic channel impairments
- Stop-and-wait protocols have poor utilization; pipelining dramatically improves performance
- Go-Back-N and Selective Repeat are two pipelined approaches with different trade-offs
- Window size constraints are critical for protocol correctness