Computer Networks

CSEN 503

Transport Layer Services and Protocols (Part 01)

Media Engineering and Technology | GUC

October 21, 2023

Table of Contents

Transport-Layer

- The transport layer enables multiple processes on one host to communicate with multiple processes on another remote host over the network.
- The transport layer extends the network layer services from enabling two end systems to communicate to allowing processes on the end systems to communicate.
- One key challenge for transport-layer protocols is enabling reliable message delivery between remote processes when the underlying infrastructure may lose and corrupt data.

Transport-Layer Protocols

- Transport-layer protocols provide a logical communication link between the processes running on different end systems.
- This logical link hides the complex communication paths (physical infrastructure) between the two end systems.
- Transport-layer protocols are implemented in the end systems, not in the router or switches (except SDN switches).
- The transport layer protocol will break each message received from the application layer into small chunks (segments) and send these chunks over the network by handing them to the network layer.
- On the destination end system, the transport layer receives the segments from the network layer. It reconstructs the application layer message from the sender and hands it over to the application layer at the destination.

Transport Layer in TCP/IP Networks

- The UDP (User Datagram Protocol) and the TCP (Transmission Control Protocol) are the most common transport-layer protocols in TCP/IP networks.
- The UDP provides **unreliable**, **connectionless** transport services to the network application.
- The TCP provides **reliable**, **connection-oriented** services to the network application.
- The IP (Internet Protocol) is the most common and important network layer protocol in TCP/IP networks.
- The IP protocol provides logical communication between hosts.
- The IP service model is a **best-effort** delivery service (unreliable service)

Transport-Layer Services/Protocols

What are the main functions any transport layer protocol needs to provide?

- Extending host-to-host delivery to process-to-process delivery (aka transport-layer multiplexing and demultiplexing)
- Provide data integrity validation and error detection (not error correction or recovery)

Providing error detection and integrity checking does not mean reliable data transfer.
The network application could mix/use more than one transport layer protocol.

Multiplexing and Demultiplexing

- A network process could have one or more network sockets.
- The transport layer protocol does not deliver the messages directly to the process; instead, it provides the message to a socket.
- Each network socket must have a unique identifier (Why?)
- Getting the message from the socket, dividing it into segments, and passing the segments to the network layer is multiplexing.
- Getting the segments from the network layer, identifying the socket that should receive the message, and passing it to the socket is demultiplexing.

Note: The multiplexing and demultiplexing functions occur in any communication layered architecture whenever multiple protocols use a single protocol at the adjacent lower layer.

Multiplexing and Demultiplexing

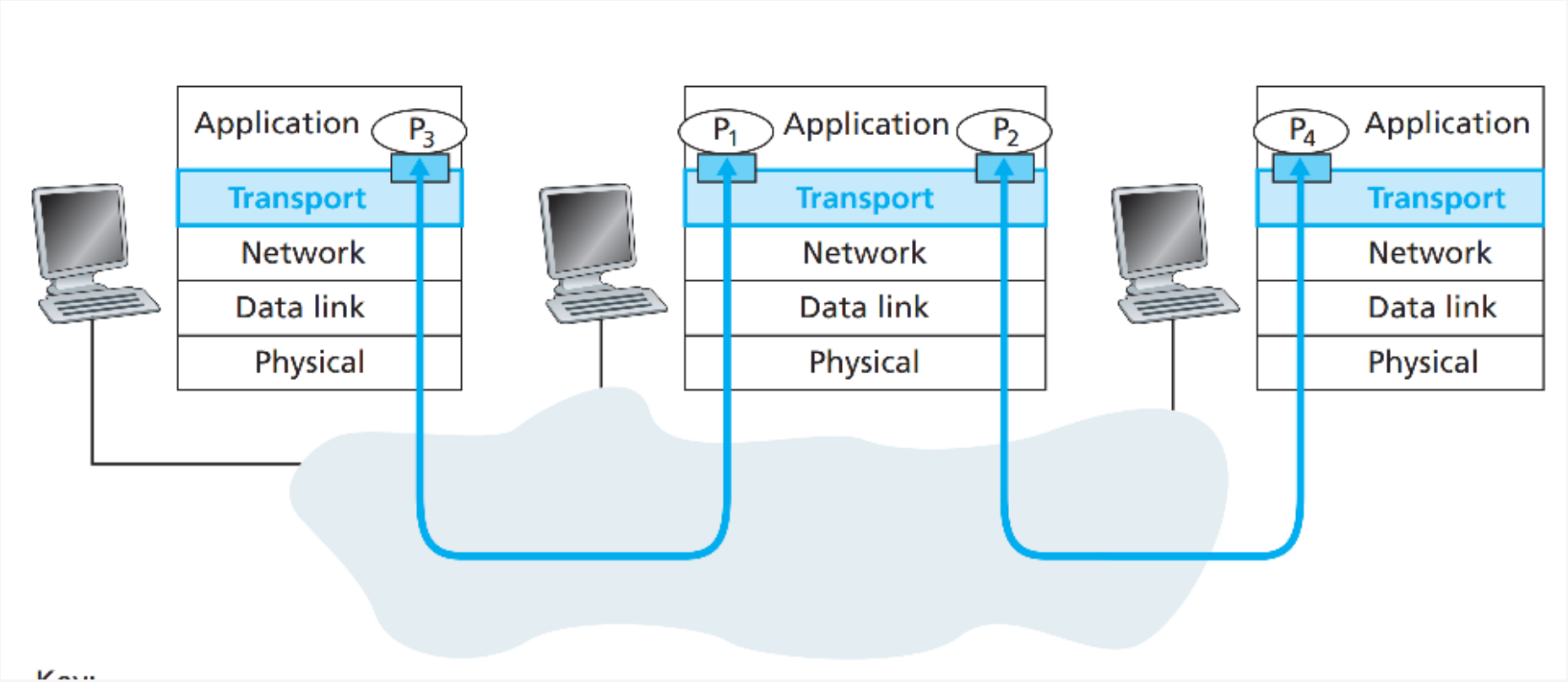


Figure: Transport-layer multiplexing and demultiplexing

Transport Layer Sockets

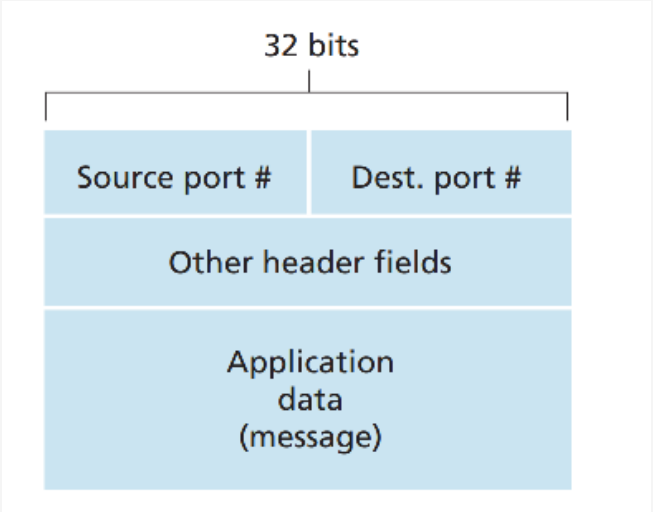


Figure: Source and destination port-number fields in a transport-layer segment

- Any transport layer protocols must be able to distinguish between network sockets.
- Therefore, each socket must have a unique identifier.
- The transport layer segment or message must carry this identifier.

UDP and TCP Sockets

- The unique identifier of the socket is the combination of port numbers and IP addresses.
- A UDP socket is identified by the destination IP address and the destination port number.
- If two UDP segments have different source IP addresses and/or source port numbers but have the same destination IP address and destination port number. The two segments will be passed to the same destination process via the same destination socket.
- A TCP socket is identified by the destination IP address, the destination port number, the source IP address, and the source port number.

UDP Socket

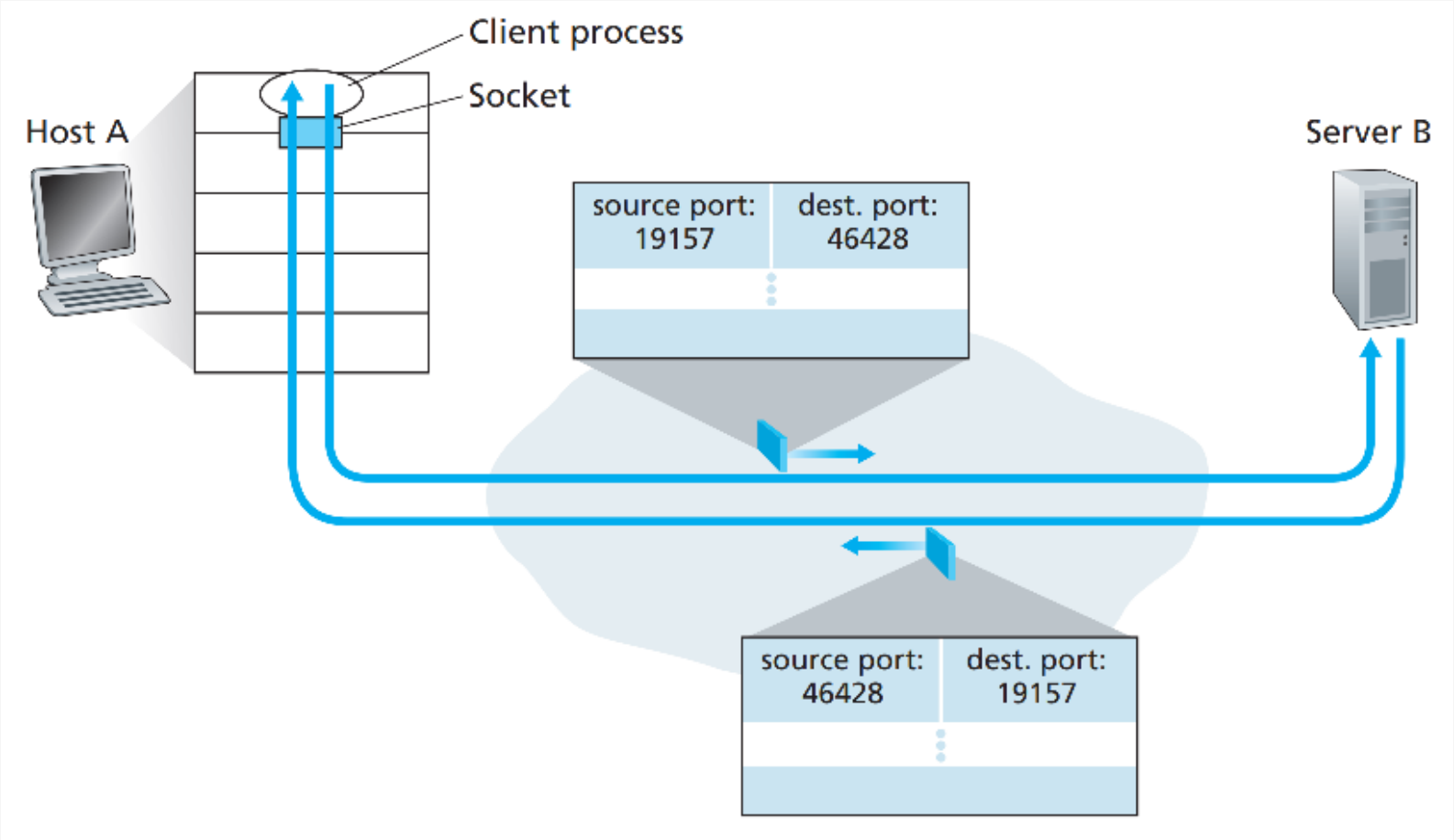


Figure: UDP Socket Identifier

TCP Socket

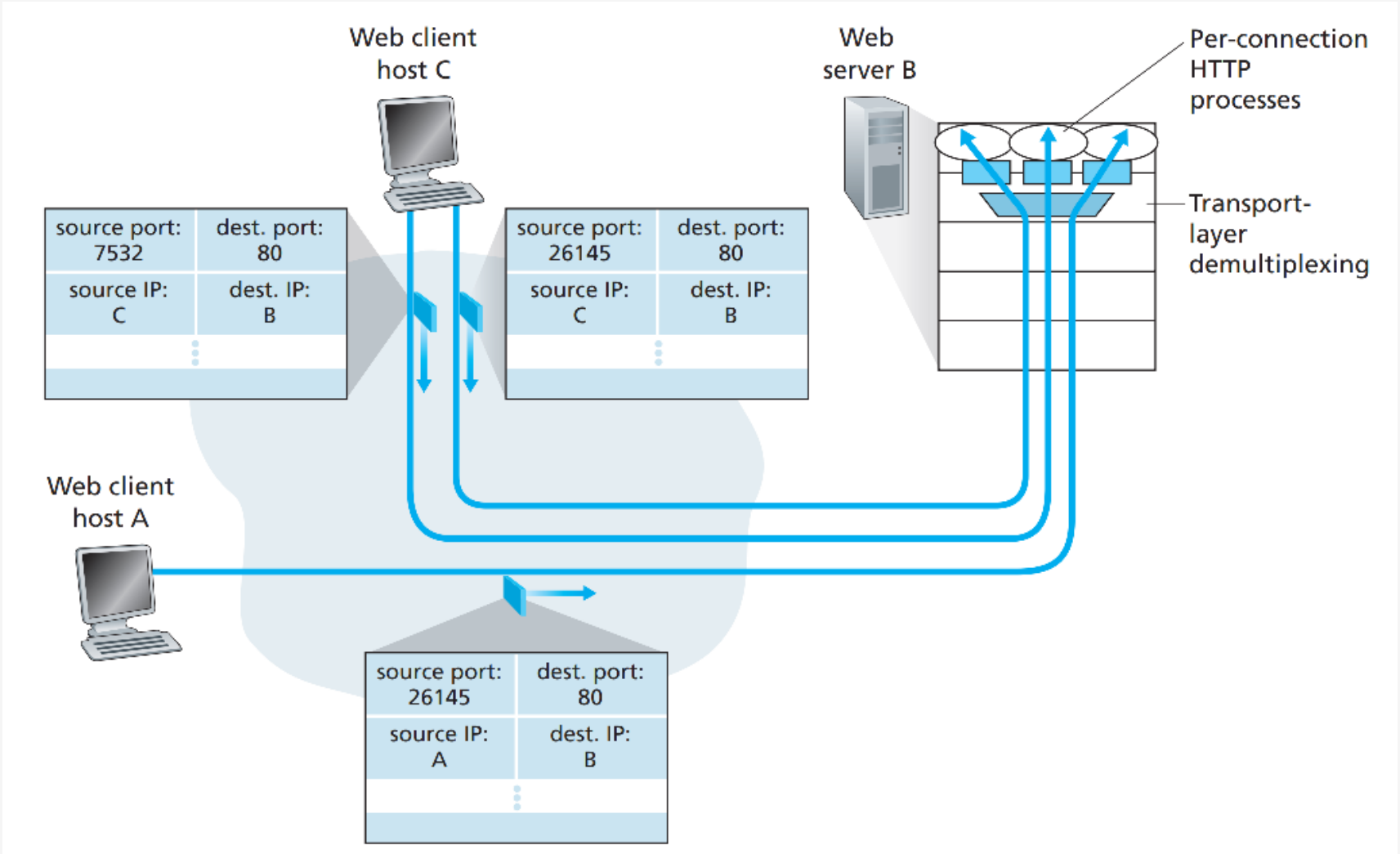


Figure: TCP Socket Identifier

Connection Less Transport Protocol: UDP

- UDP is a connectionless protocol
- UDP is an unreliable communication protocol, does not guarantee message delivery.

Why use UDP if UDP is doing nothing? Why not pass the message directly from the application layer to the network layer?

When Should We Use UDP?

- UDP is suitable for real-time applications that could tolerate packet loss. Minimum packet header overhead (only 8 bytes compared to TCP, which has 20 bytes)
- There is no connection state or and no need to establish the connection.
- If we are moving data and data integrity is critical, use TCP. For command and control (instructions), then use UDP.
- For data stream, if partial data loss is accepted, use UDP and then UDP if real-time application.

TCP Socket

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Name translation	DNS	Typically UDP

Figure: Popular Internet applications and their underlying transport protocols

UDP Segment Structure

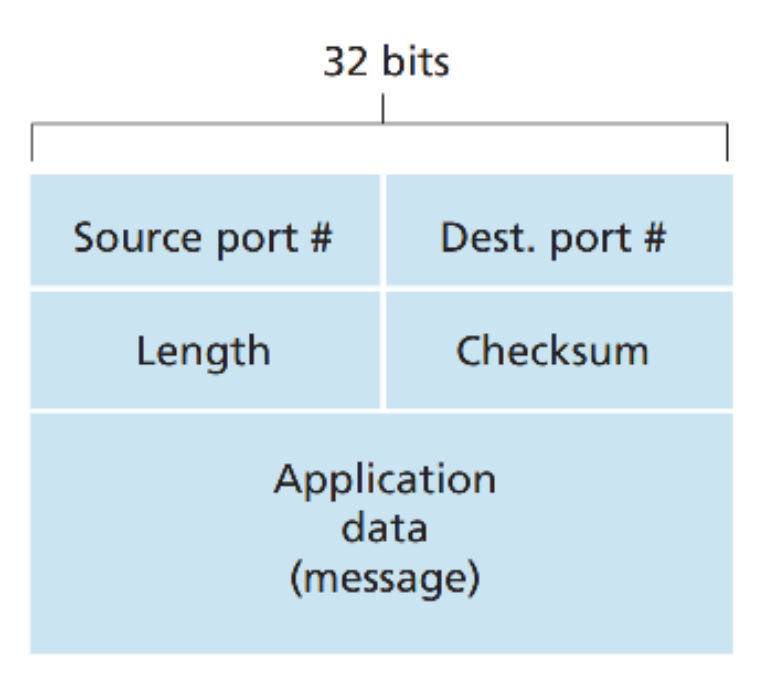


Figure: UDP segment structure

UDP Segment Structure

- The UDP segment is divided into header and msg payload.
- The UDP header consists of 8 bytes. That is divided into four fields, each consisting of two bytes (16 bits).
- The four fields of the UDP header are(source port number, destination port number, length, and checksum)
- The length field specifies the number of bytes in the UDP segment, including the header and data (payload) Why??. The data field size may differ from one UDP segment to the next.
- The checksum filed is provided for error detection. But UDP does nothing to recover from error.

Reliable Data Transfer

What do we mean by reliable data transfer?

- Data are guaranteed to arrive at the destination (no data loss)
- Data integrity is guaranteed (no transferred data bits are corrupted like flipped from 0 to 1, or vice versa)
- All messages (segments) are delivered in the order in which they were sent.

Why reliable data transfer is a challenge?

The main challenge is the underlying infrastructure is unreliable (zero guarantees)

Reliable Data Transfer

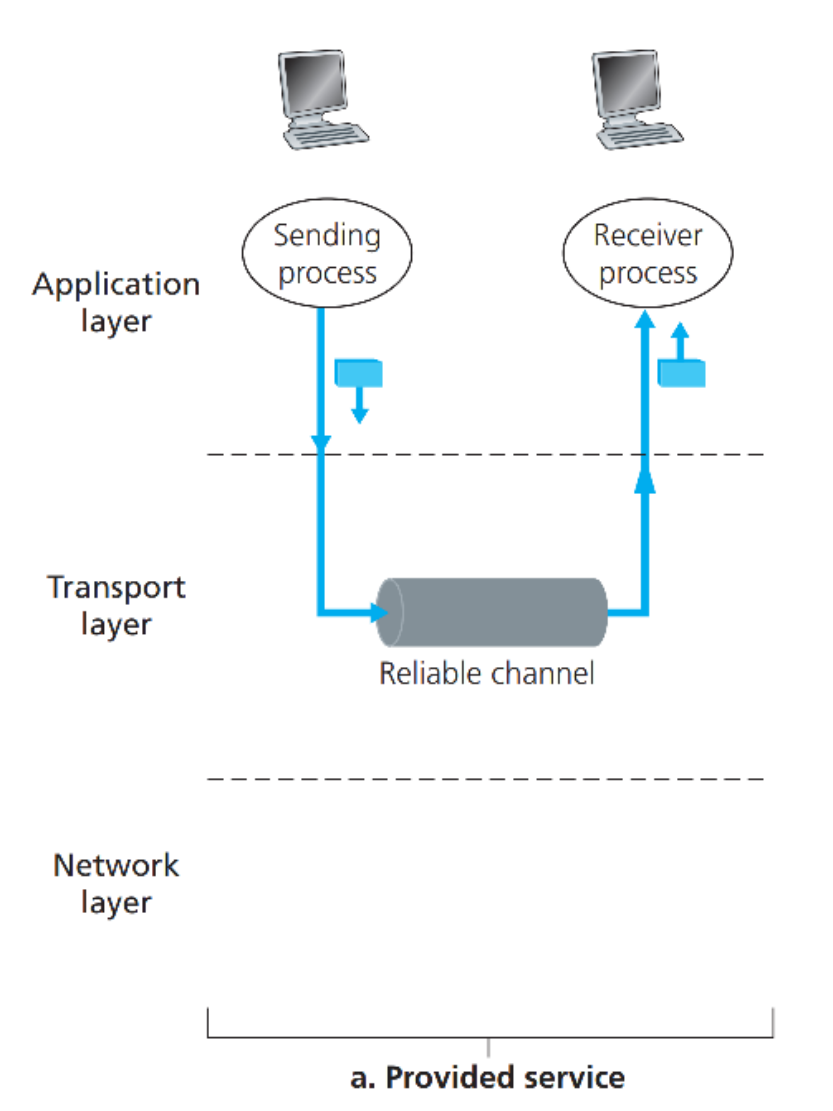


Figure: Reliable data transfer: Service model

Reliable Data Transfer

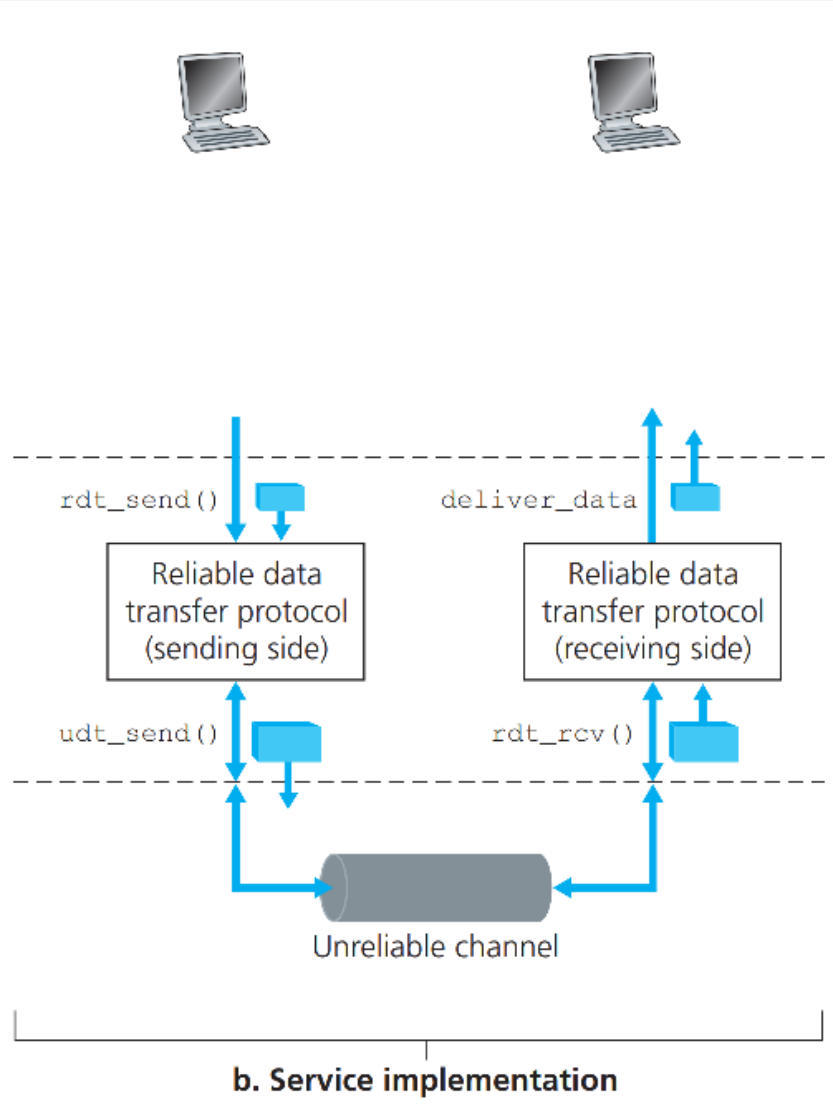


Figure: Reliable data transfer: Service Implementation

Building Reliable Data Transfer

Design Strategy

To explain how we could design a reliable data transfer protocol over unreliable infrastructure, we will start with a simple design and unrealistic assumptions and evolve our design by adding complexity and realistic assumptions a few at a time.

RDT v1.0

Reliable Data Transfer (RDT) over a Perfectly Reliable Channel

Assumptions

1. Reliable channel
2. Data is sent in one direction (unidirectional data transfer)
3. No need for feedback
4. We can send as many messages as we wish as fast as we can.

RDT v1.0

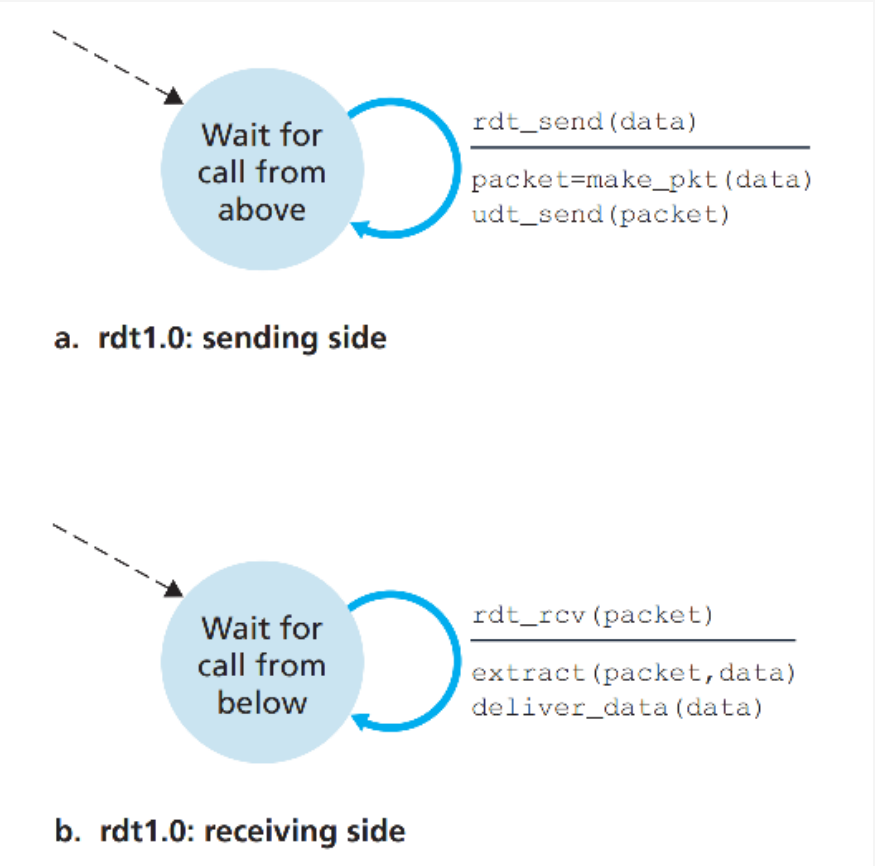


Figure: A protocol for a completely reliable channel

RDT v2.0

Reliable Data Transfer over a Channel with Bit Errors

Assumptions

1. Message corruption is possible (bits are corrupted like flipped from 0 to 1, or vice versa)
2. No packet or messages lose
3. No out-of-order packets or messages.

RDT v2.0

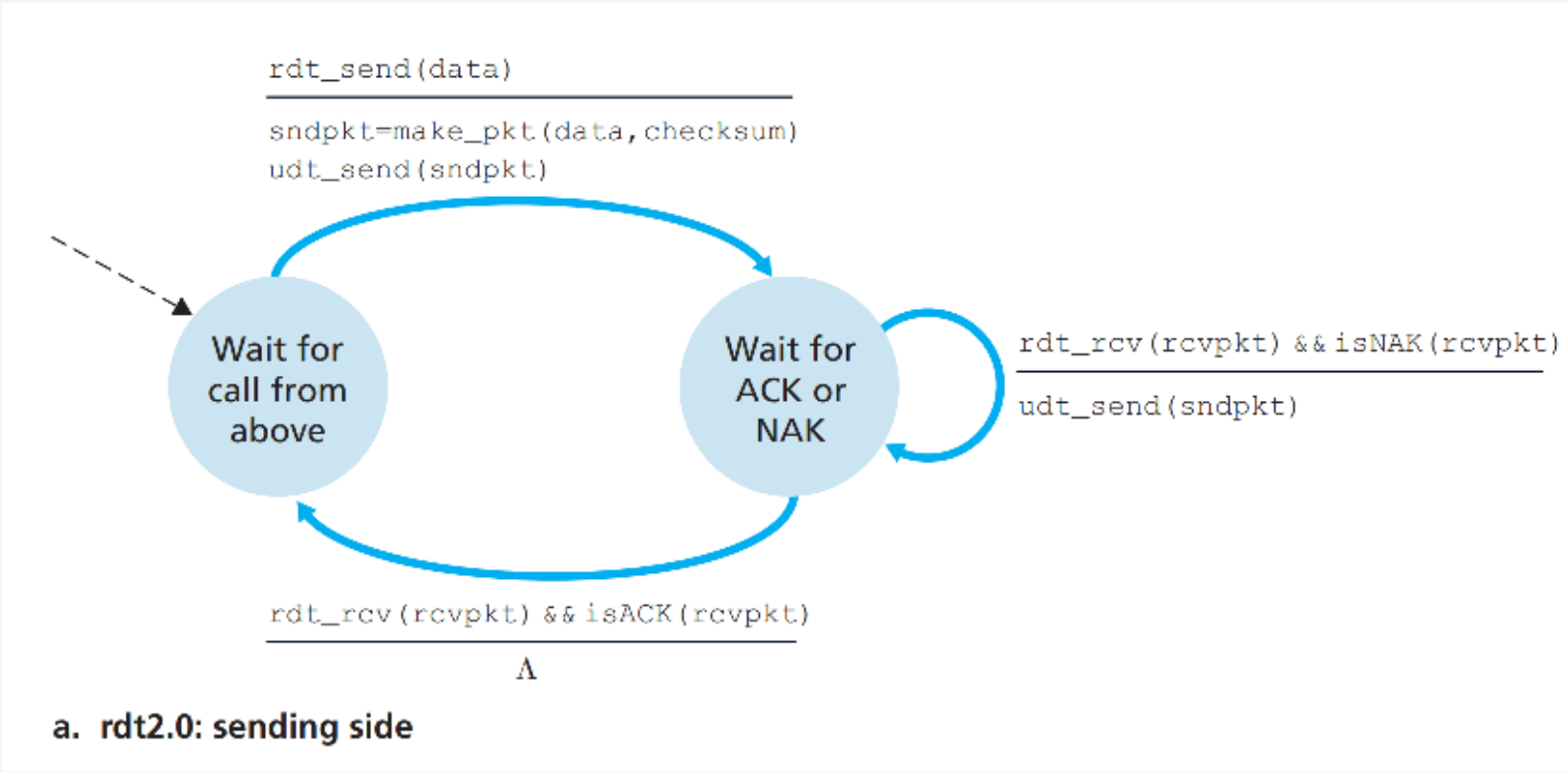
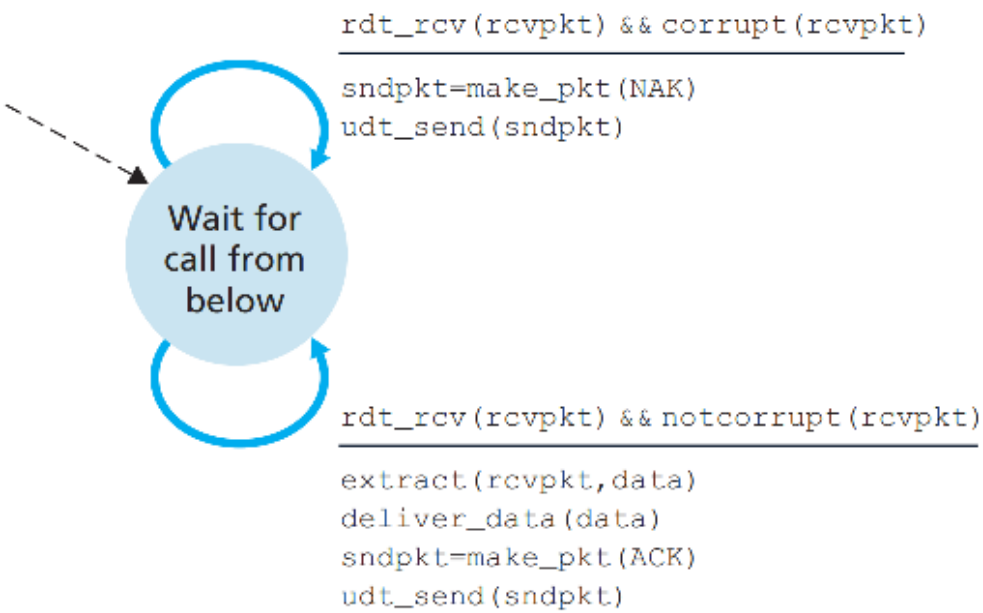


Figure: rdt2.0: sending side

RDT v2.0



b. rdt2.0: receiving side

Figure: rdt2.0: receiving side

RDT v2.0 Behaviours

- The protocol, in this case, requires feedback from the receiver side
- Positive feedback from the receiver indicates that the message's integrity is preserved.
- Negative feedback from the receiver indicates receiving a corrupted message. On negative acknowledgment, the sender will resend the message (repeat until) and wait for positive feedback.
- The sender needs a local buffer to store a copy of the message until it receives positive feedback from the receiver.
- The sender can not send any new message until it receives the positive acknowledgment of the last message it sent.
- The RDT2.0 belongs to the stop-and-wait protocols family.

RDT v2.0 Design Goals

- The protocol provides reliable data transfer against bit errors using the retransmission approach, ARQ (Automatic Repeat reQuest) protocol.
- The sender can not send any new message until it receives the positive acknowledgment of the last message it sent.
- The protocol can handle bit errors by implementing an error detection mechanism, uses feedback, and enable retransmission on failure.

What is the primary design flaw of this protocol?

RDT v2.0 Design Flow

What is the primary design flaw of this protocol?

If the feedback (positive or negative acknowledgment) was corrupted during the transmission. The sender will not be able to know if the receiver received the packet or not.

Possible Solutions

- The sender asks the receiver to resend the feedback
- Adding complex checksum to enable the recovery from the bit errors
- The sender on corrupted feedback could resend the packet (e.g., assume it received negative feedback)

RDT v2.1

RDT with Sequence Number over a Channel with Bit Errors

- When the sender resents the packet after receiving corrupted feedback, we introduce a new problem: duplicate packets at the receiver side.
- To solve the duplicate packets in the sender-to-receiver channel, we need to enable the receiver to distinguish between old and new packets.
- The most common solution is to add a sequence number field to the transport protocol header section.

RDT v2.1

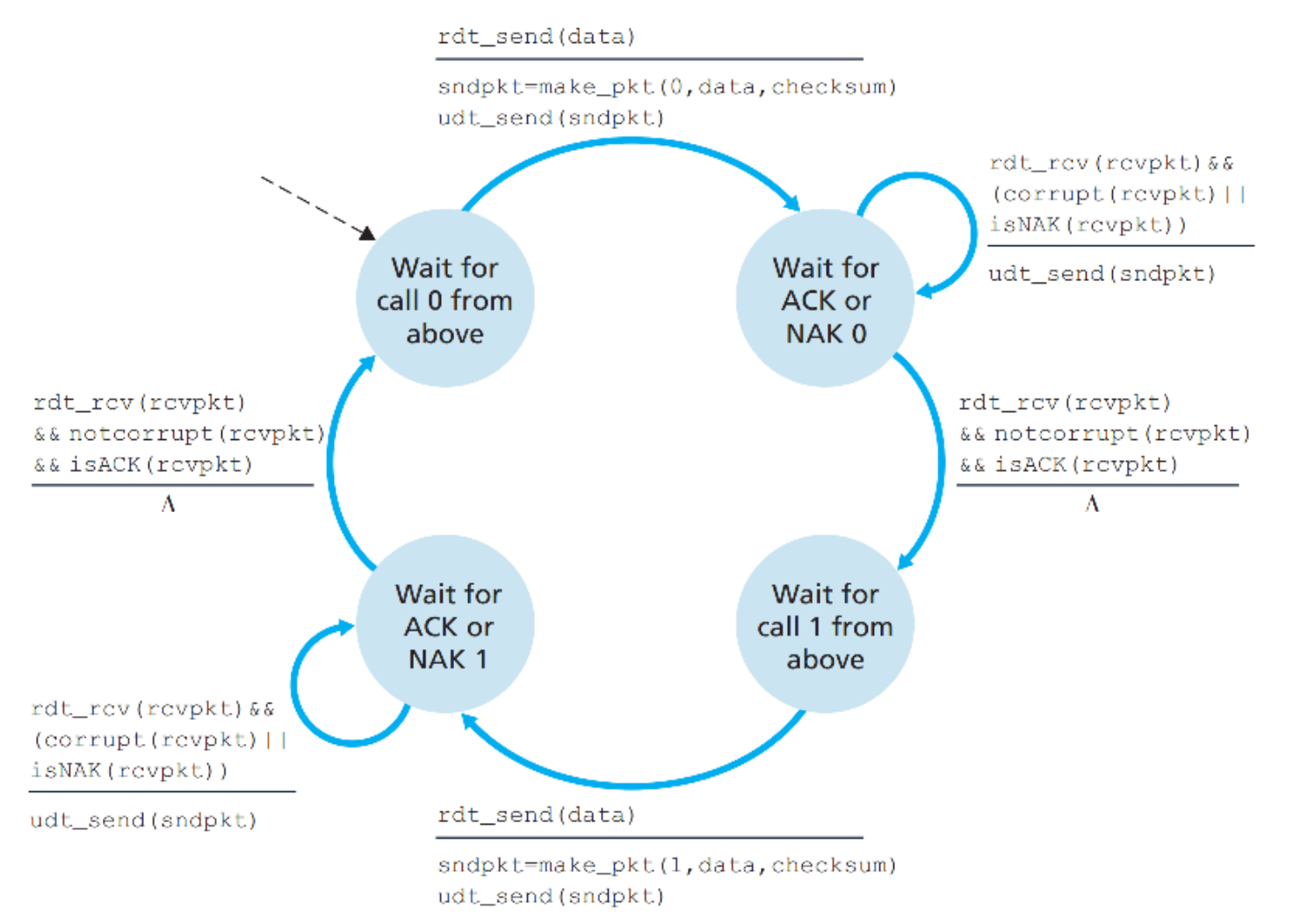


Figure: rdt2.1 sender

RDT v2.1

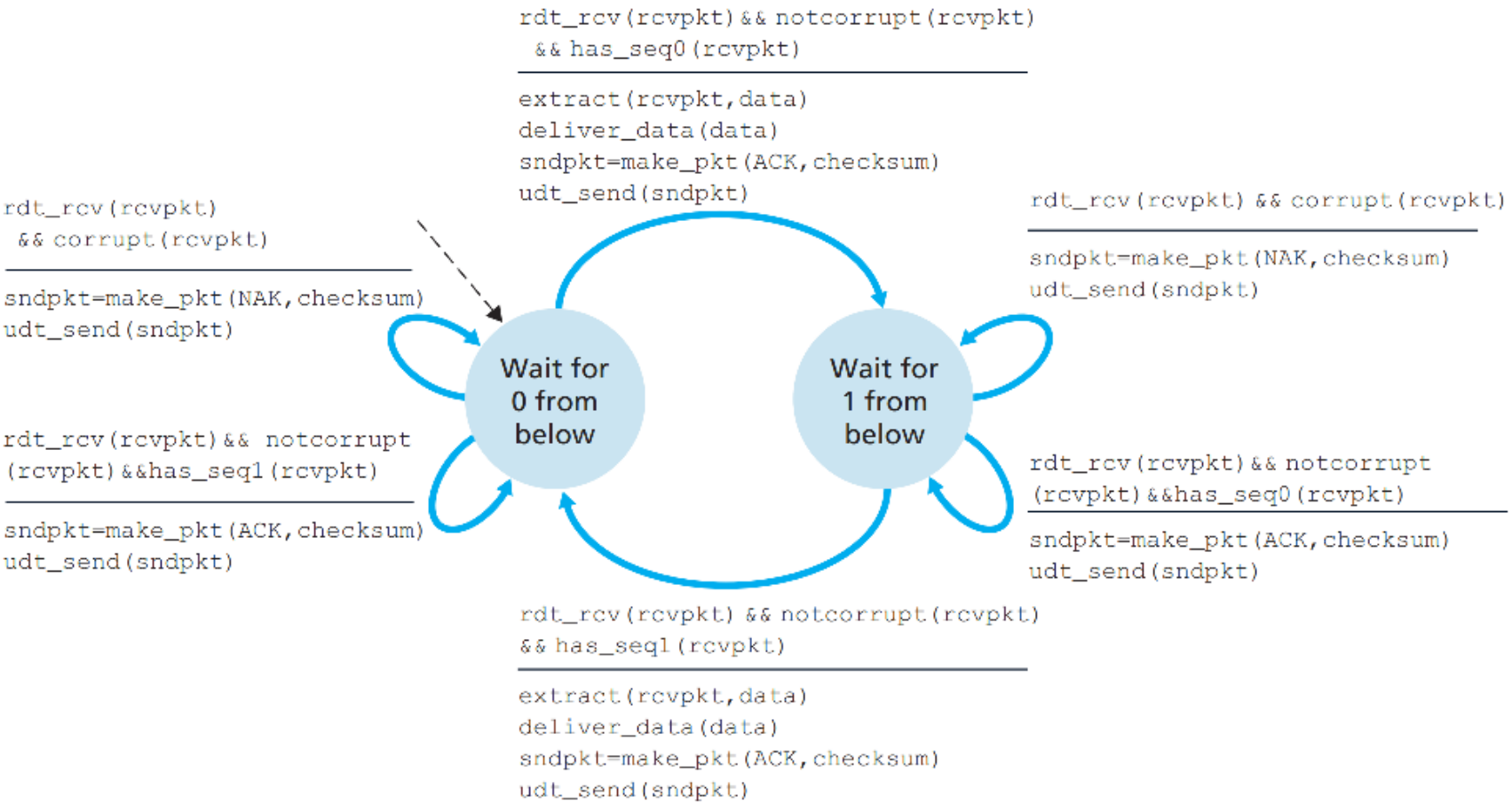


Figure: rdt2.1 receiver

RDT v2.2

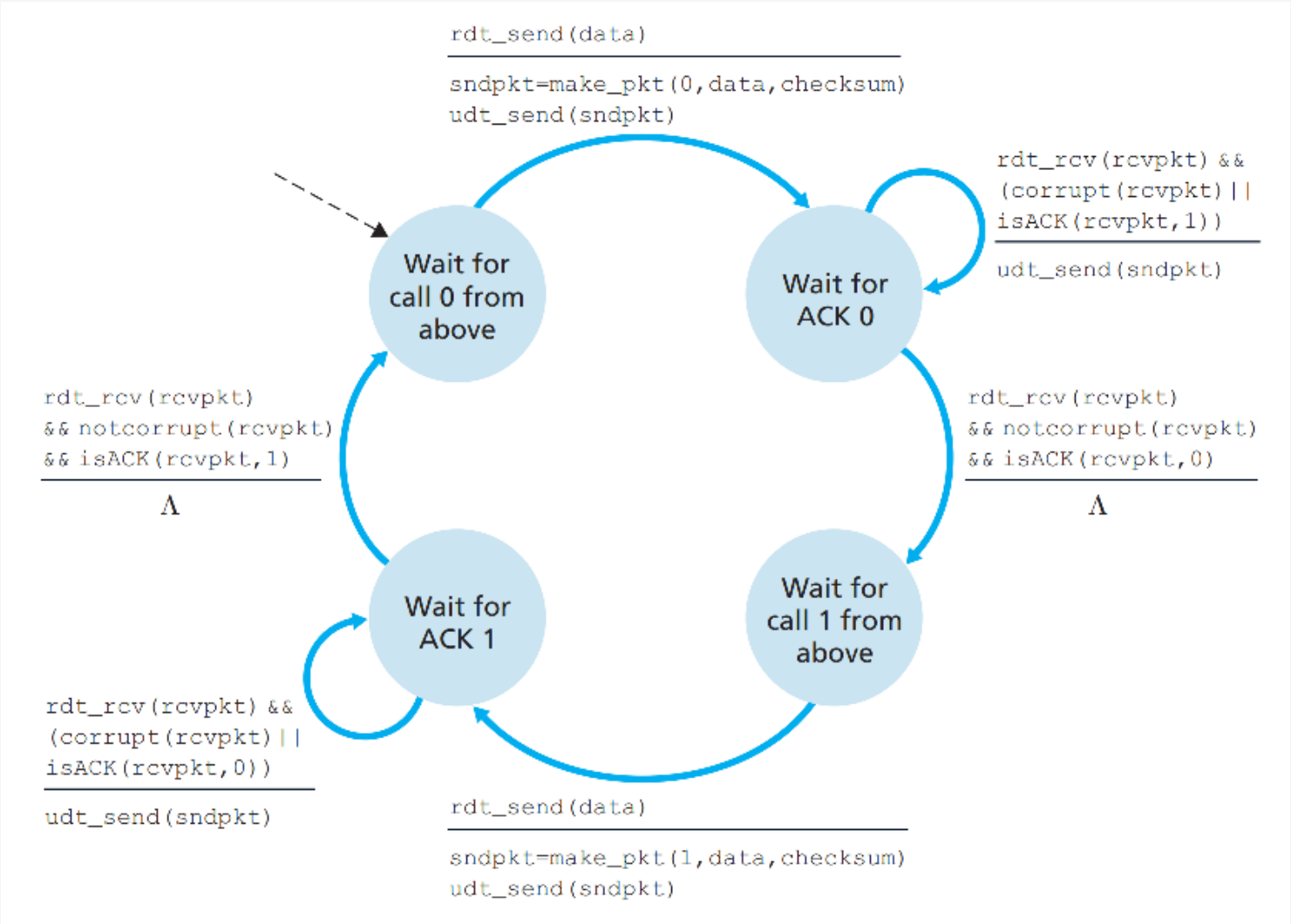


Figure: rdt2.2 sender

RDT v2.2

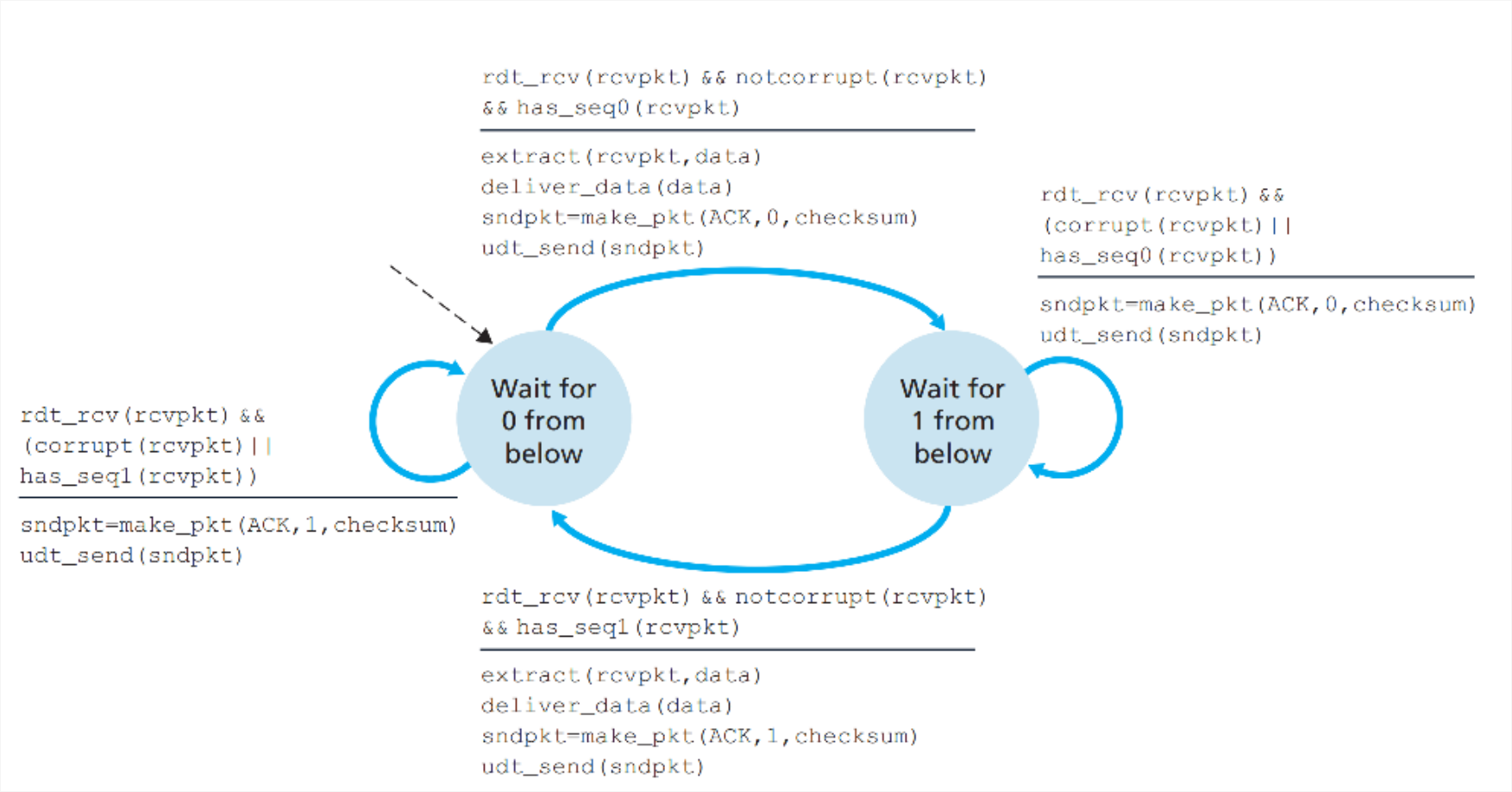


Figure: rdt2.2 receiver

RDT v3.0

Reliable Data Transfer over a Lossy Channel with Bit Errors

Assumptions

1. Assumptions for RDT 2.0
2. The massages|packets or the acknowledgment might get lost
3. The sender is responsible for detecting lost packets and resolving them using retransmission.

Detecting Packet Loss

How could the sender detect packet loss?

- There is no way the sender could be sure that the packet is lost and will never arrive at the receiver.
- There is no way the sender could be sure whether the packet or the acknowledgment is lost.

The solution is to implement a time-based retransmission mechanism using a count-down timer.

RDT v3.0 with Timer

How to use the count-down timer?

1. Choose a value for the countdown timer
2. Start the timer each time a packet is sent.
3. Respond to a timer interrupt.
4. Stop the timer.

How to choose the value for the count-down timer?

What could be the effect of premature timeout?

RDT v3.0

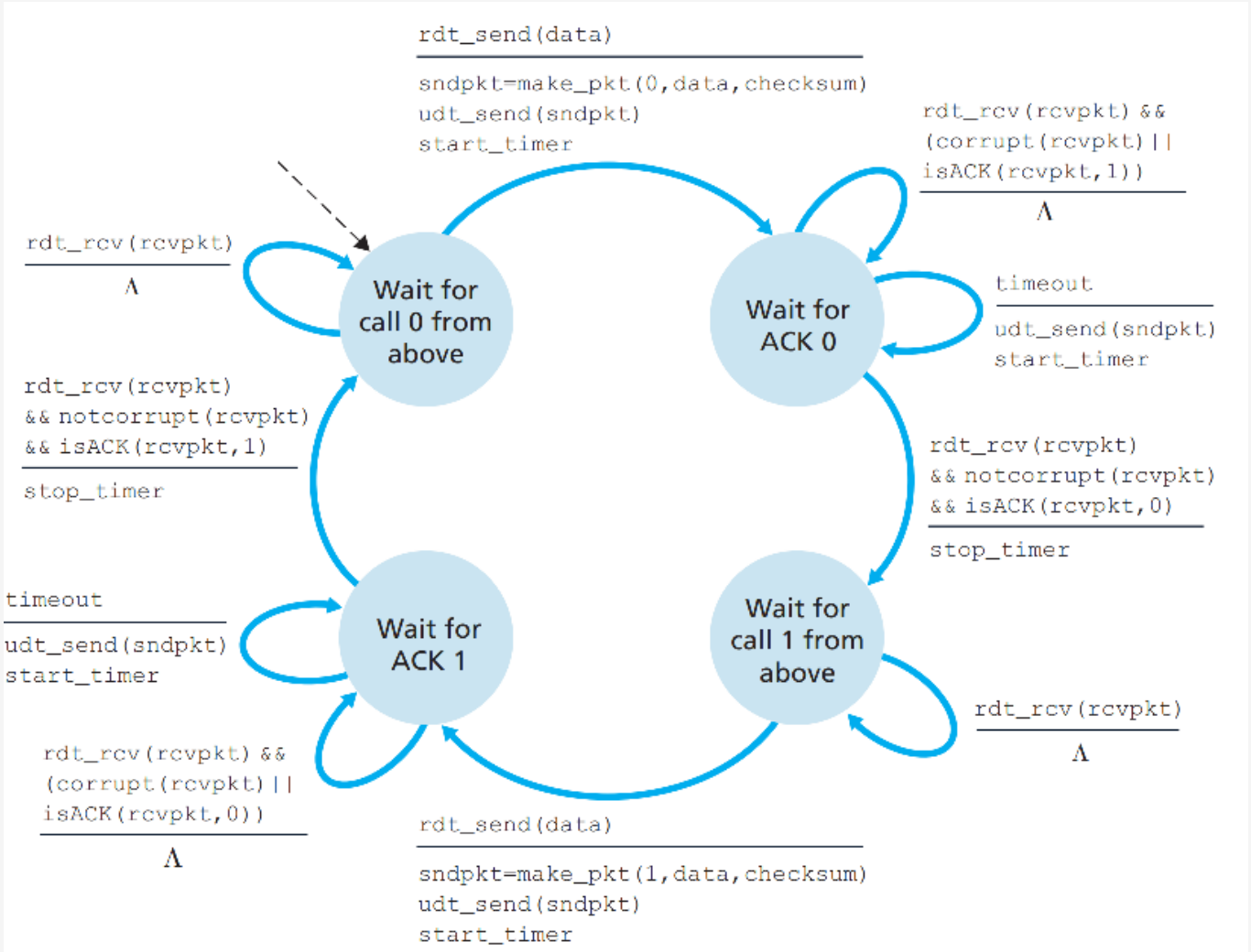


Figure: rdt3.0 sender

RDT v3.0

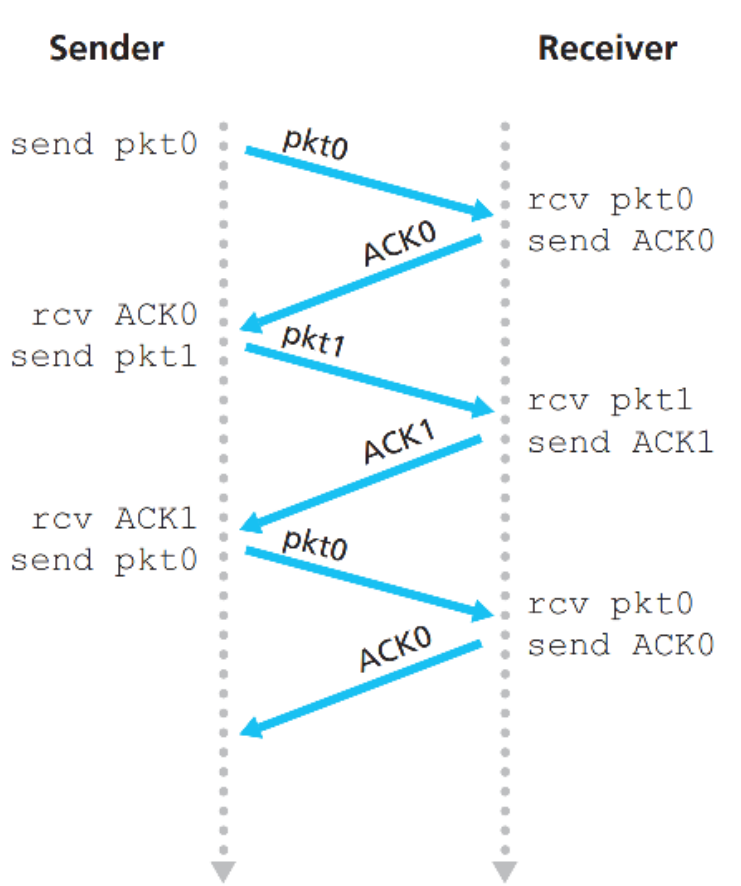


Figure: rdt3.0 Normal Operations

RDT v3.0

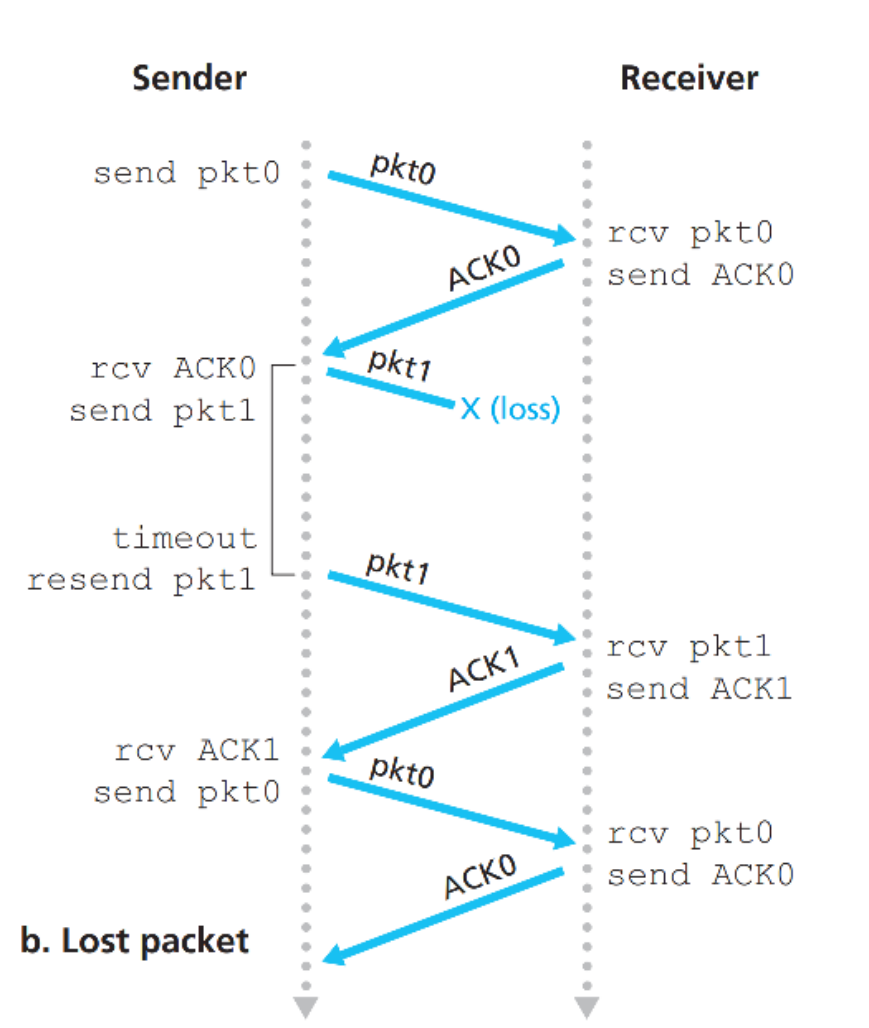


Figure: rdt3.0 Packet Loss

RDT v3.0

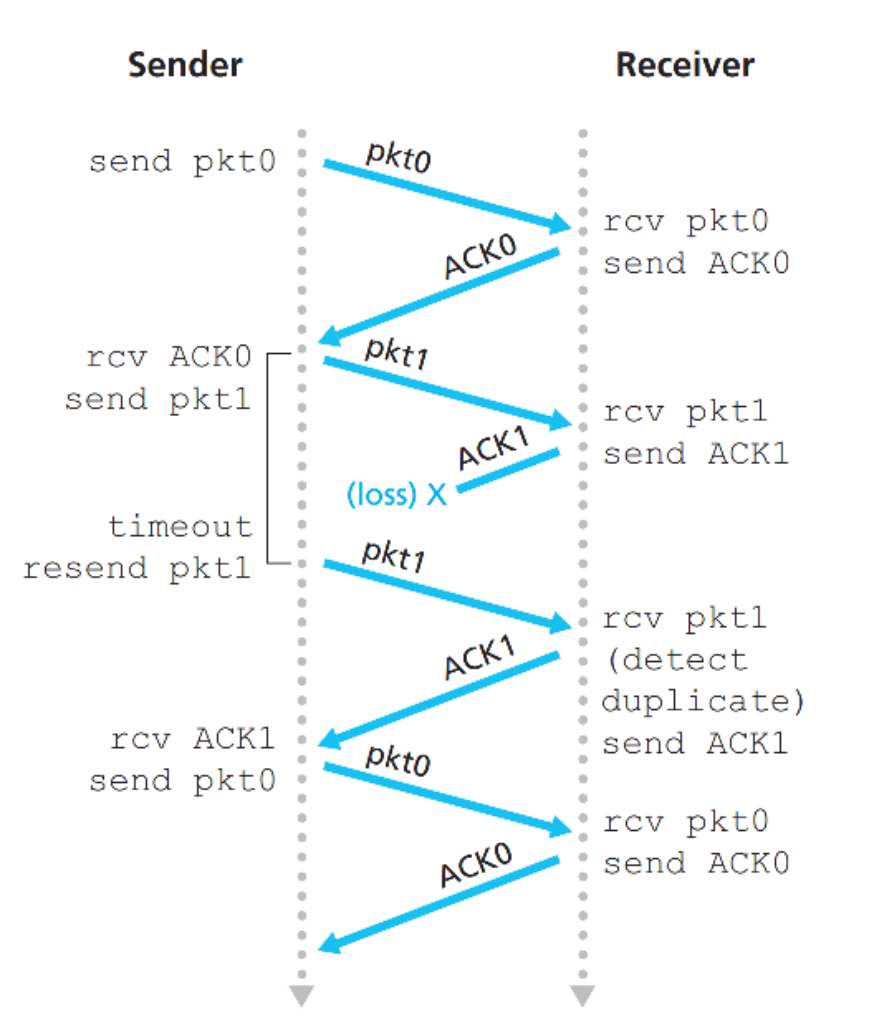


Figure: rdt3.0 Lost ACK

RDT v3.0

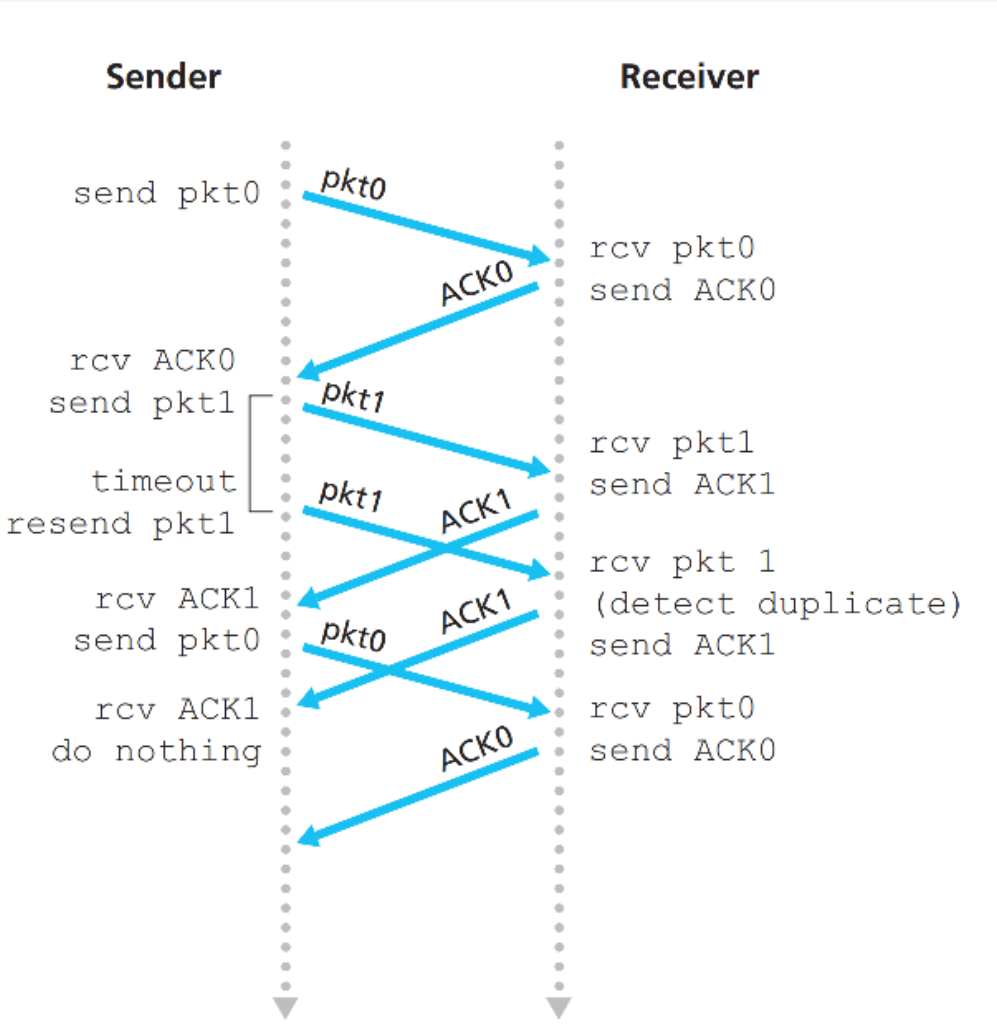


Figure: rdt3.0 Premature timeout

Pipelined Reliable Data Transfer Protocols

- RDF v3.0 is a reliable data transfer protocol. However, it is an inefficient protocol. This is because it uses a stop-and-wait pattern.
- RDF 3.0 is sometimes known as the alternating-bit protocol. Because packet sequence numbers alternate between 0 and 1.
- A better solution is to use a pipelining pattern instead of the stop-and-wait.
- With pipelining, the sender is allowed to send multiple packets without waiting for acknowledgments.

What is needed to enable pipelining?

Transport Control Protocol

- Pipelined Reliable Data Transfer Protocols.
- Transport Control Protocol
- Congestion Control in TCP

The End (Questions?)

References I


James Forshaw, *Attacking network protocols: A hacker's guide to capture, analysis, and exploitation*, 1st ed., No Starch Press, USA, 2017.


John S. Gero and Thomas Mc Neill, *An approach to the analysis of design protocols*, Design Studies **19** (1998), no. 1, 21–61.


James F. Kurose and Keith W. Ross, *Computer networking: A top-down approach*, 7 ed., Pearson, Boston, MA, 2016.


M.Rose, *Beep: Building blocks for application protocols.*, 2001.


Rose and Malamud, *Rfc3117:on the design of application protocols*, 2001.


Andrew S. Tanenbaum and Maarten van Steen, *Distributed systems: Principles and paradigms*, 2 ed., Pearson Prentice Hall, Upper Saddle River, NJ, 2007.