MAC0352 - Redes de Computadores e Sistemas Distribuídos

Daniel Macêdo Batista

IME - USP, 27 de Maio de 2021

Roteiro

RDTs

RDTs

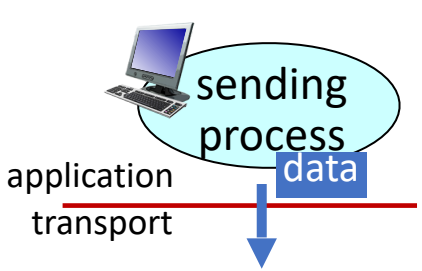
RDTs

Principles of reliable data transfer

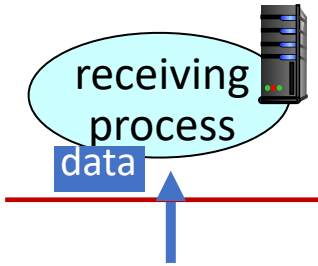
reliable service *abstraction*

Principles of reliable data transfer

reliable service *abstraction*



sender-side of reliable data transfer protocol

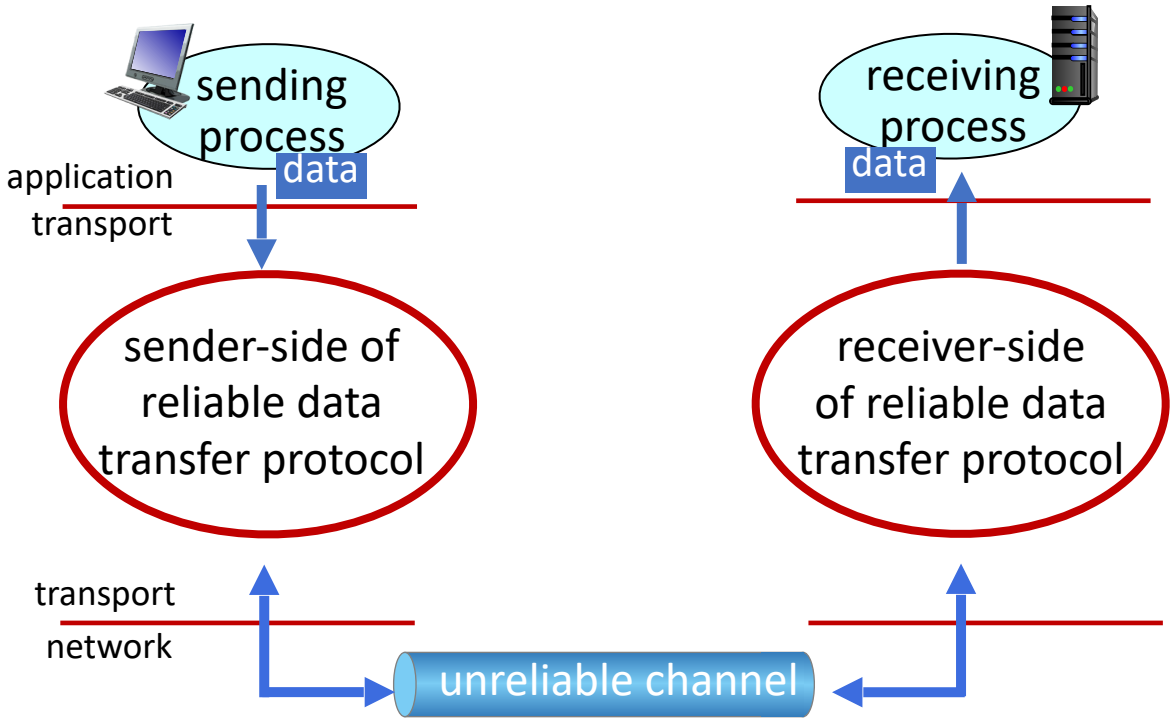


receiver-side of reliable data transfer protocol

reliable service *implementation*

Principles of reliable data transfer

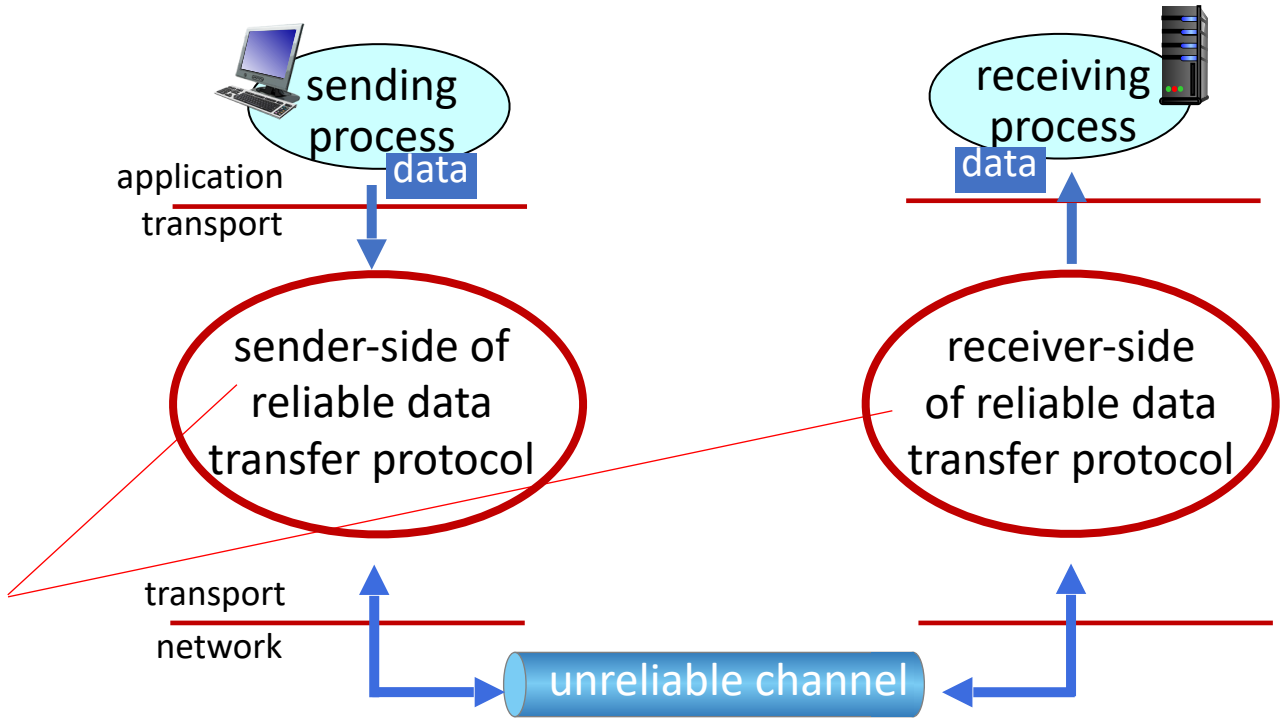
reliable service *abstraction*



reliable service *implementation*

Principles of reliable data transfer

Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)

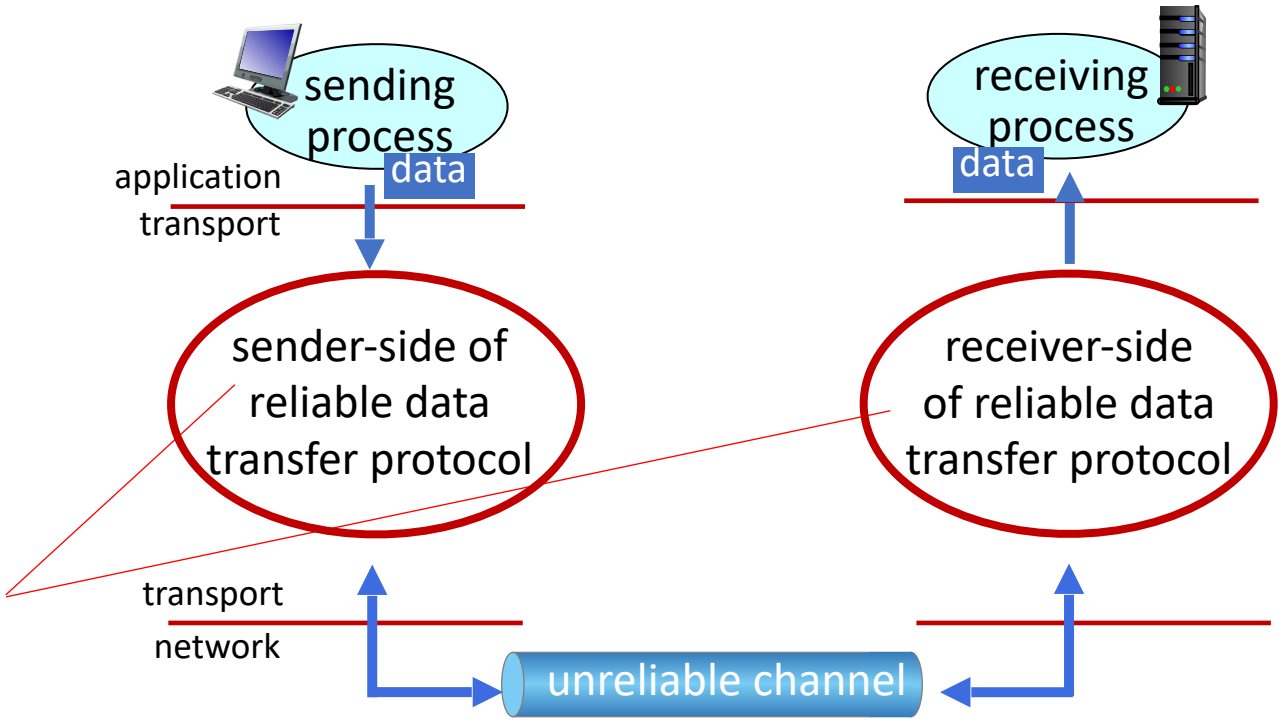


reliable service *implementation*

Principles of reliable data transfer

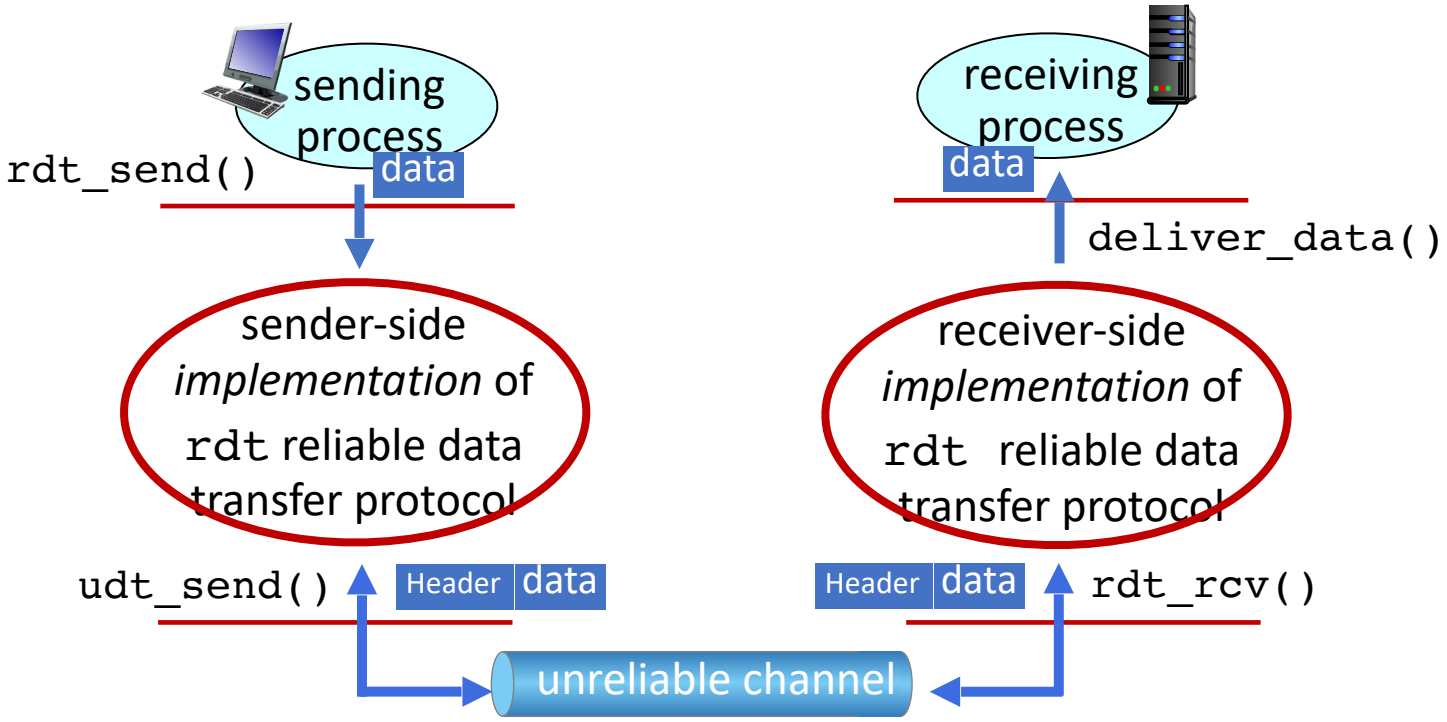
Sender, receiver do *not* know the "state" of each other, e.g., was a message received?

- unless communicated via a message



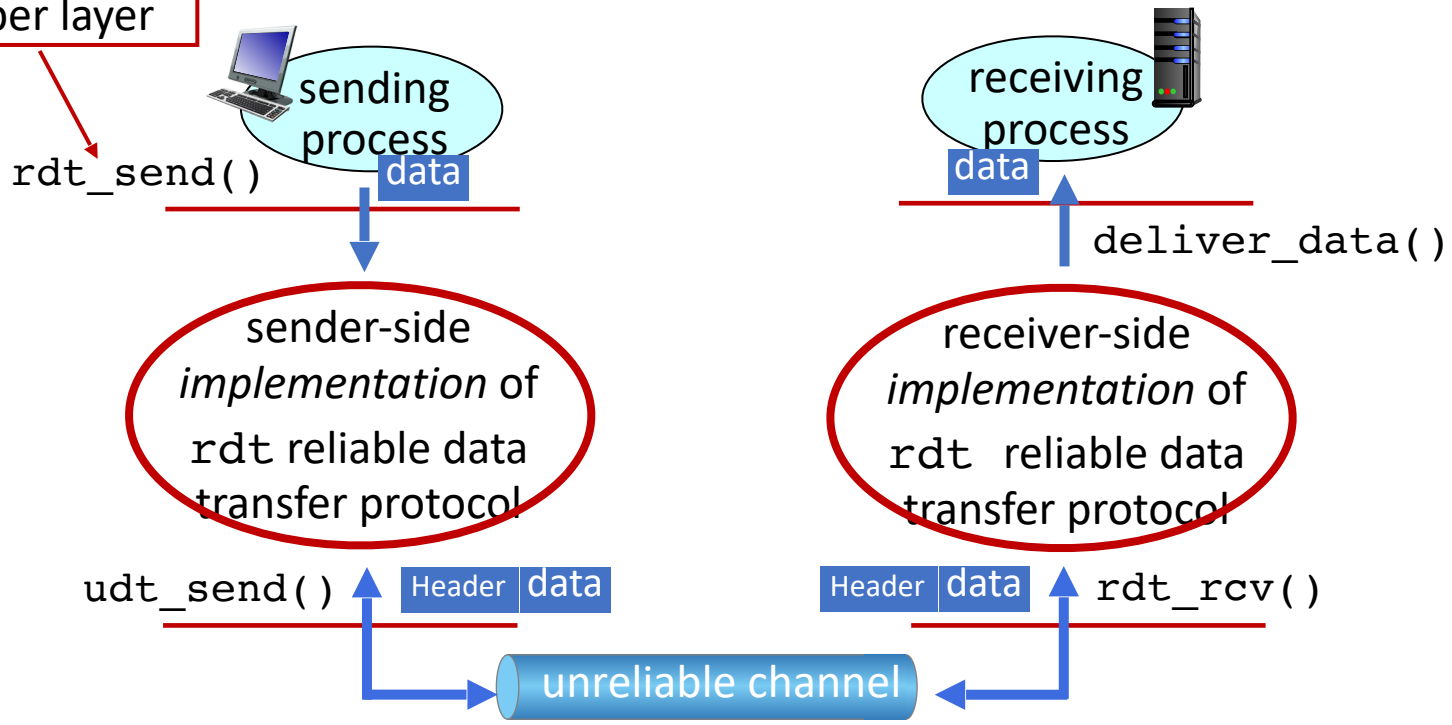
reliable service *implementation*

Reliable data transfer protocol (rdt): interfaces



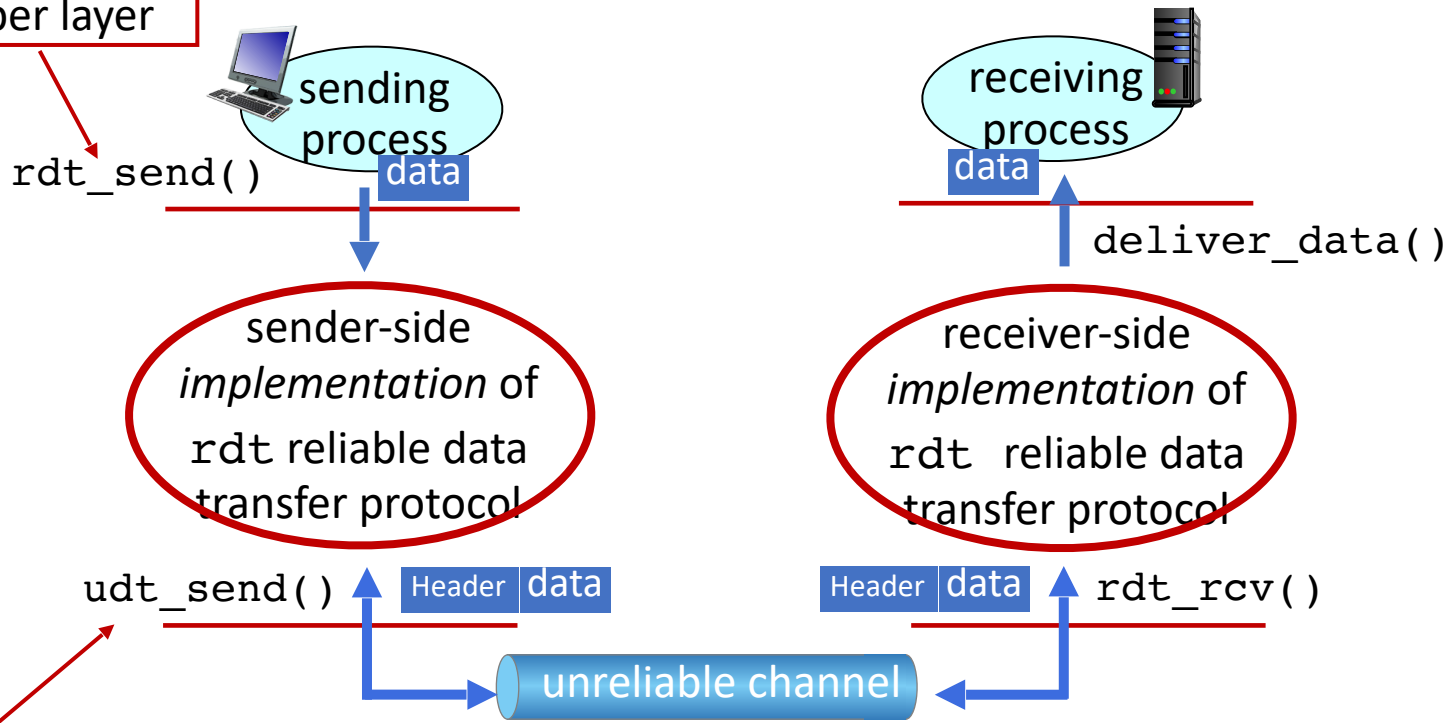
Reliable data transfer protocol (rdt): interfaces

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer



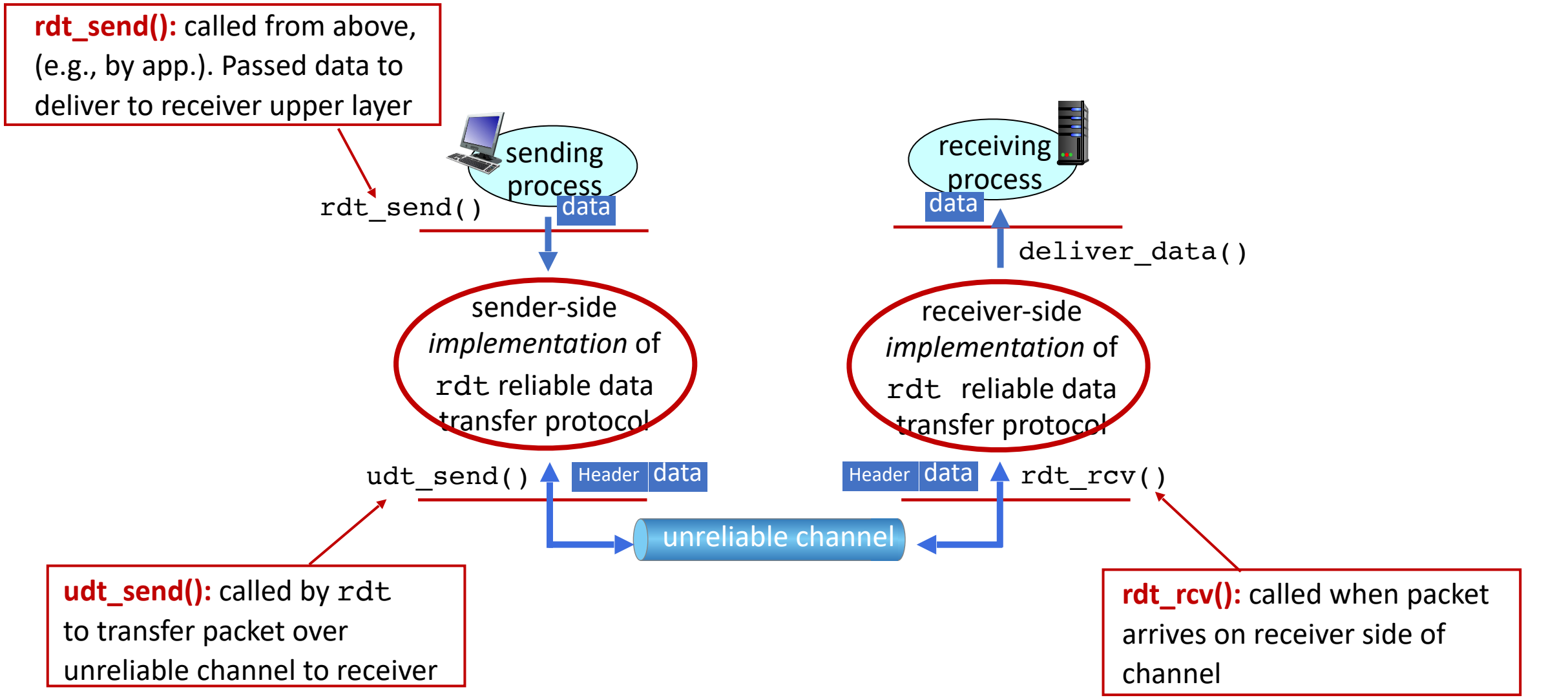
Reliable data transfer protocol (rdt): interfaces

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

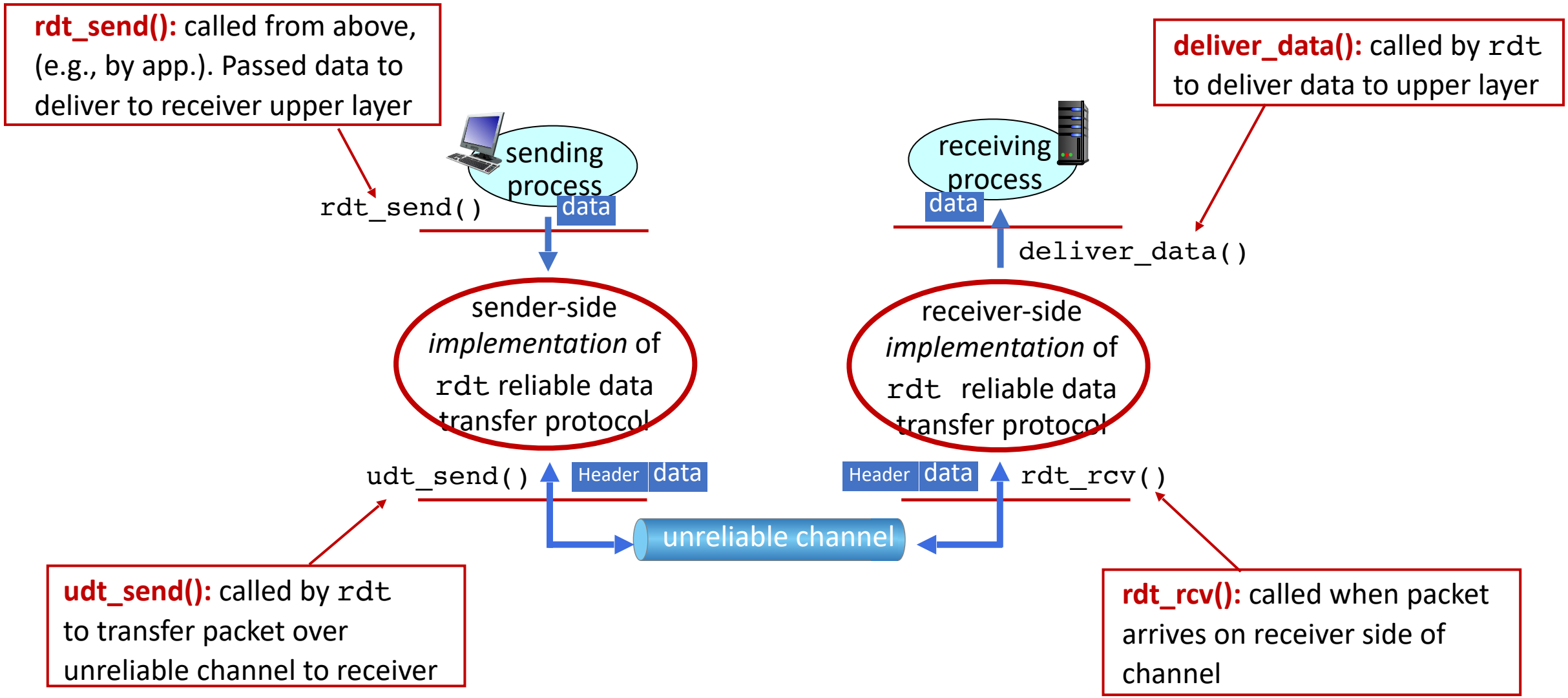


udt_send(): called by rdt to transfer packet over unreliable channel to receiver

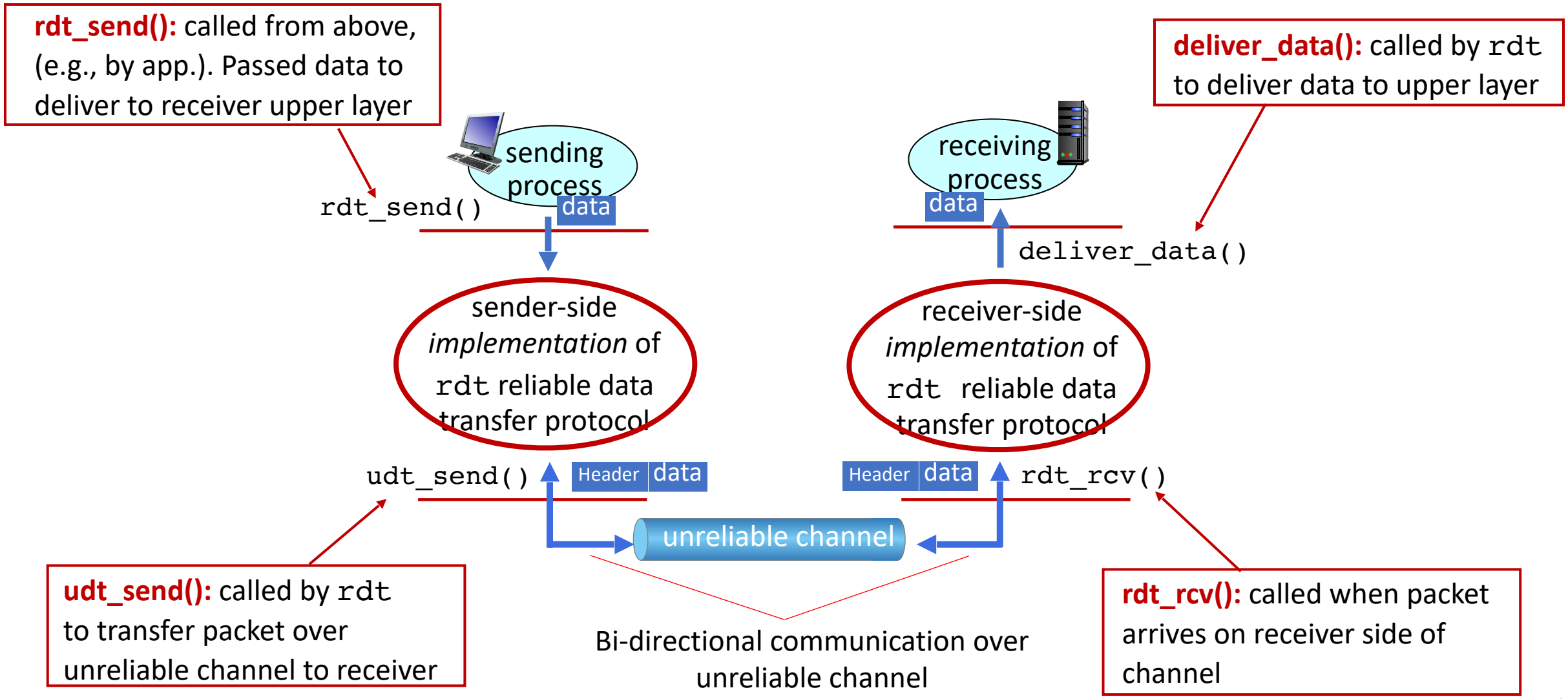
Reliable data transfer protocol (rdt): interfaces



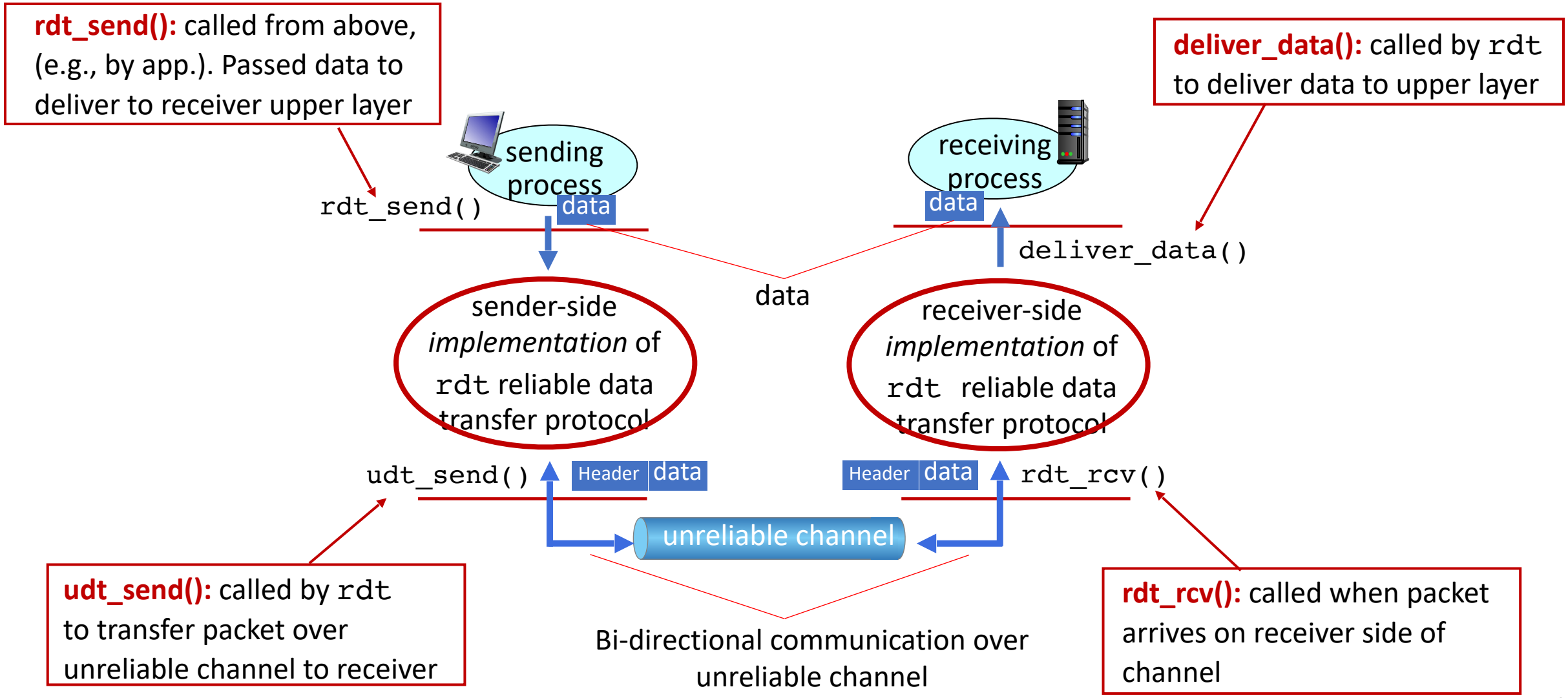
Reliable data transfer protocol (rdt): interfaces



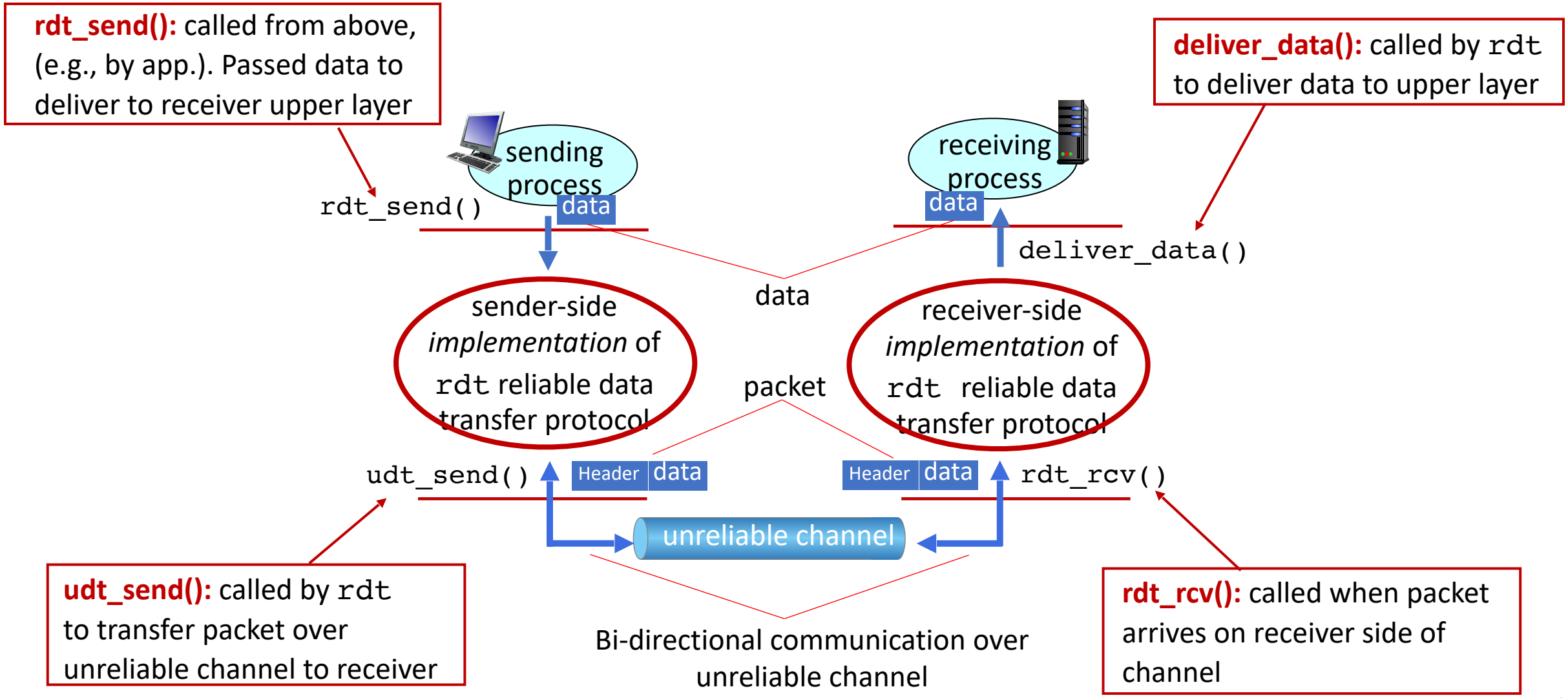
Reliable data transfer protocol (rdt): interfaces



Reliable data transfer protocol (rdt): interfaces



Reliable data transfer protocol (rdt): interfaces

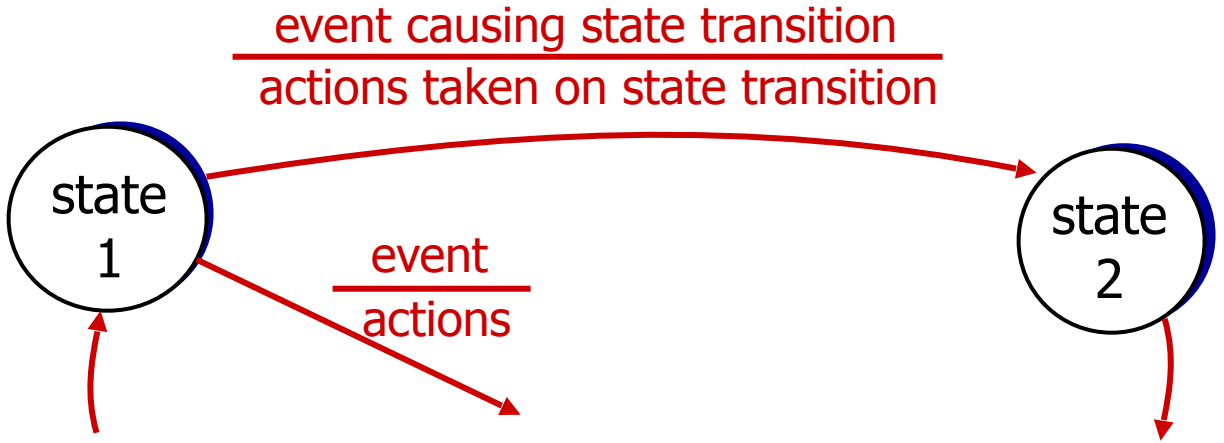


Reliable data transfer: getting started

We will:

- incrementally develop sender, receiver sides of reliable data transfer protocol (`rdt`)
- consider only unidirectional data transfer
  - but control info will flow in both directions!
- use finite state machines (FSM) to specify sender, receiver

state: when in this "state" next state uniquely determined by next event

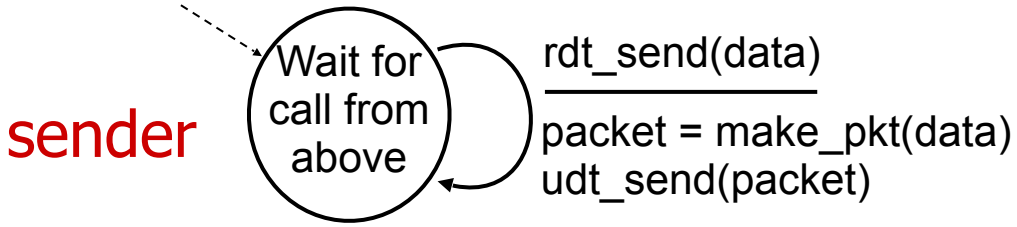


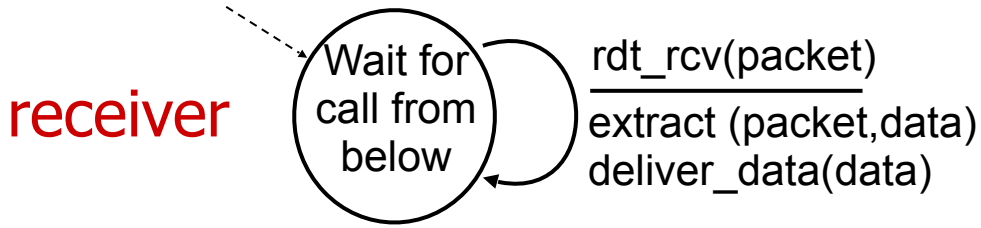
RDT 1.0

- Canal completamente confiável
  - Não há erros
  - Não há perdas
- “Igual” ao funcionamento do UDP

rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- *separate* FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel





RDT 1.0 – Problemas

- Por que não pode ser usado no mundo real?

RDT 2.0

- Canal com erros de bit
  - Todos os pacotes chegam mas algum pode chegar com conteúdo corrompido
- Como verificar que há erros?
- Como se recuperar dos erros?
  - Protocolos ARQ (Automatic Repeat reQuest - ou Query)

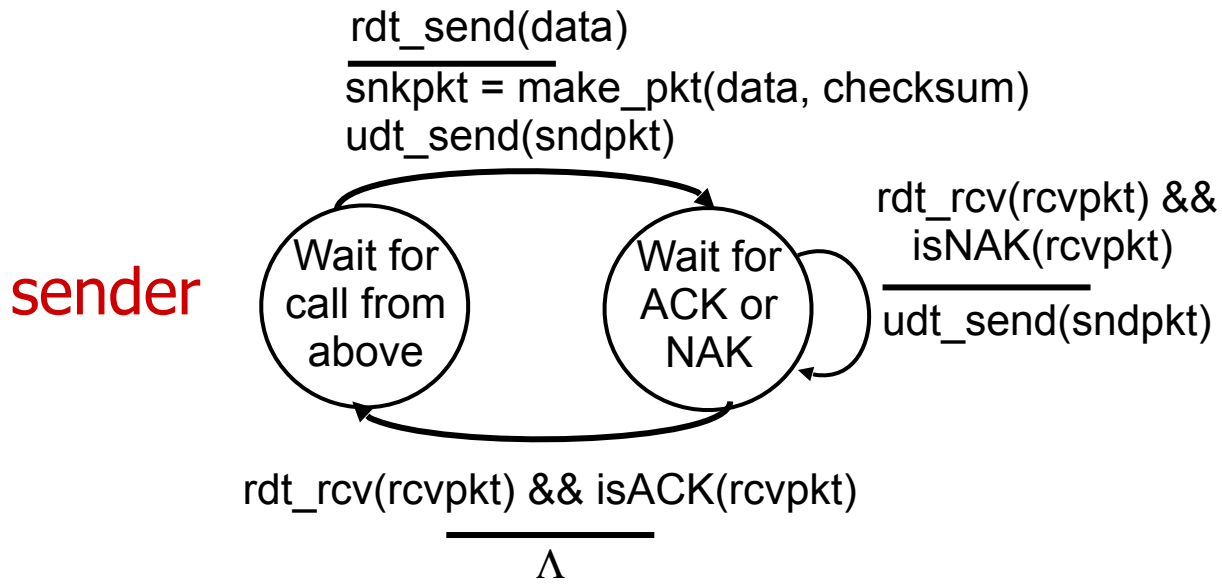
rdt2.0: channel with bit errors

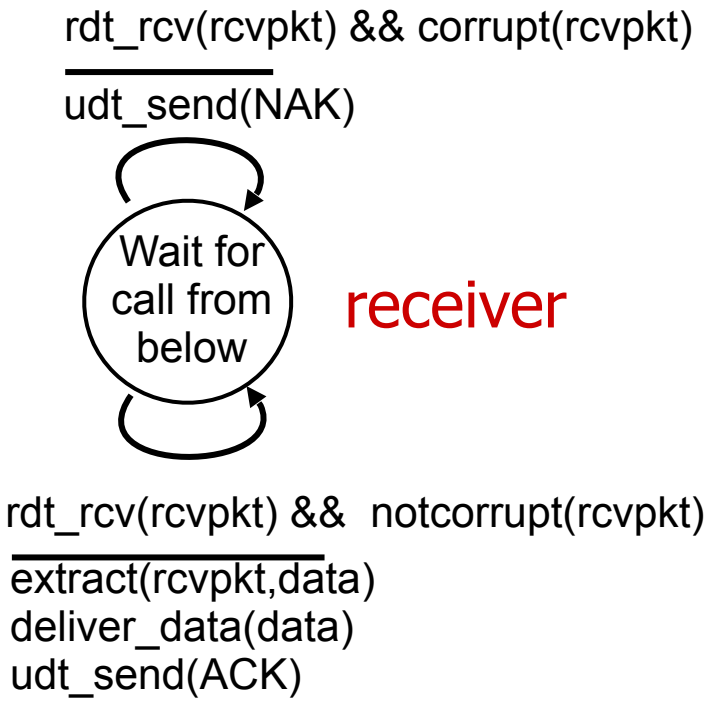
- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender *retransmits* pkt on receipt of NAK

stop and wait

sender sends one packet, then waits for receiver response

rdt2.0: FSM specification





Note: "state" of receiver (did the receiver get my message correctly?) isn't known to sender unless somehow communicated from receiver to sender

- that's why we need a protocol!

RDT 2.0 – Problemas

- Estamos tomando medidas para todos os pacotes que possam vir com erros?

RDT 2.0 – Problemas

- Estamos tomando medidas para todos os pacotes que possam vir com erros?
  - Erros no ACK/NACK precisam ser tratados: reenvia o pacote se ACK/NACK está corrompido
  - Como saber se o pacote é novo ou se é repetido?

RDT 2.0 – Problemas

- Estamos tomando medidas para todos os pacotes que possam vir com erros?
  - Erros no ACK/NACK precisam ser tratados: reenvia o pacote se ACK/NACK está corrompido
  - Como saber se o pacote é novo ou se é repetido?
    - Os pacotes precisam de identificadores (números de sequência)

RDT 2.1

- Agora vai enviar o identificador do pacote (0 ou 1)
- Por enquanto basta ser 0 ou 1 para saber se o pacote é o que foi recém-enviado ou se é um novo