Technische Universität Berlin

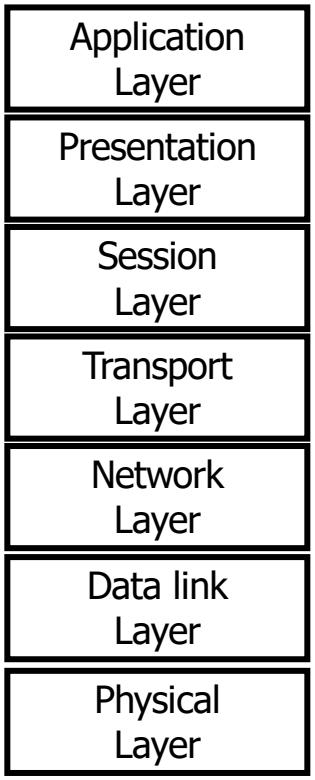
# Computer Networks

Error Control

# Chapter

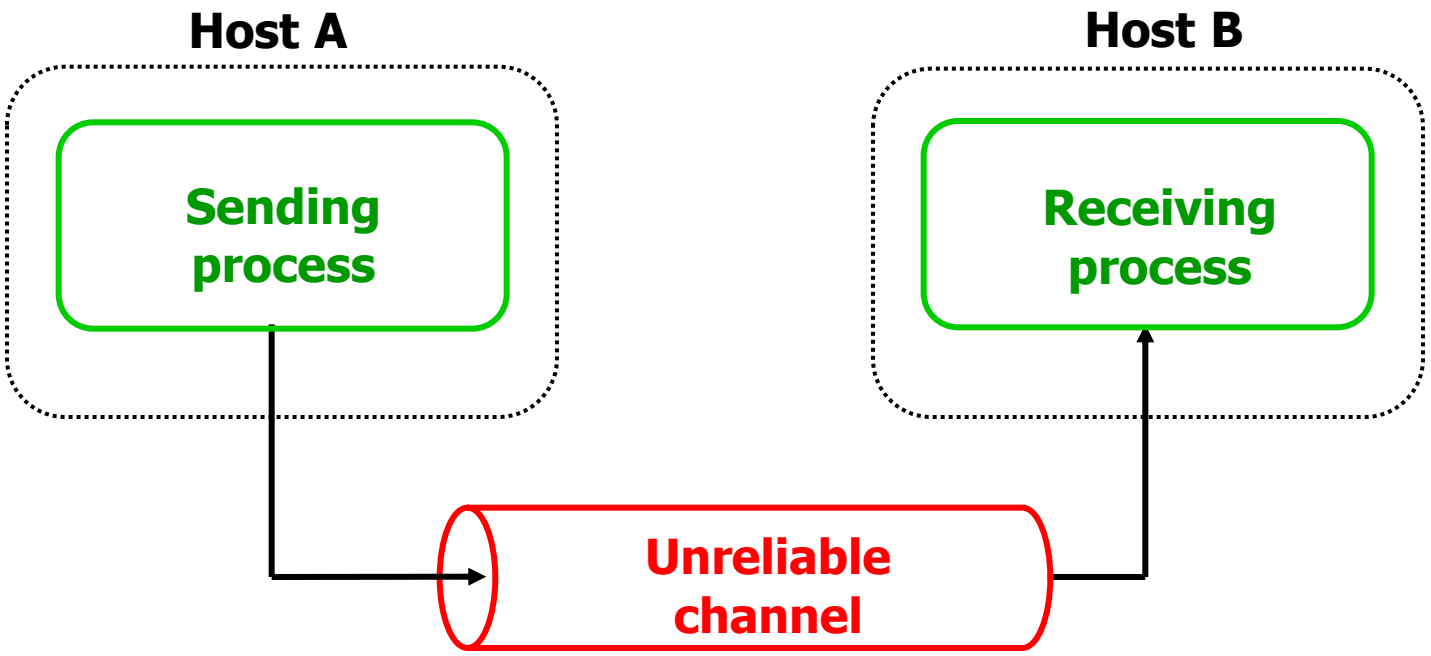
1. Introduction
2. Protocols
3. Application layer
4. Web services
5. Distributed hash tables
6. Time synchronization
7. **Error control**
   - **Stop-and-Wait**
   - **Sliding window**
   - **Performance analysis**
8. Transport layer
9. TCP performance
10. Network layer
11. Internet protocol
12. Data link layer

Top-Down-Approach



# Error control

# Error control




- Noise, interference, buffer overflow, failures of components, etc. lead to bit errors and packet loss
- This can be compensated by protocols supporting error detection, acknowledgments, and repeated transmissions

# Error control

- We study three fundamental protocols for error control
  - **Stop-and-Wait**
    - Sender adds checksum (best a cyclic redundancy check, CRC) to the packet in order to identify bit errors
    - Receiver sends acknowledgment (ACK) if packet was received w/o errors
    - If acknowledgement is not received until a deadline (timeout), the packet will be sent again
    - As duplicates can be generated, sequence numbers (SQN) are added to identify such duplicates
    - Bandwidth-delay-product has huge impact; if very large, the sender will spend most of the time waiting for acknowledgments
  - Sliding window protocols
    - Multiple packets are sent in a burst to fill the channel between sender and receiver
    - **Go-Back-N** and **Selective Repeat**
    - Protocols differ in terms of timeout, acknowledgment handling, and repeated transmissions

# Stop-and-Wait

# Stop-and-Wait

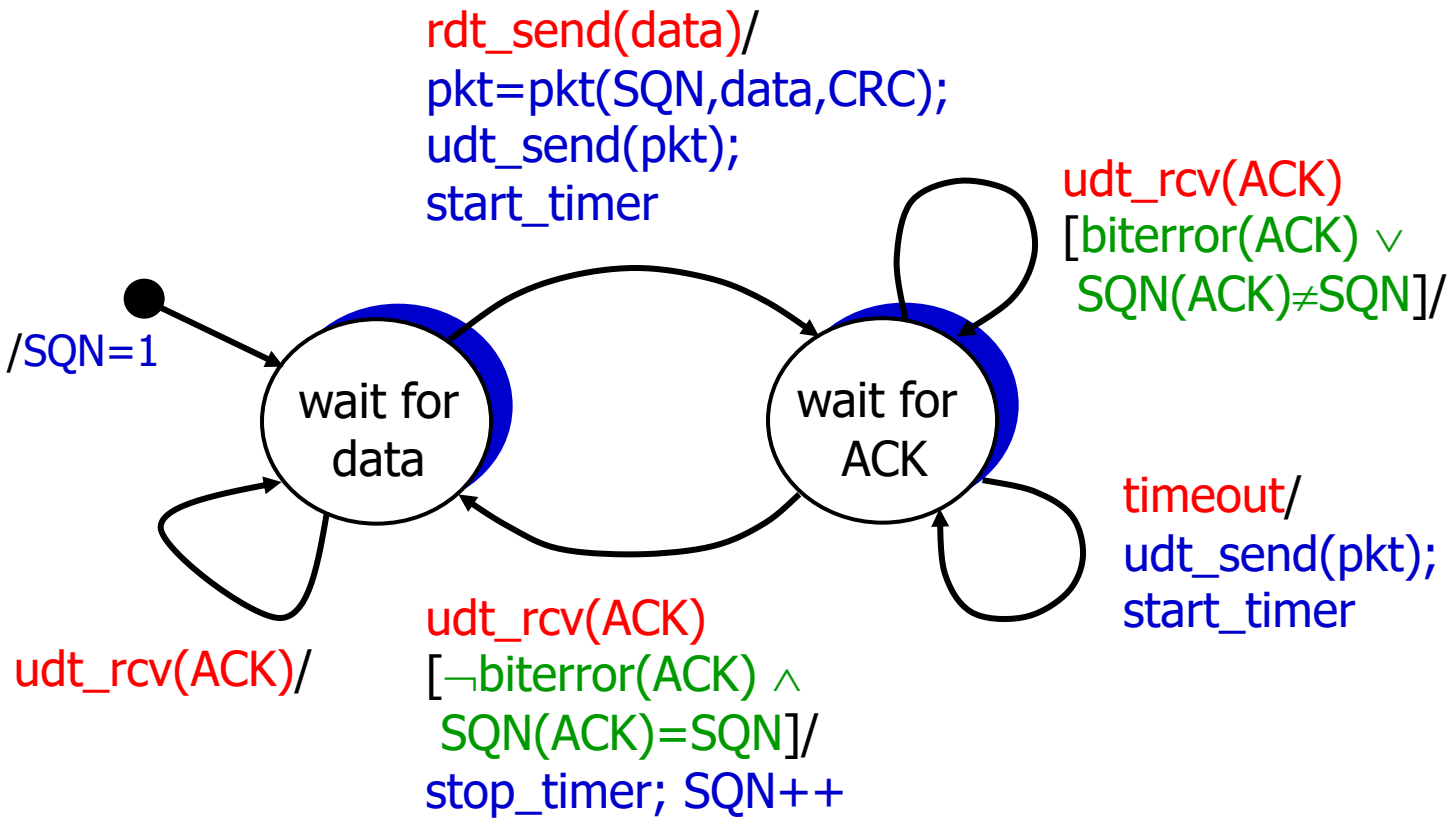
- How does stop-and-wait work?
- Informal:

  - Senders
    1. Send packet with current SQN, start timer
    2. If correct ACK is received, increment SQN and restart at step 1
    3. If timeout occurs, resend packet and restart timer

  - Receiver
    - If correct packet is received (no bit error, current SQN), send ACK with current SQN then increment SQN, otherwise send last ACK again

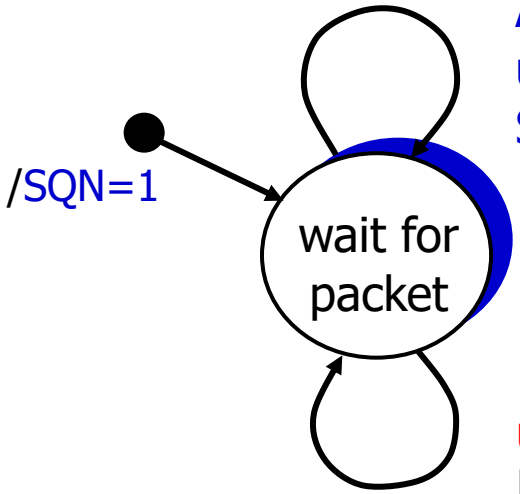
# Stop-and-Wait

- Sender:



# Stop-and-Wait

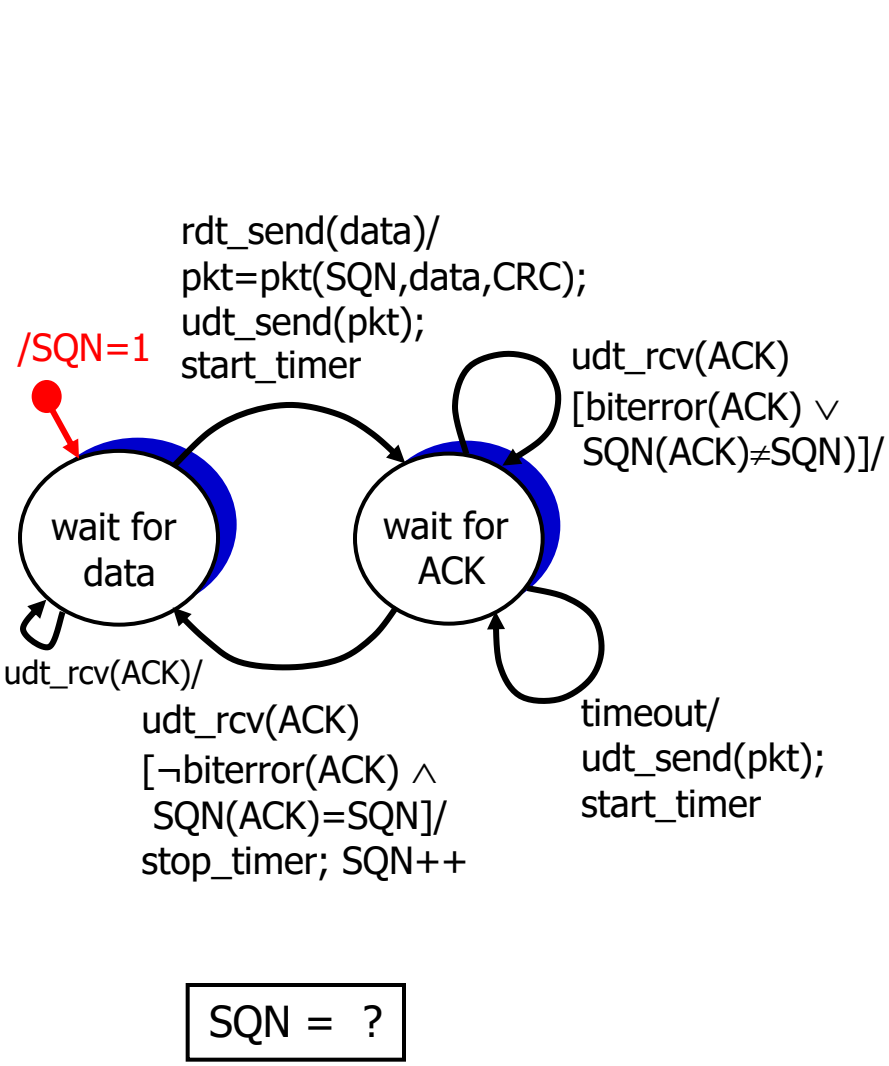
- Receiver:



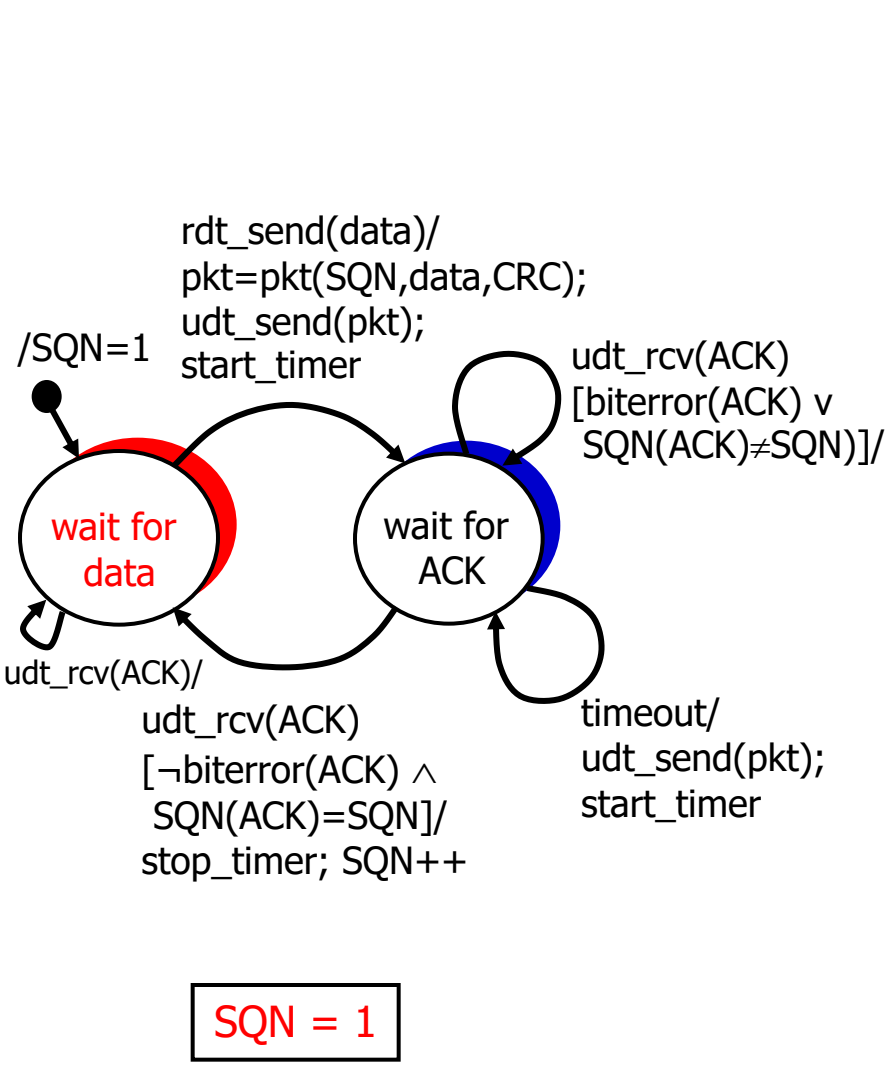

udt_rcv(pkt)
[¬biterror(pkt) ∧ SQN(pkt)=SQN]/
data=extractdata(pkt);
rdt_rcv(data);
ACK=ACK(SQN,CRC);
udt_send(ACK);
SQN++

udt_rcv(pkt)
[biterror(pkt) ∨ SQN(pkt)≠SQN]/
udt_send(ACK)

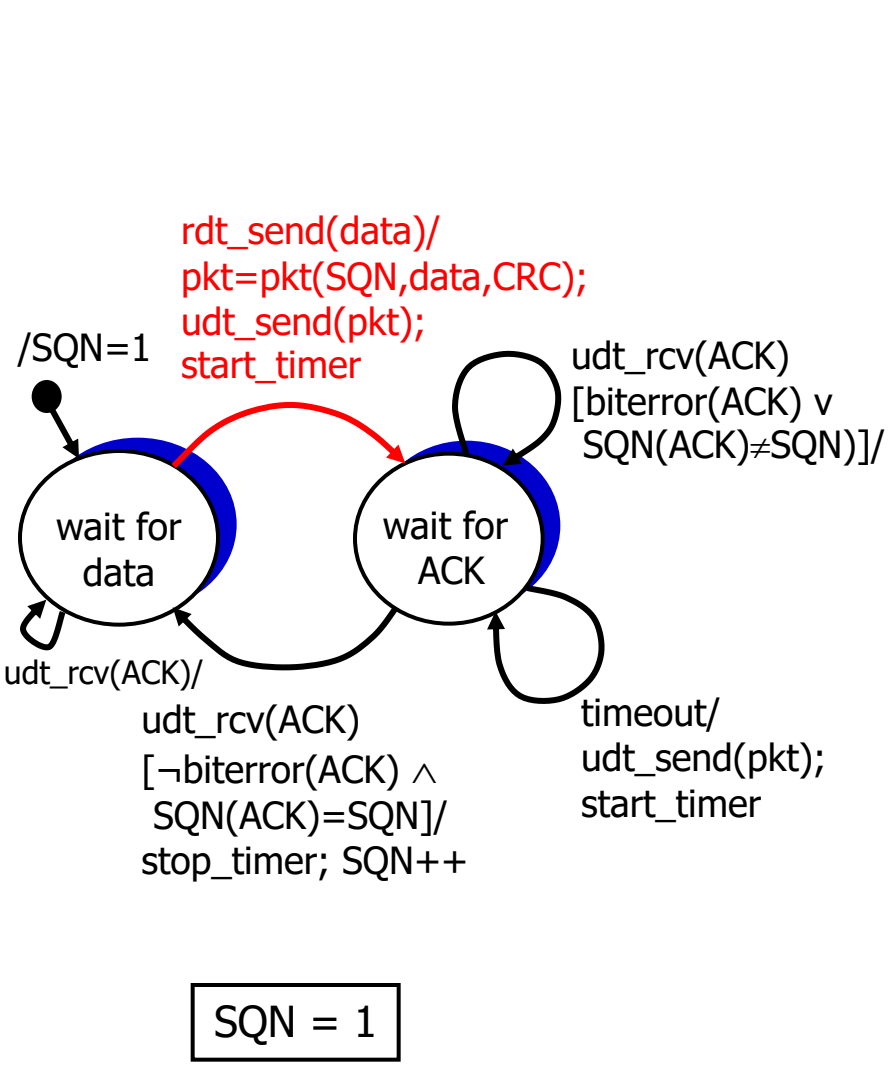
# Stop-and-Wait: normal execution



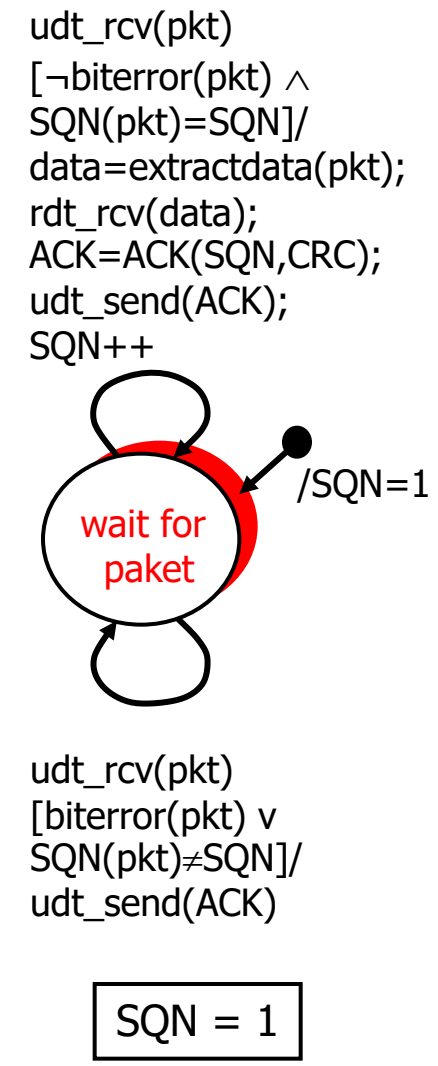
# Stop-and-Wait: normal execution



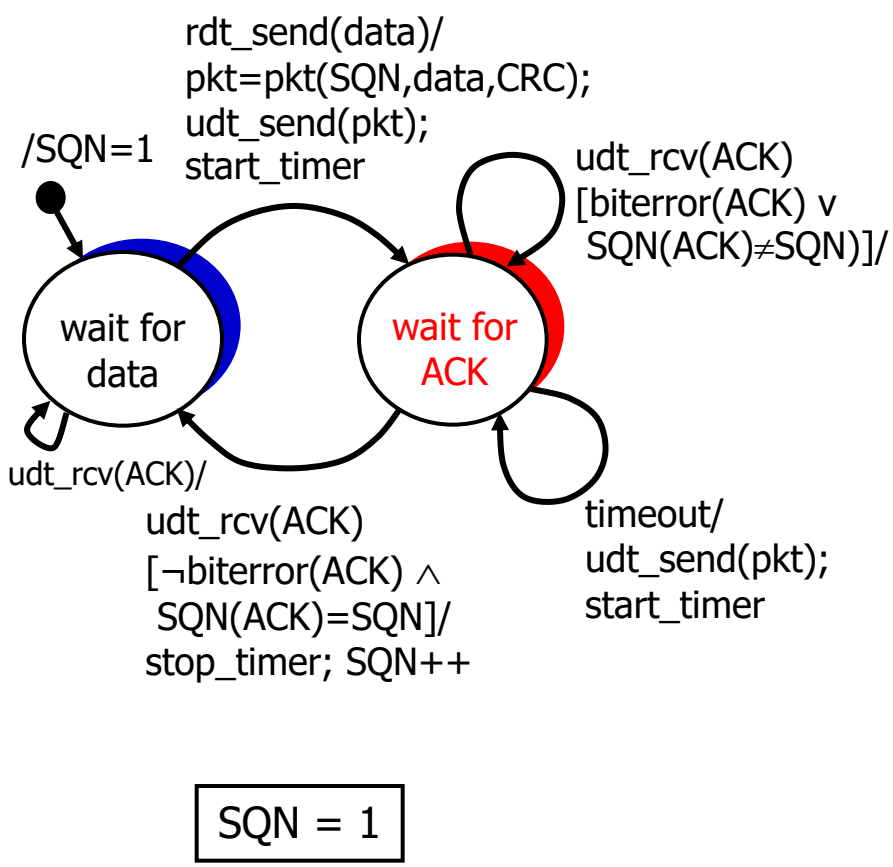
# Stop-and-Wait: normal execution

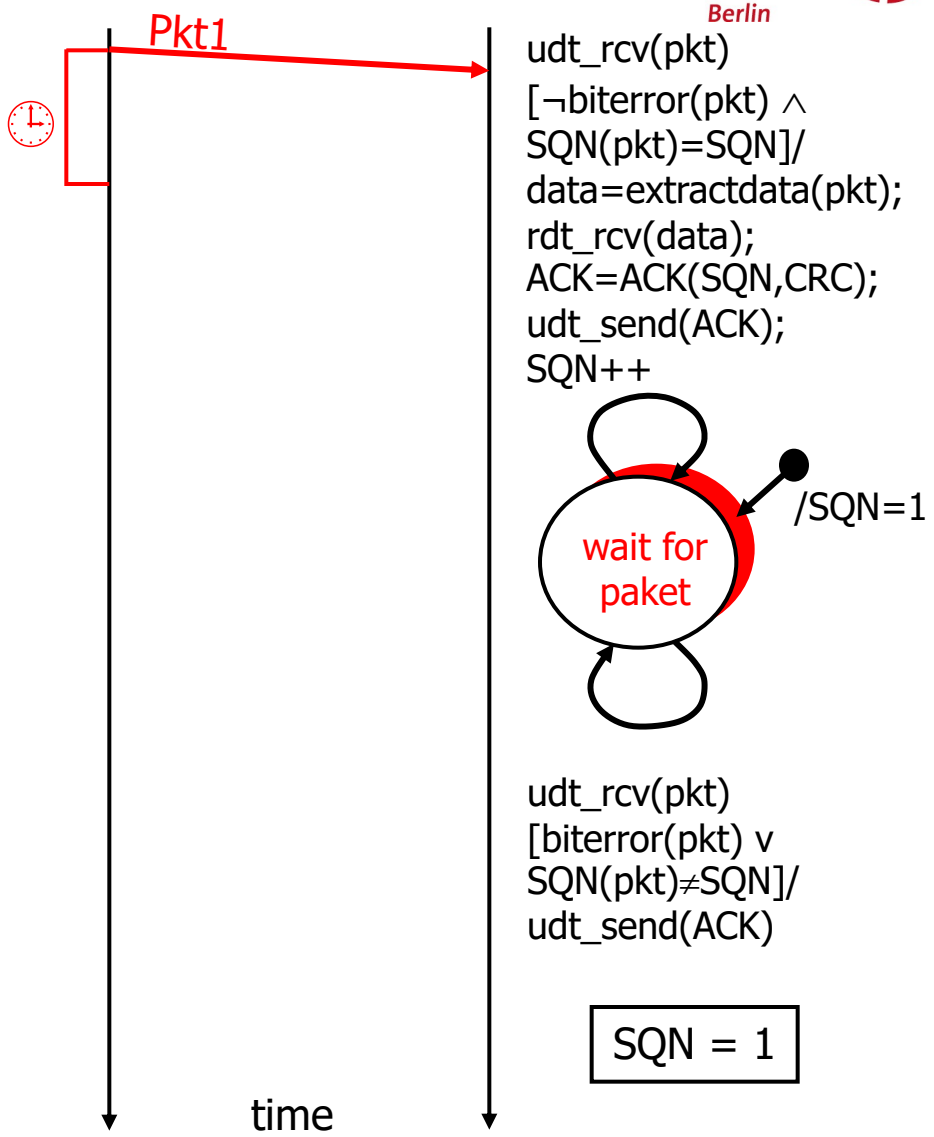



time

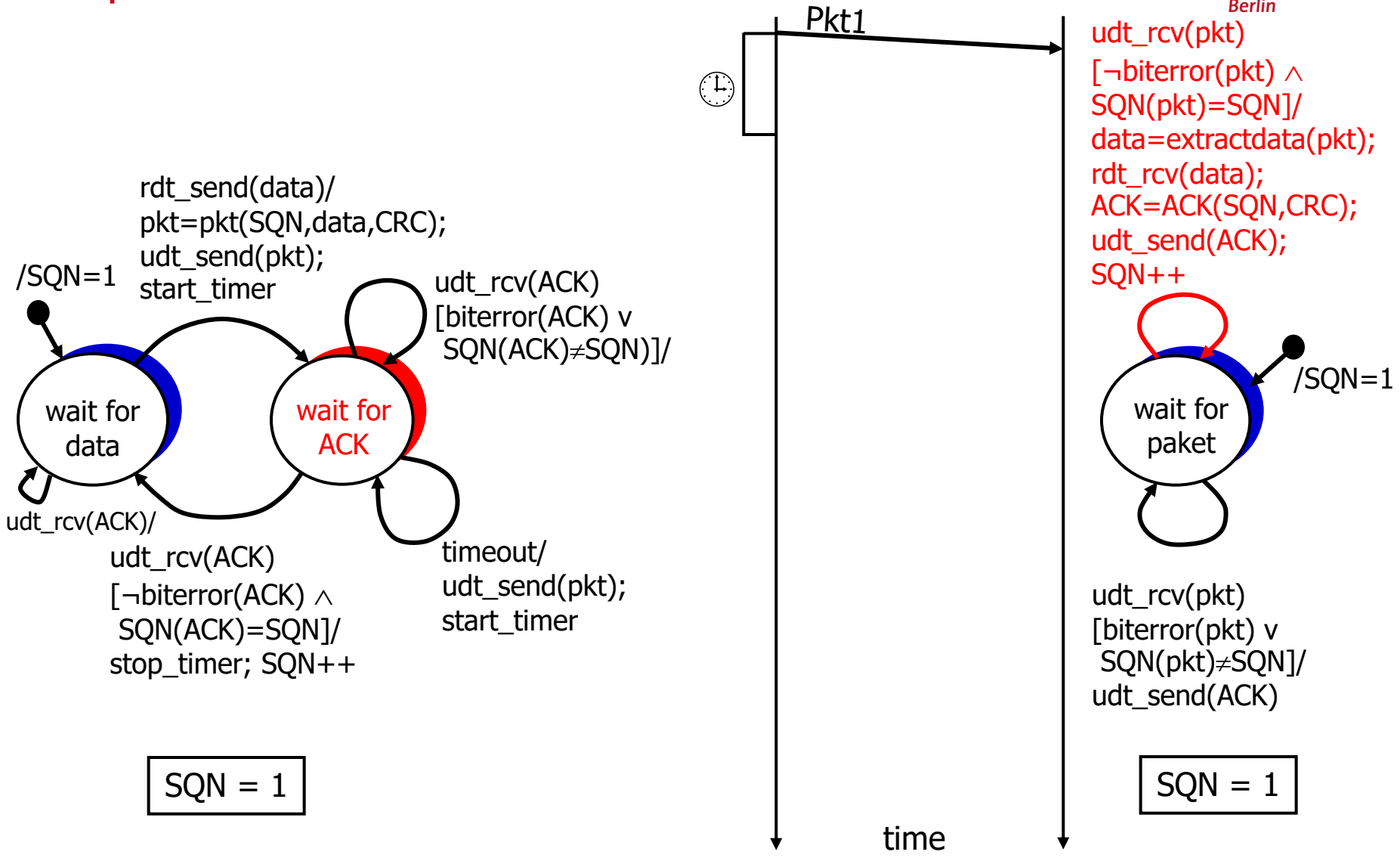


# Stop-and-Wait: normal execution

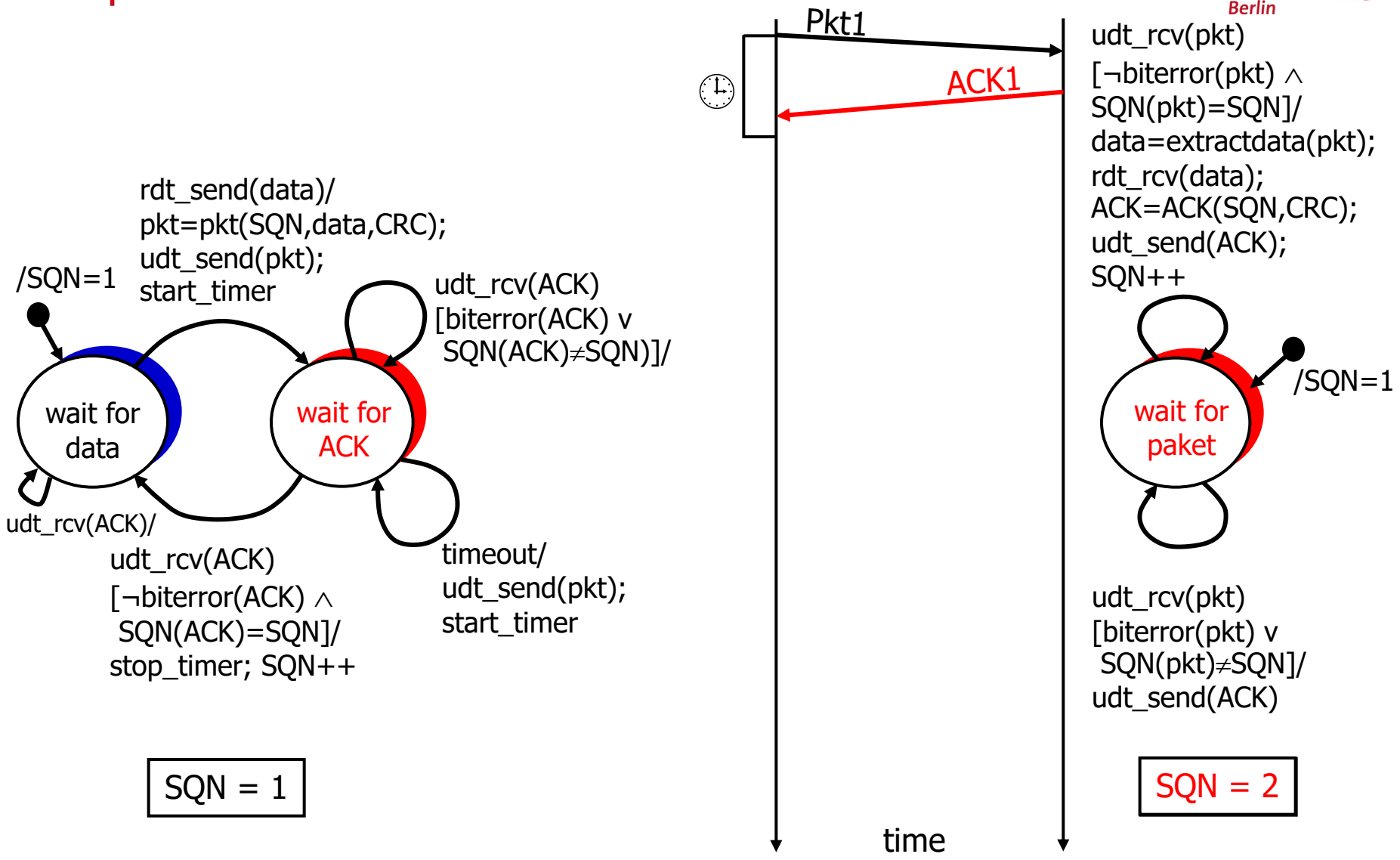


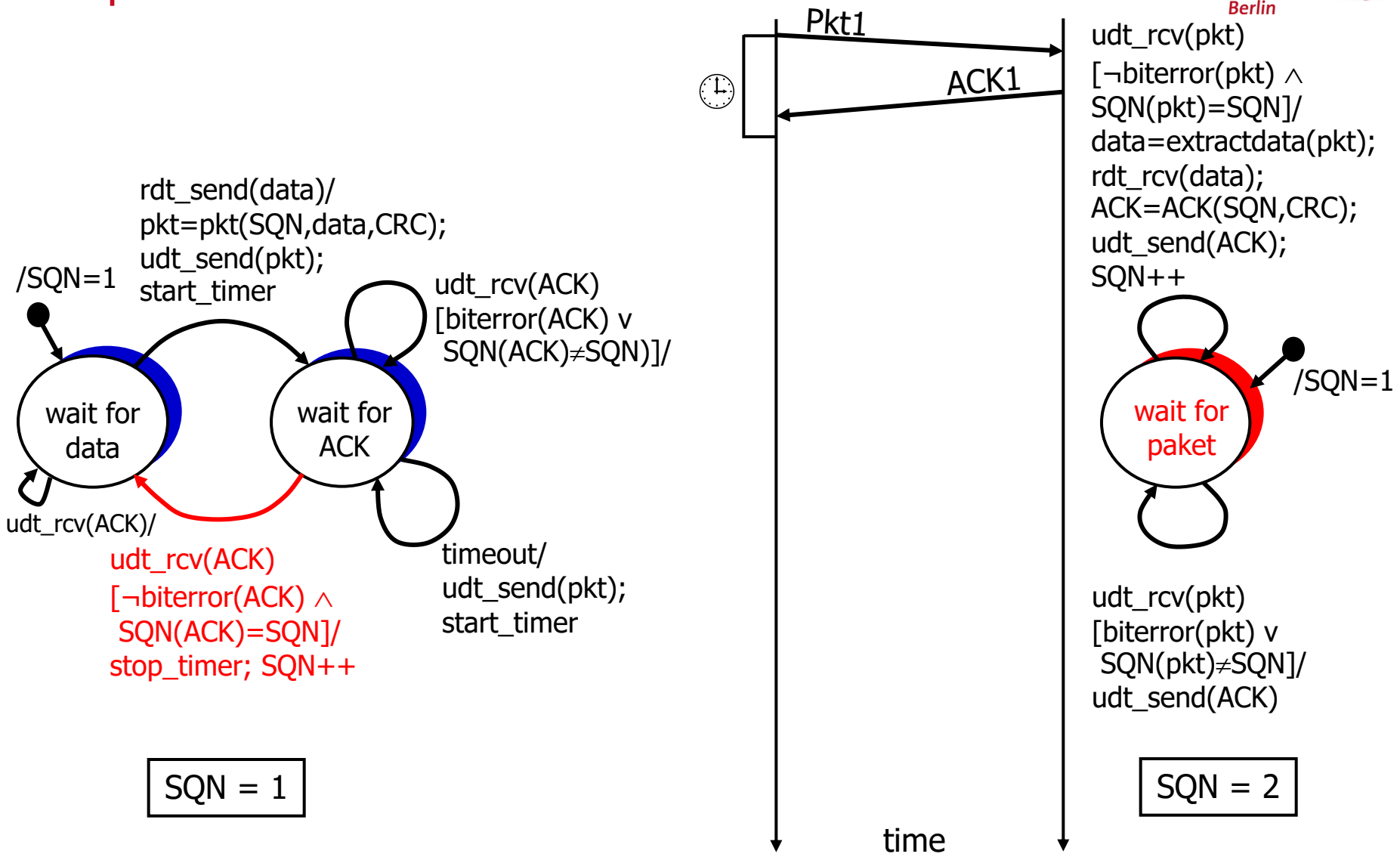
# Stop-and-Wait: normal execution



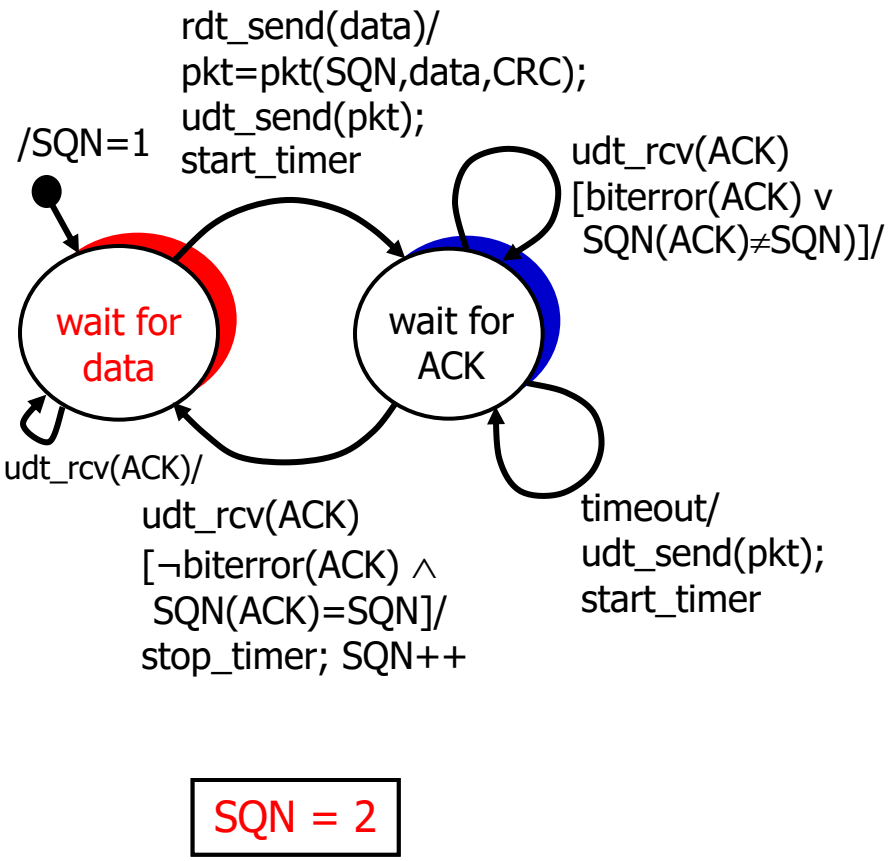
# Stop-and-Wait: normal execution

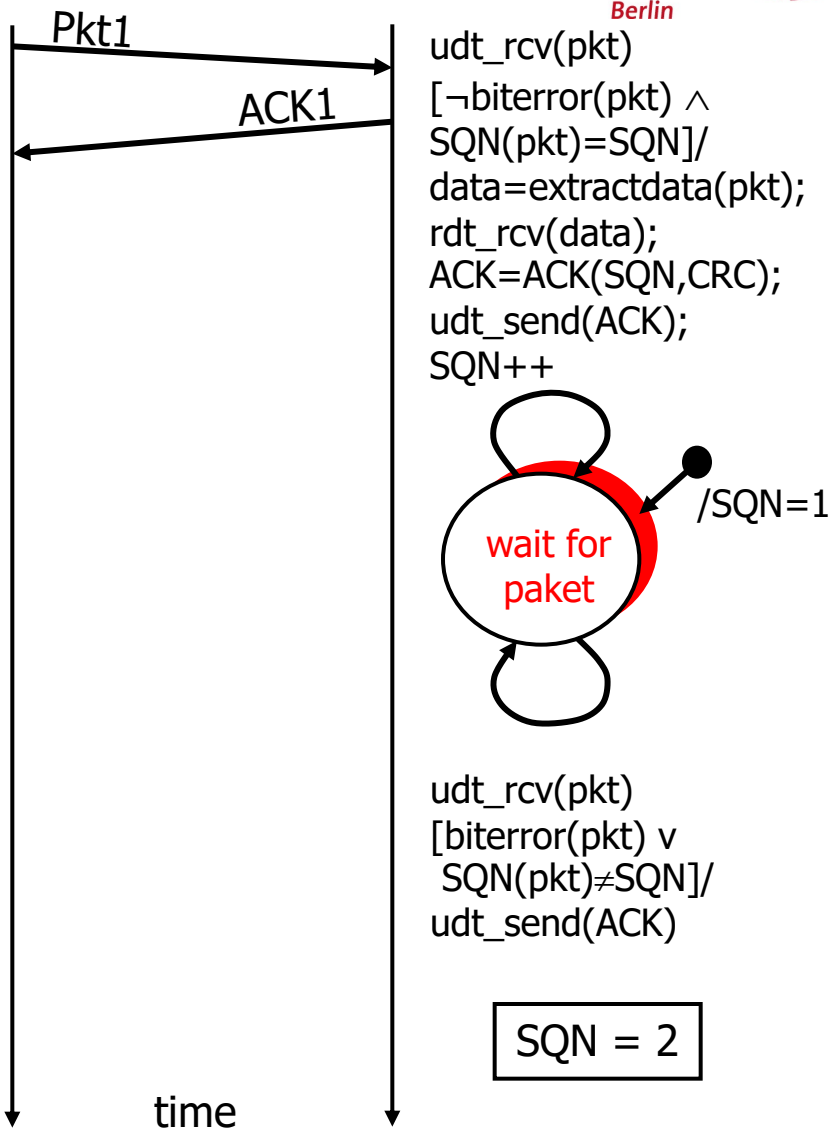


# Stop-and-Wait: normal execution

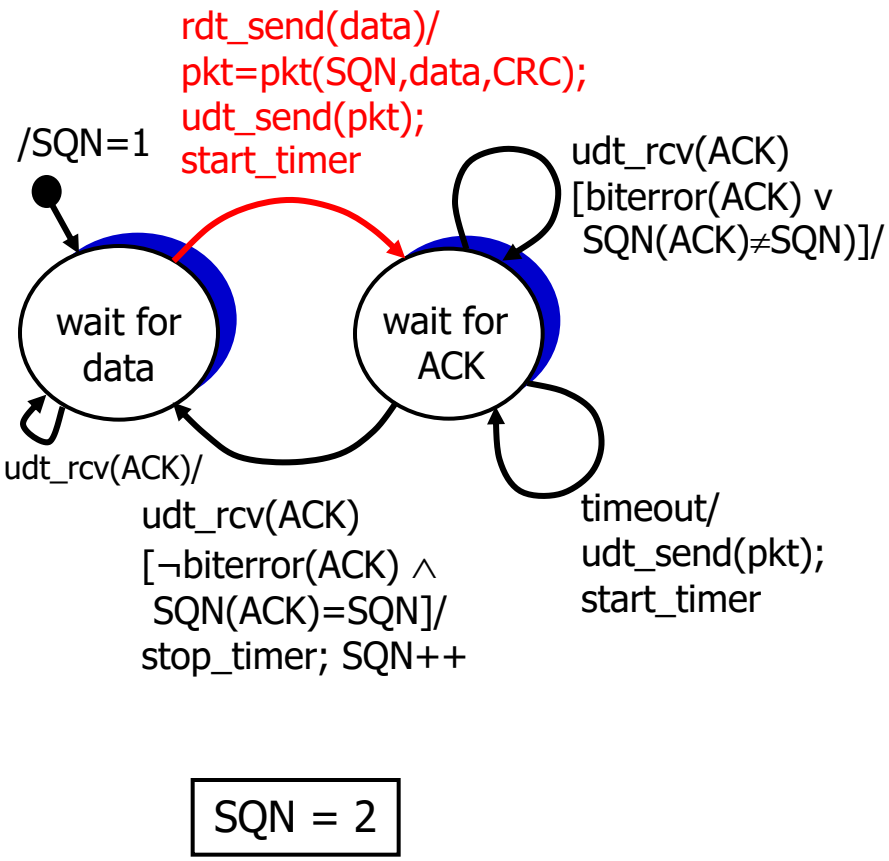


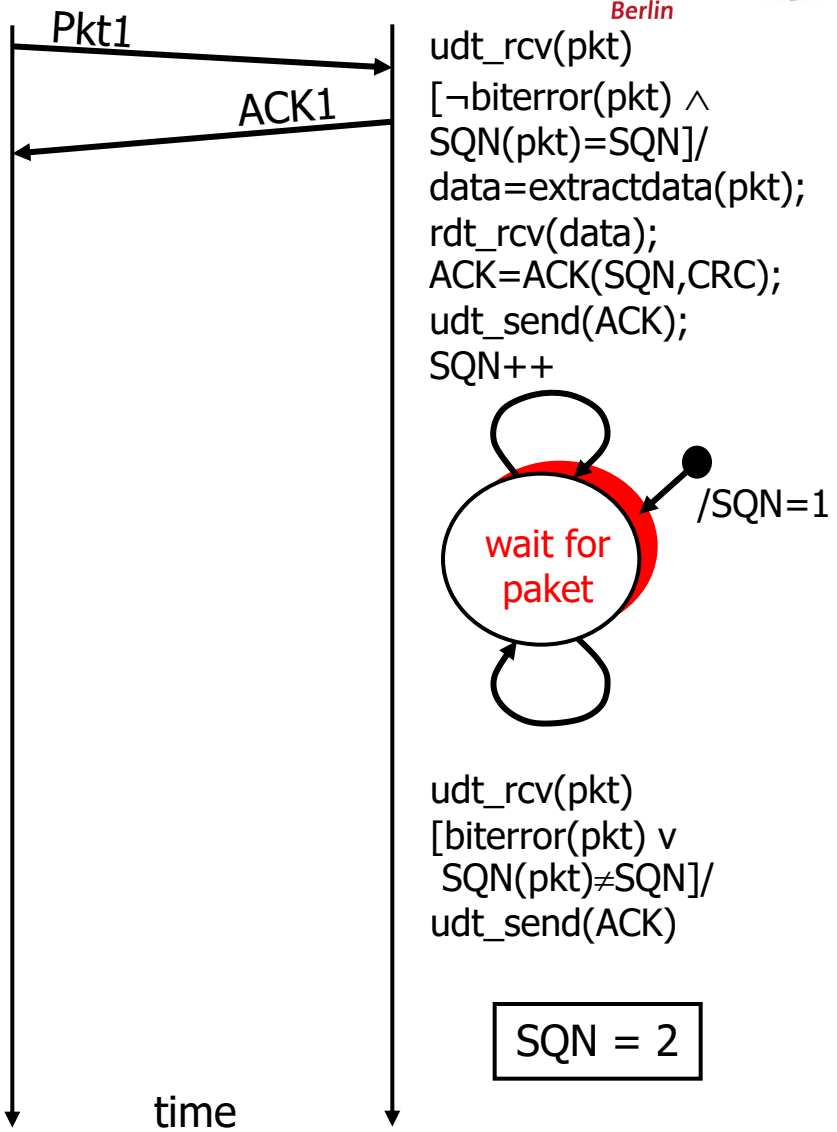
# Stop-and-Wait: normal execution



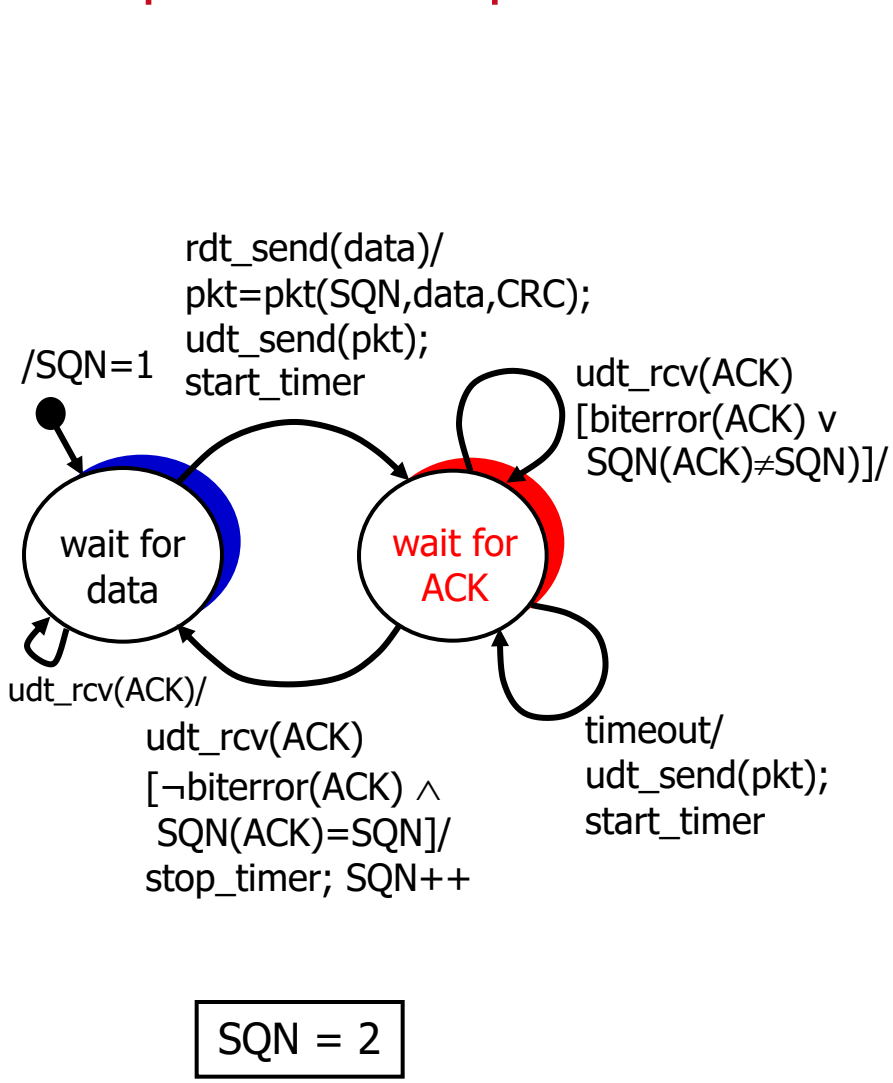



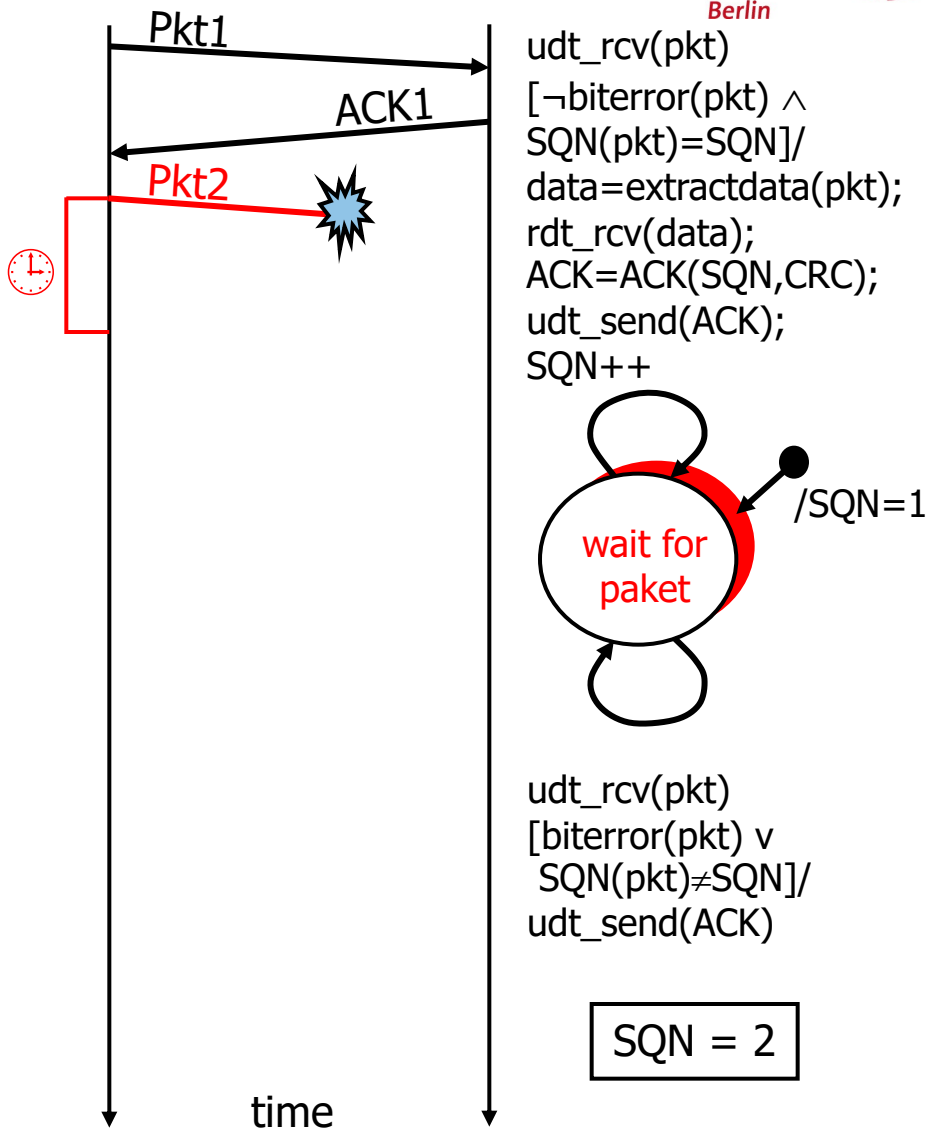
# Stop-and-Wait: packet loss



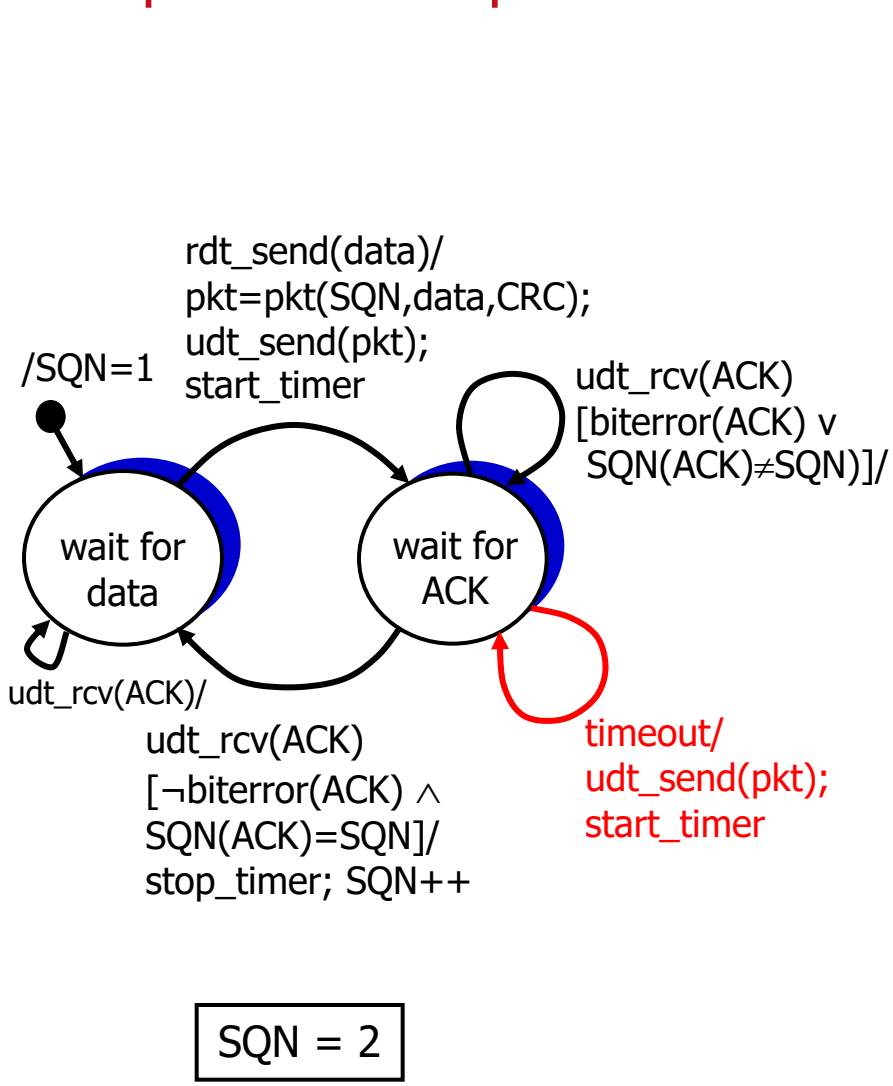



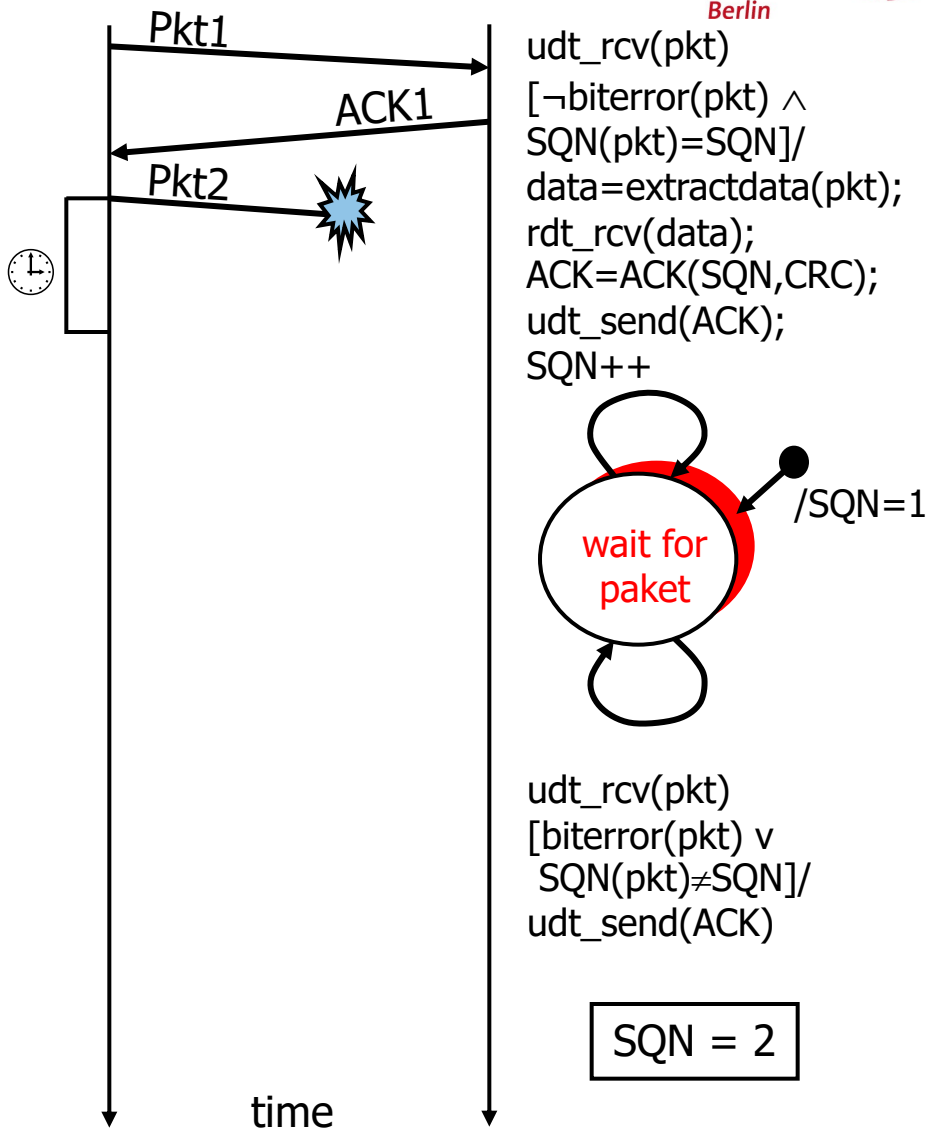
# Stop-and-Wait: packet loss



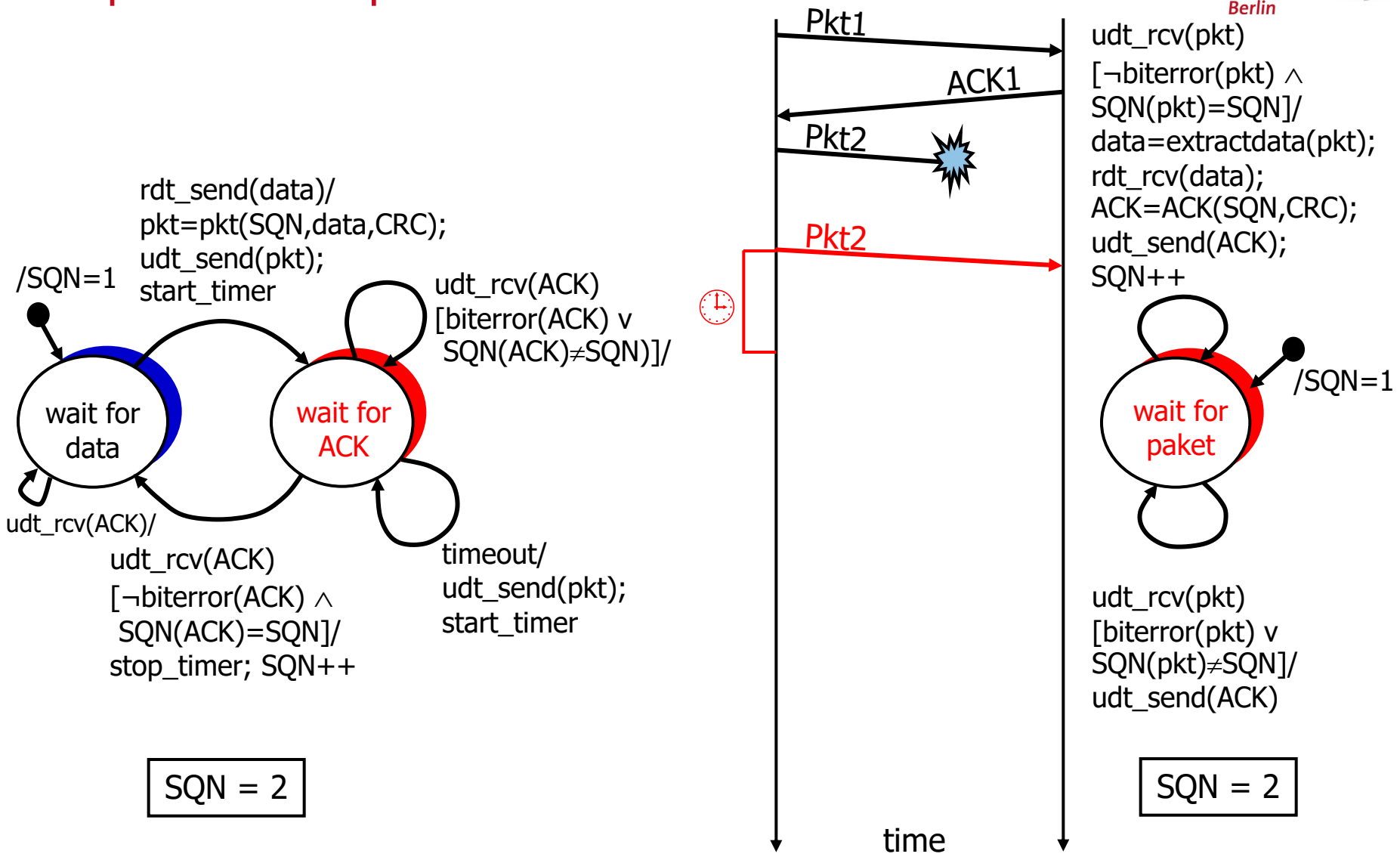



# Stop-and-Wait: packet loss

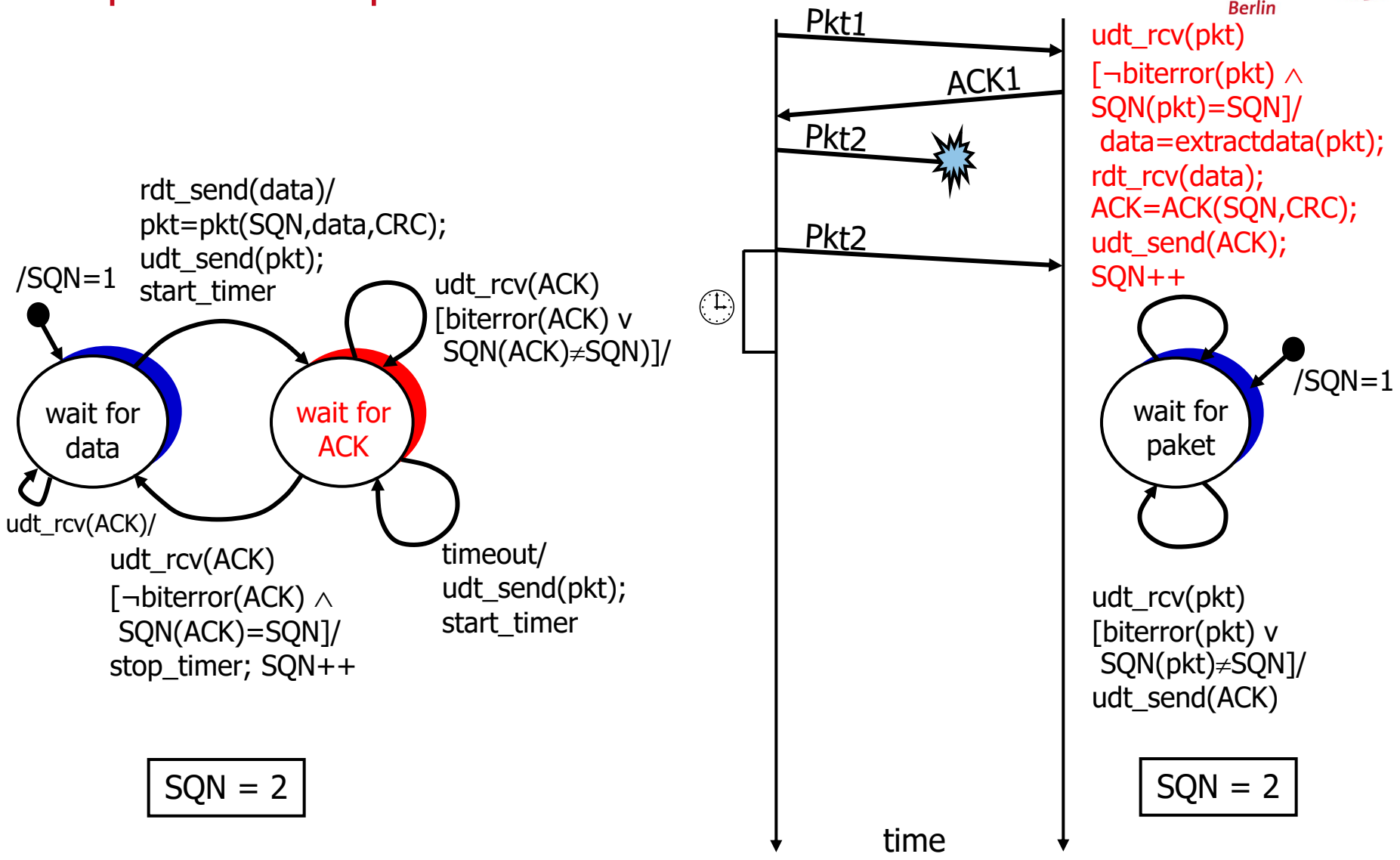


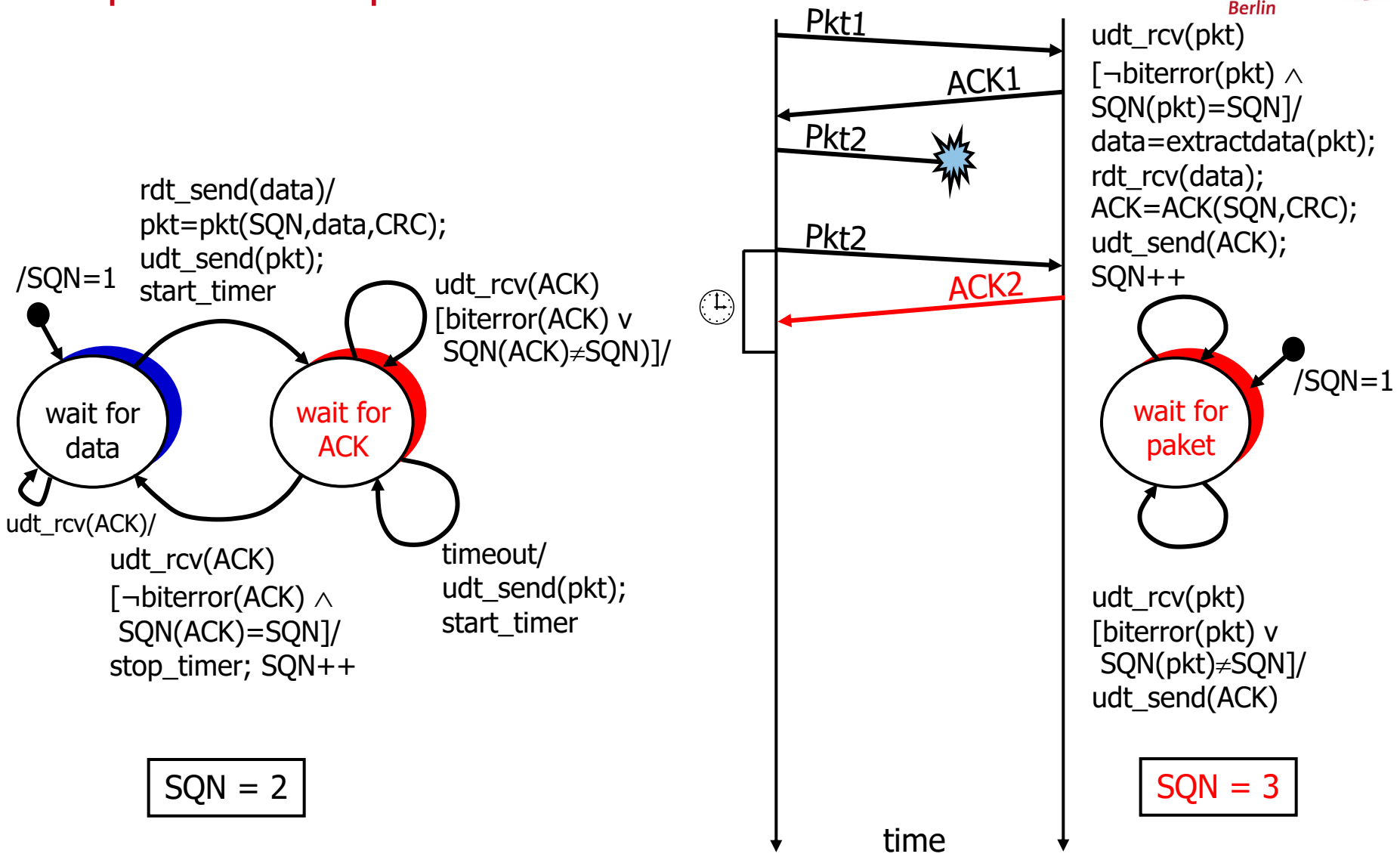
# Stop-and-Wait: packet loss



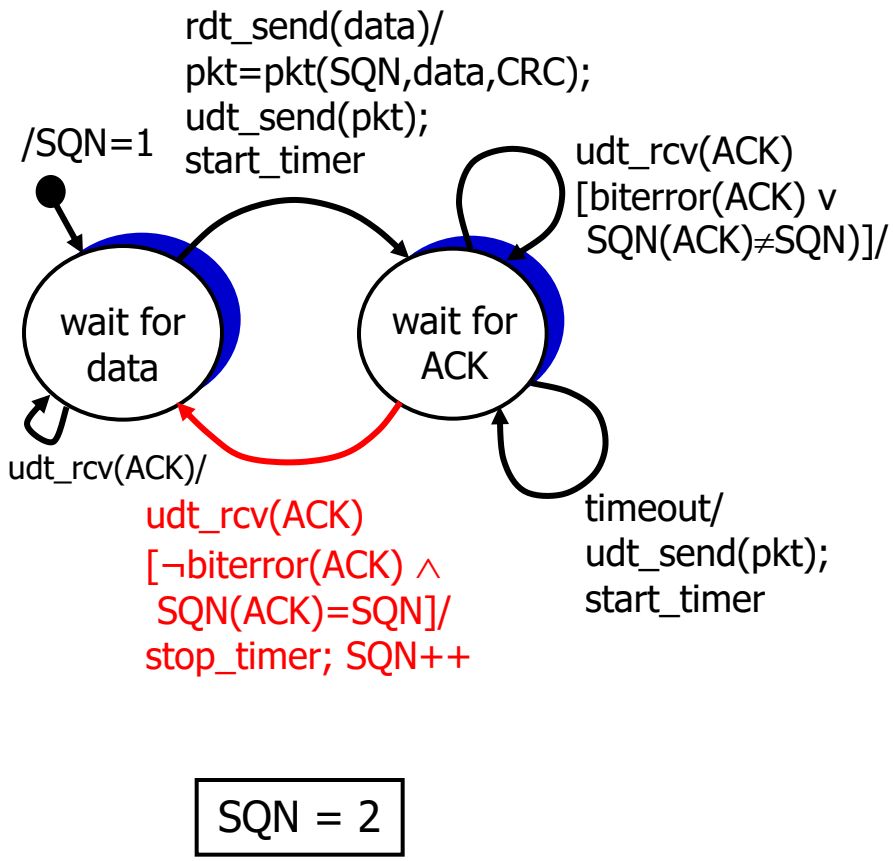
# Stop-and-Wait: packet loss

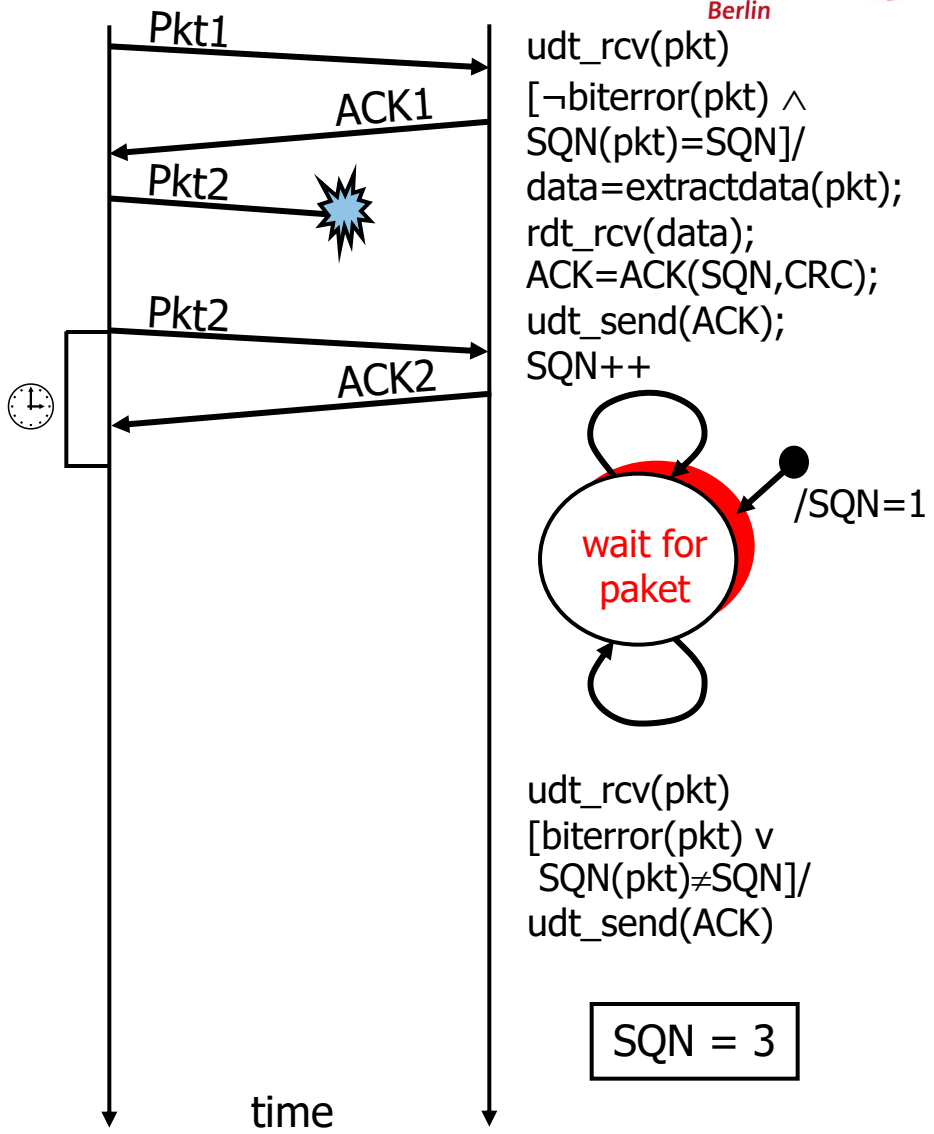


# Stop-and-Wait: packet loss

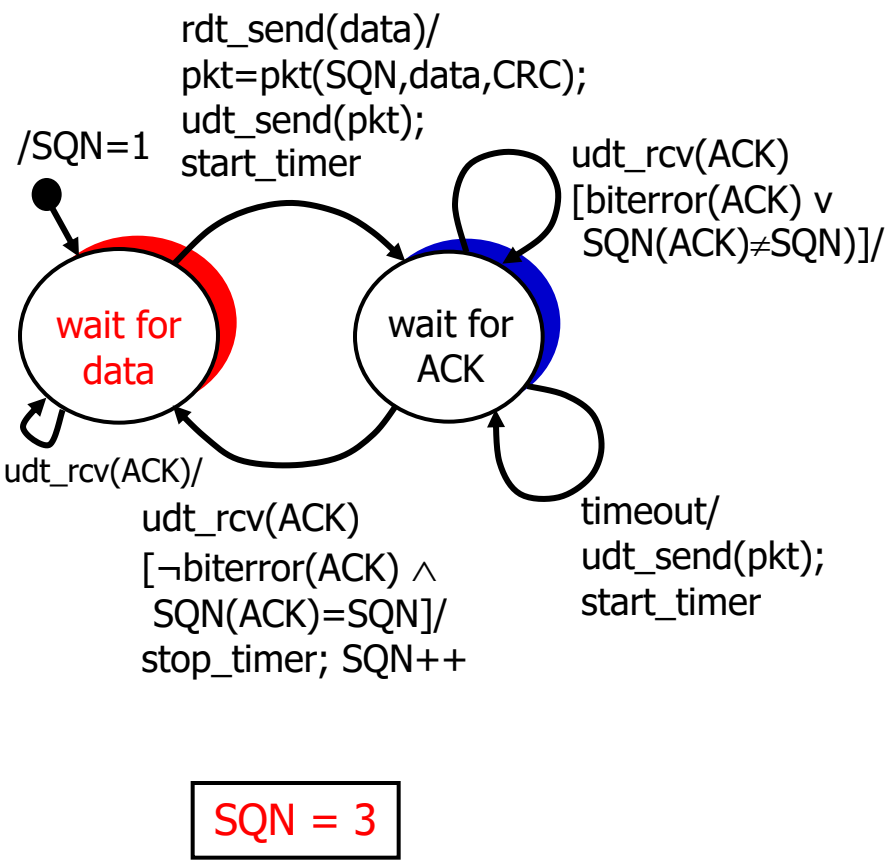


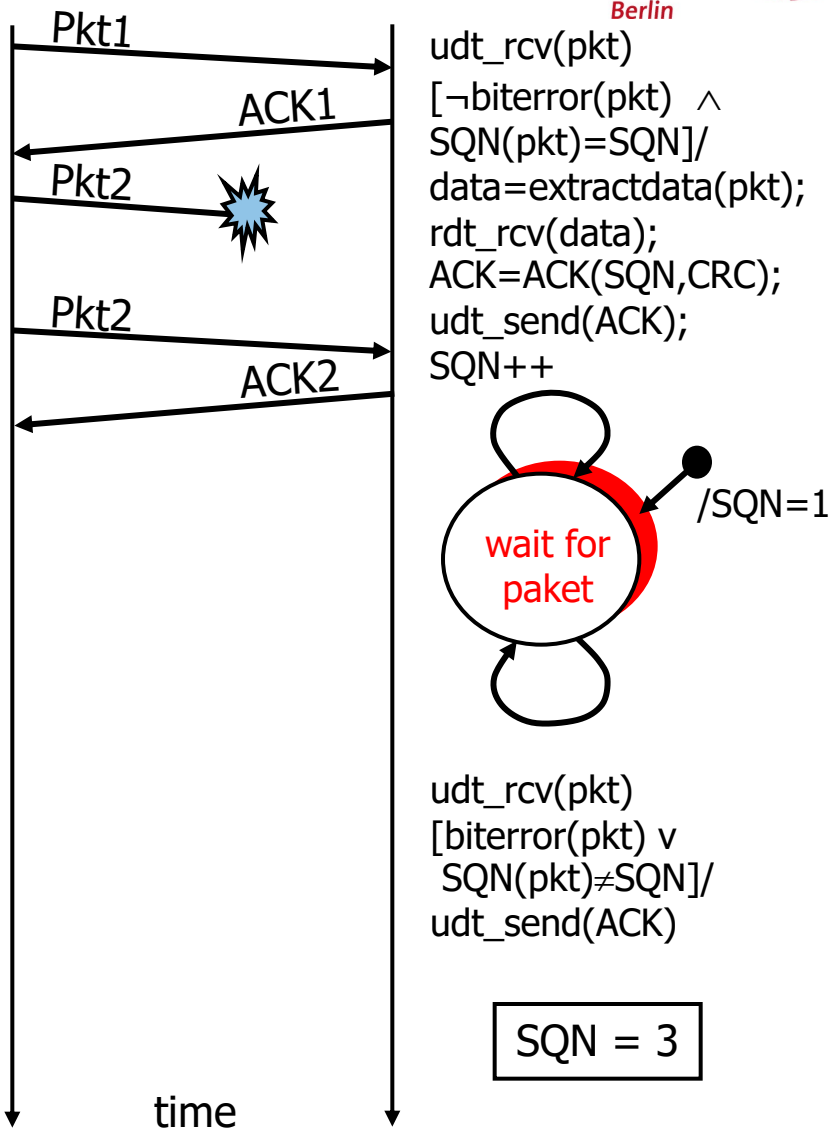
# Stop-and-Wait: packet loss



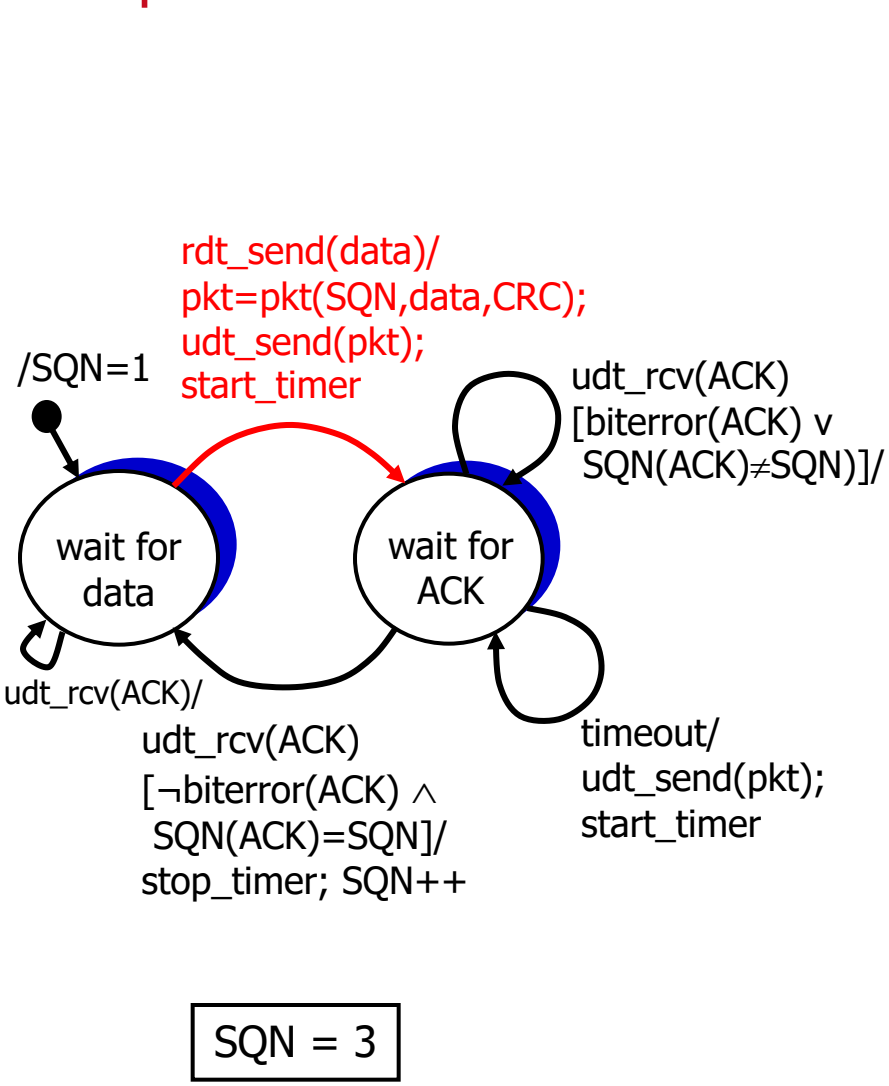



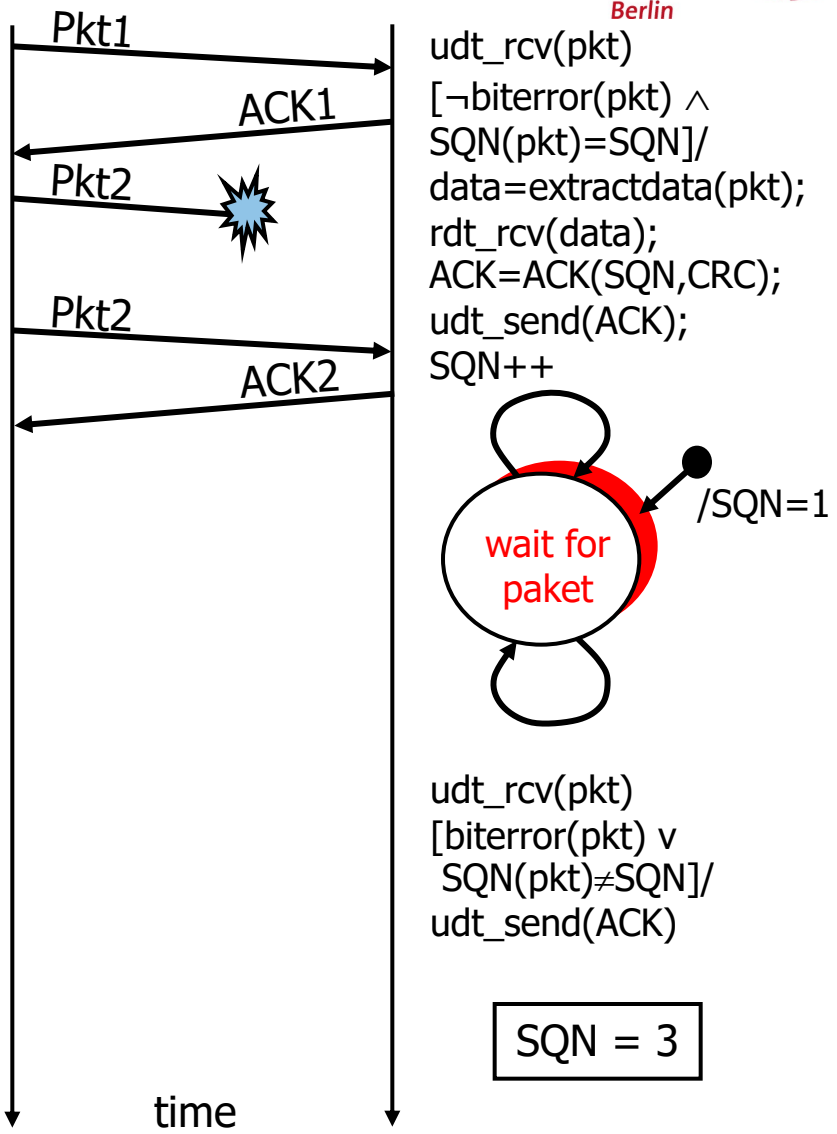
# Stop-and-Wait: packet loss



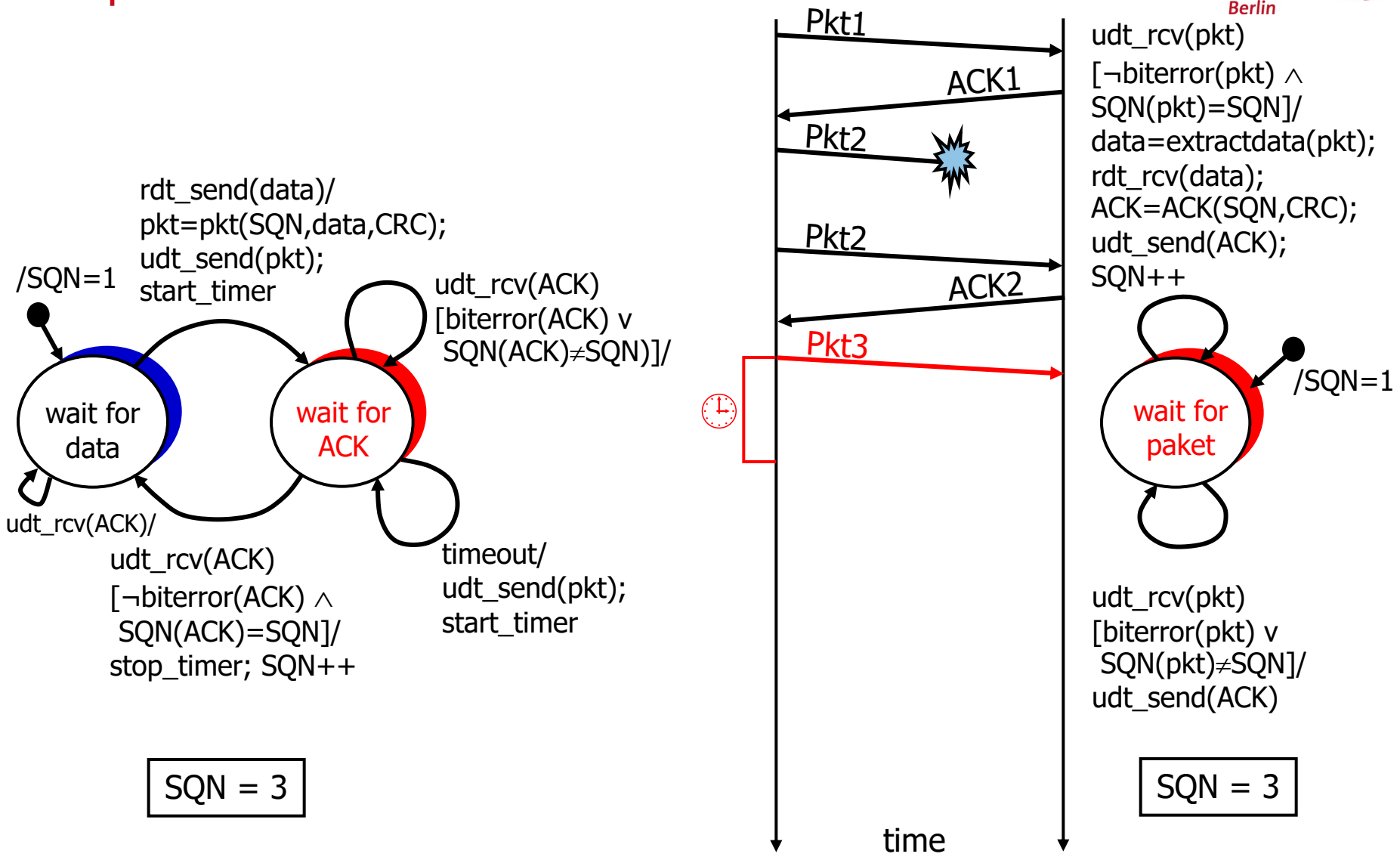



# Stop-and-Wait: loss of ACK

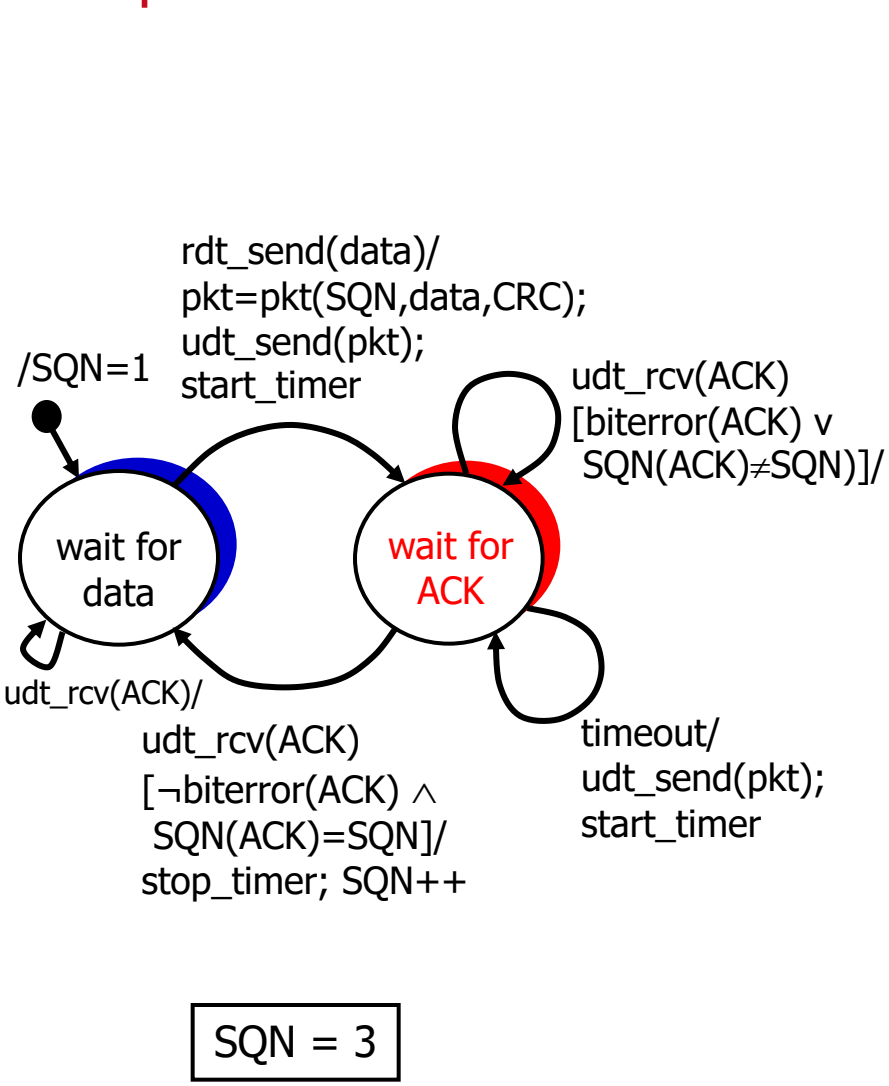


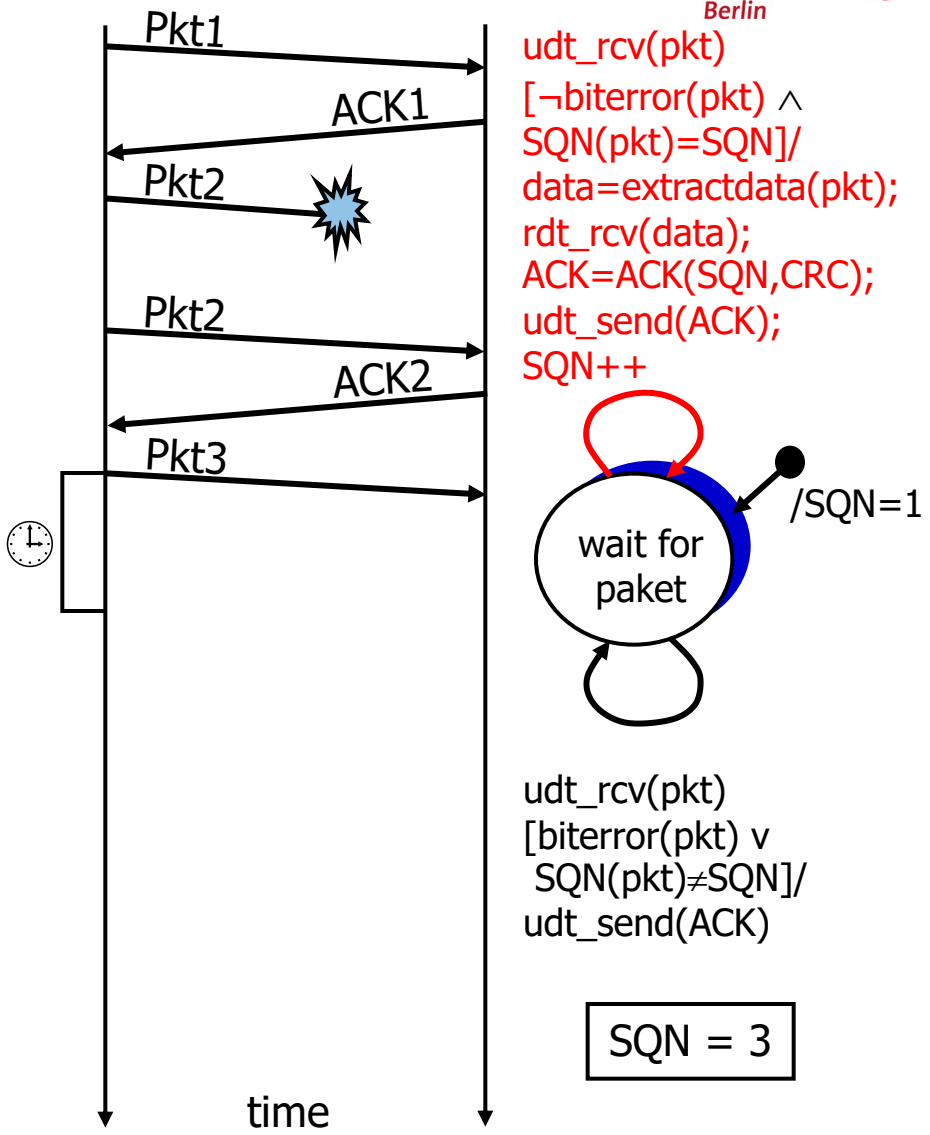



# Stop-and-Wait: loss of ACK

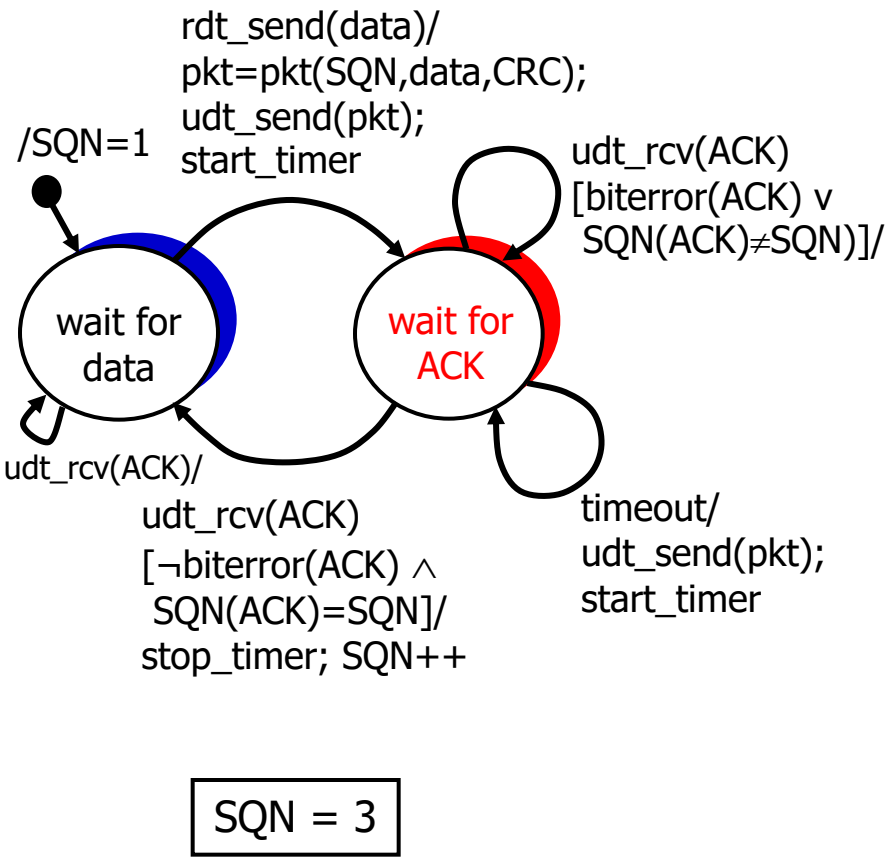


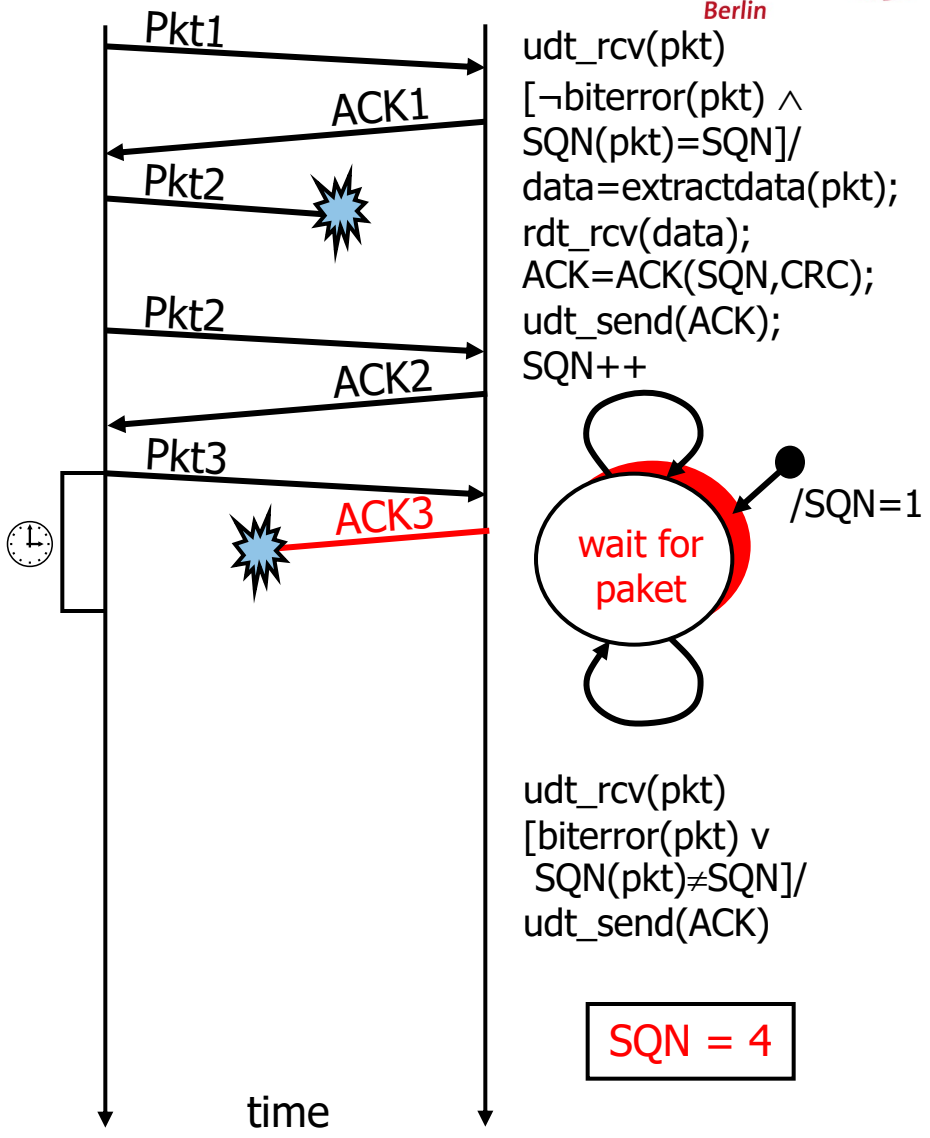
# Stop-and-Wait: loss of ACK



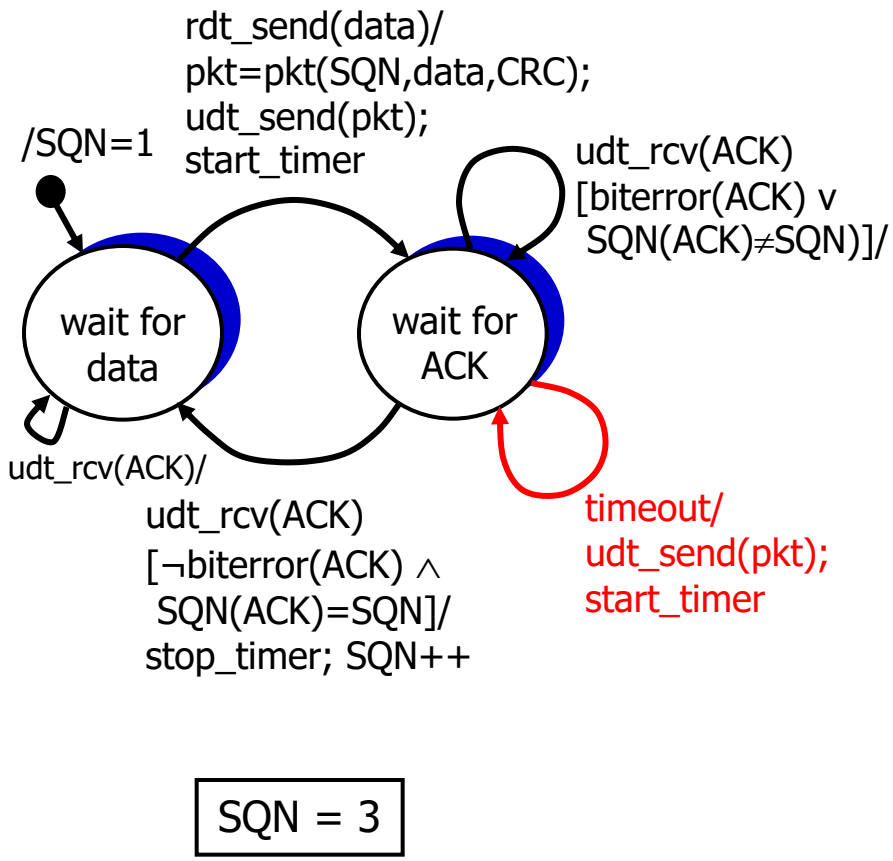



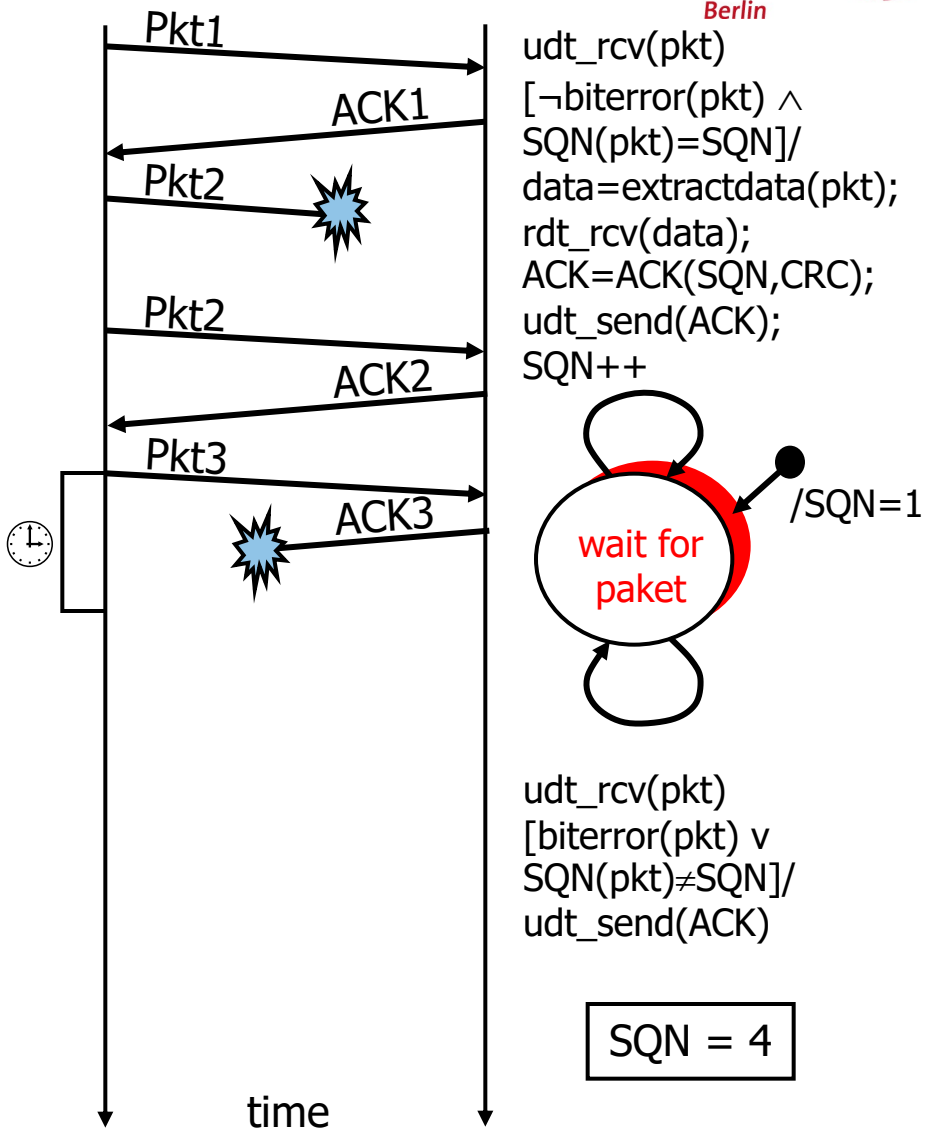
# Stop-and-Wait: loss of ACK



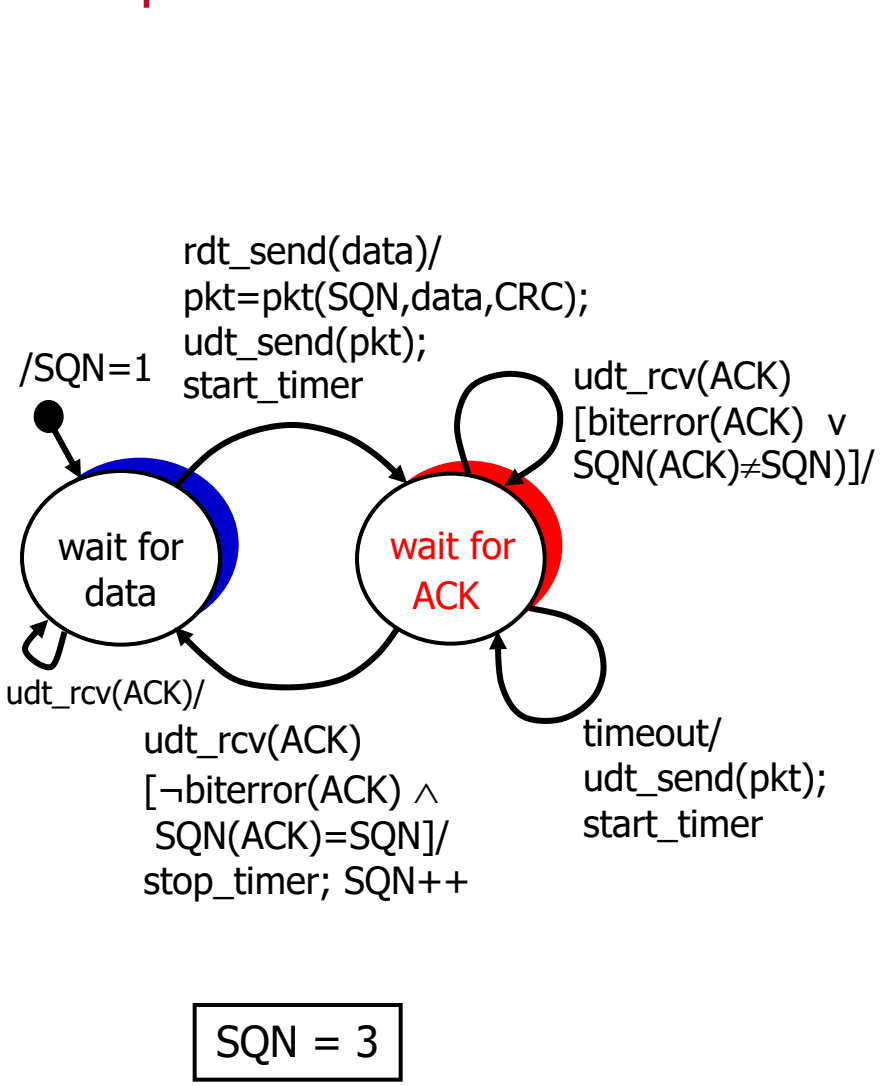



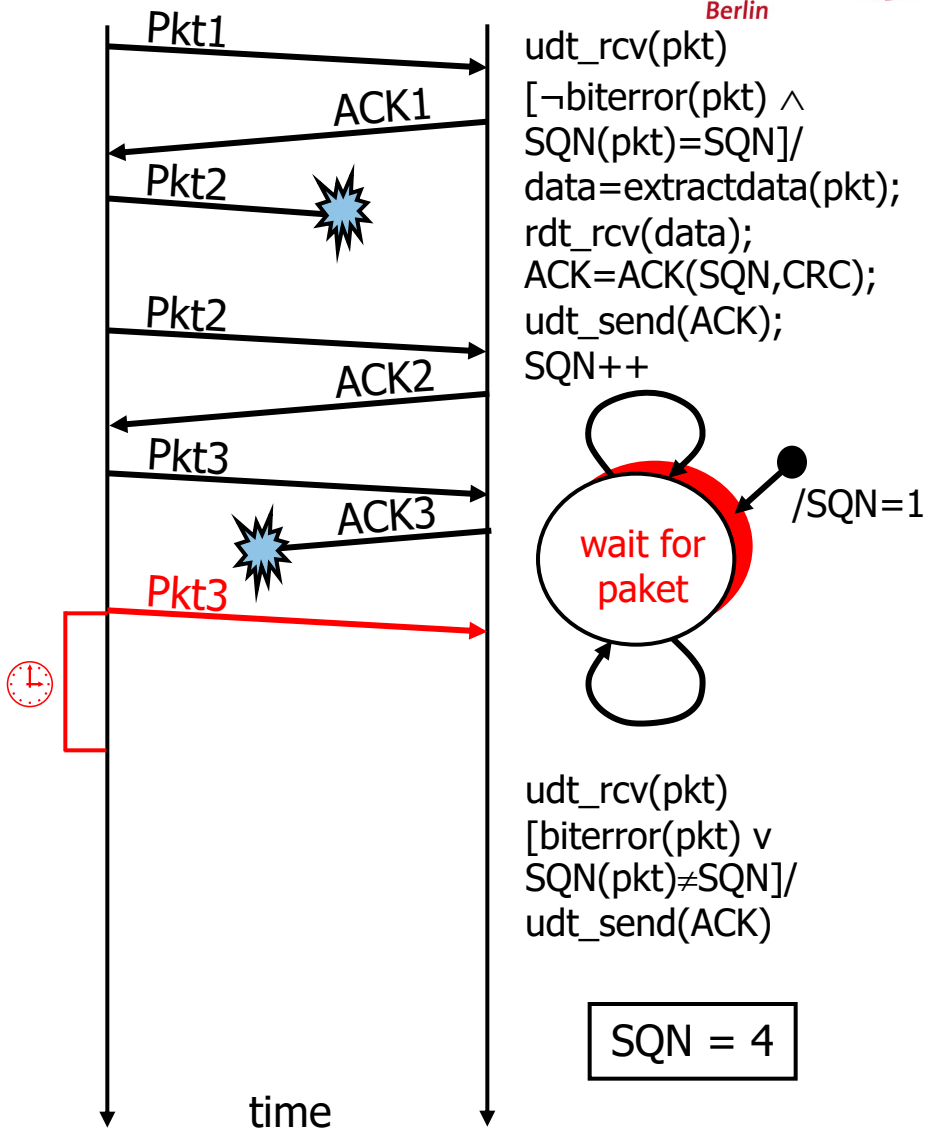
# Stop-and-Wait: loss of ACK



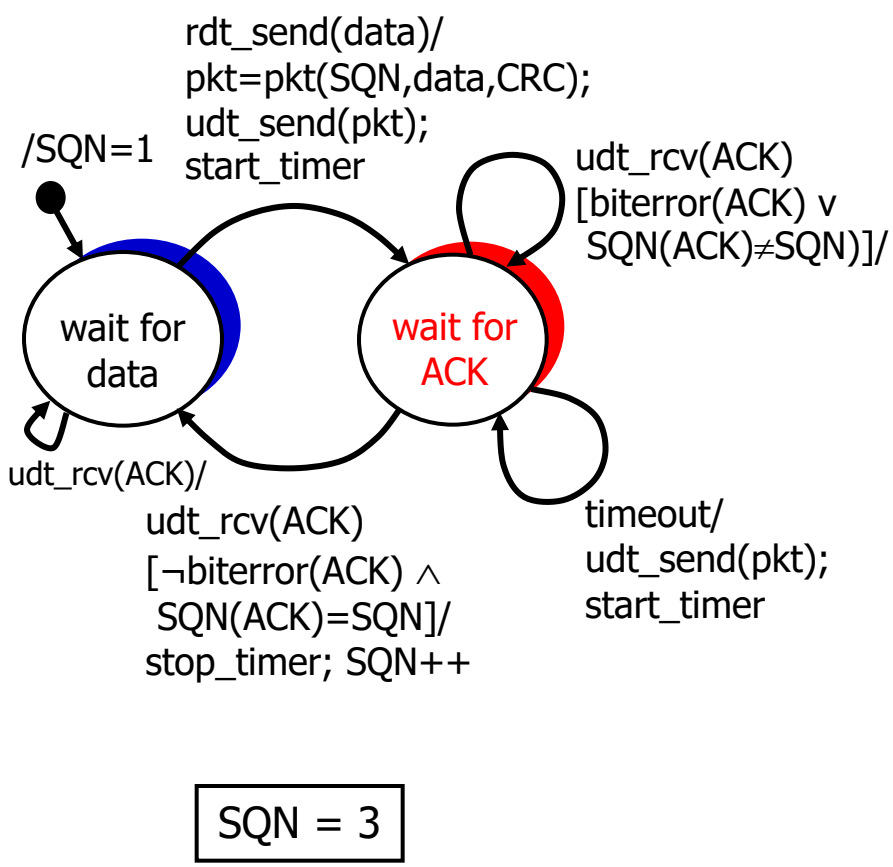



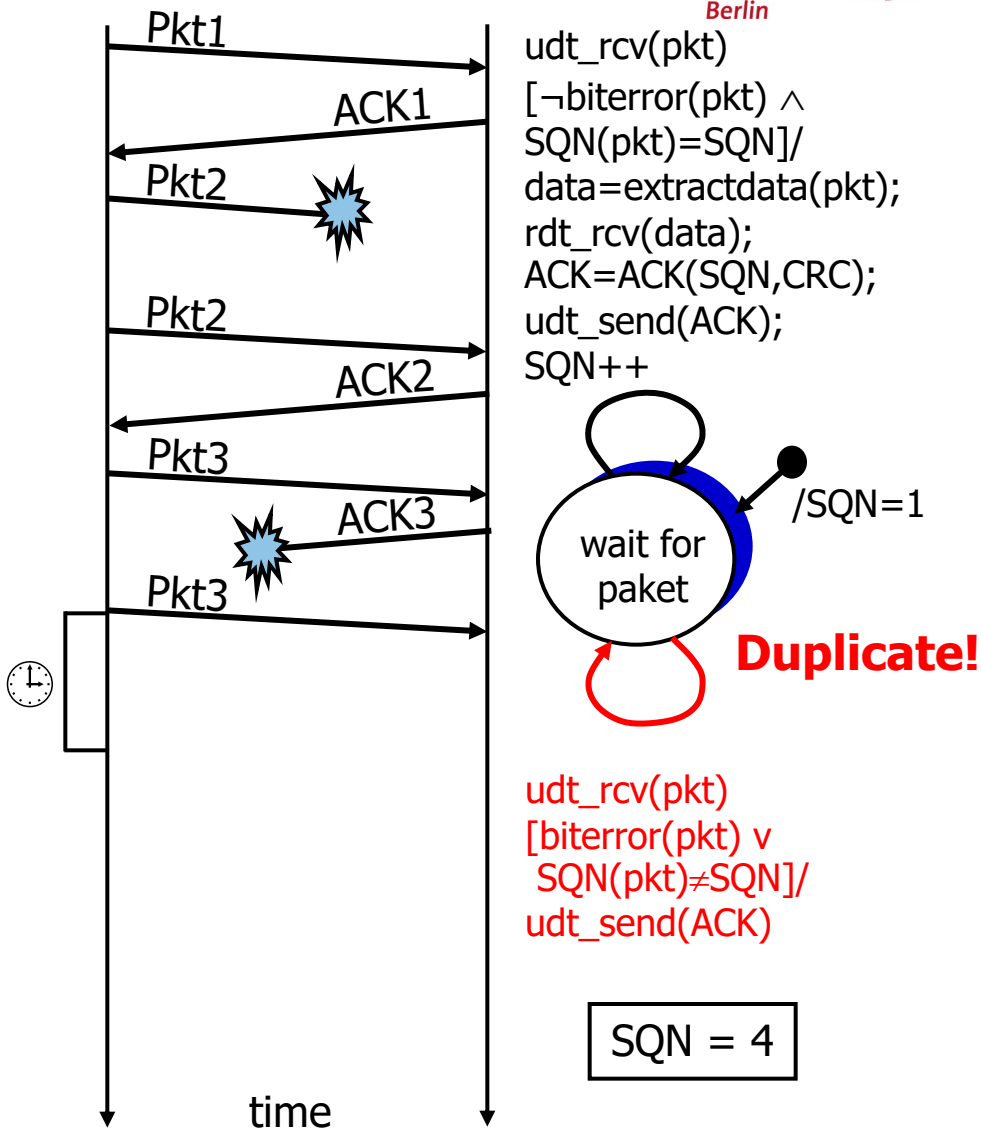
# Stop-and-Wait: loss of ACK



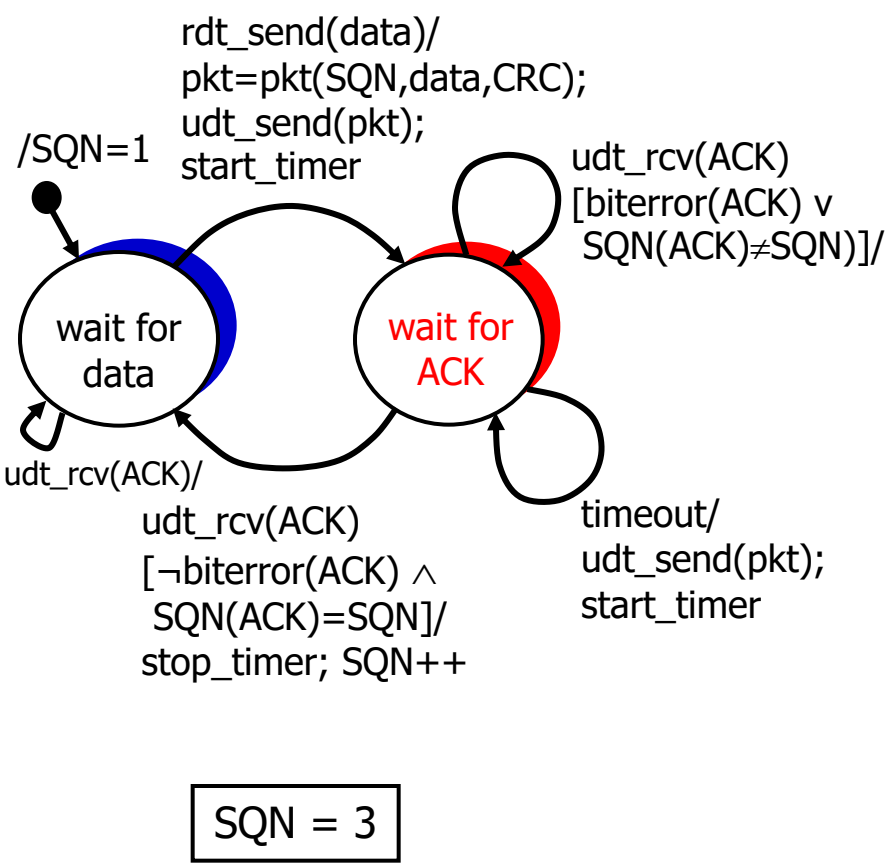



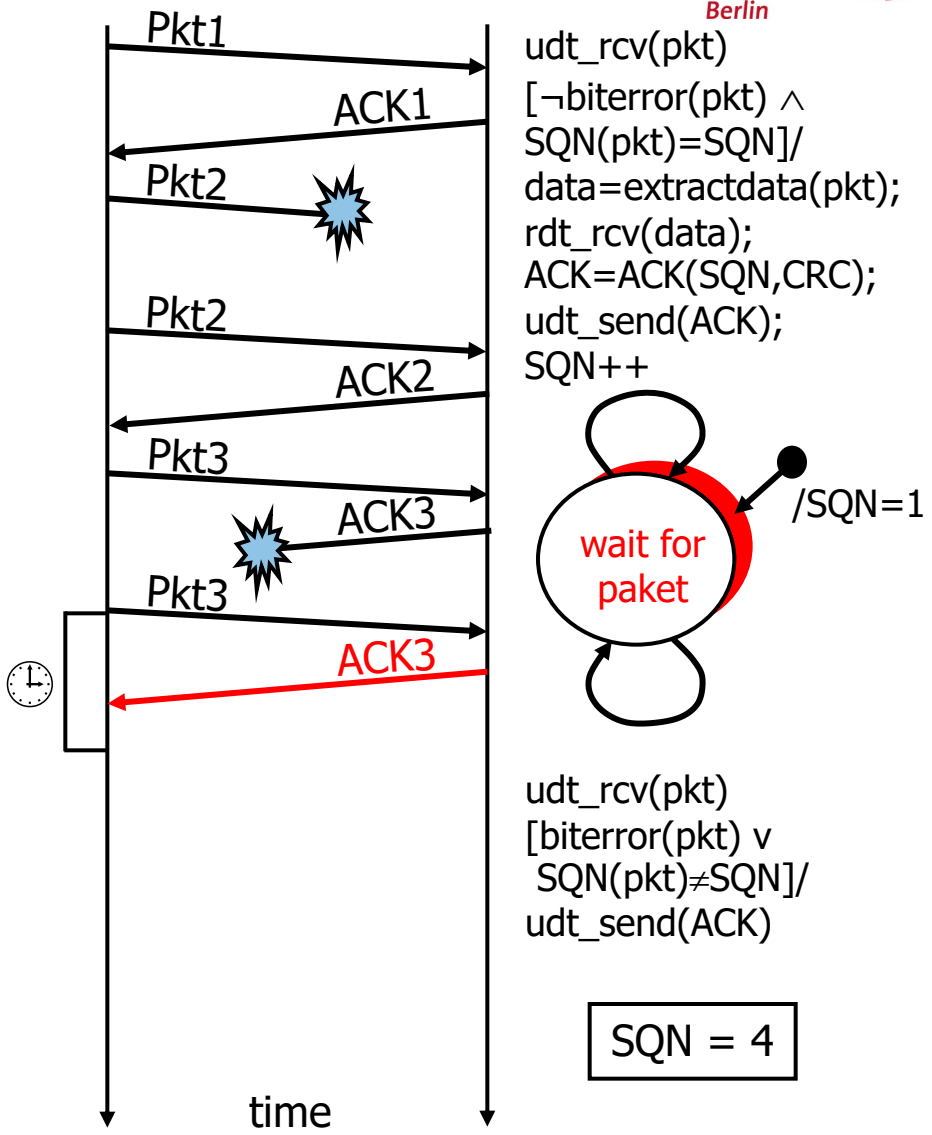
# Stop-and-Wait: loss of ACK



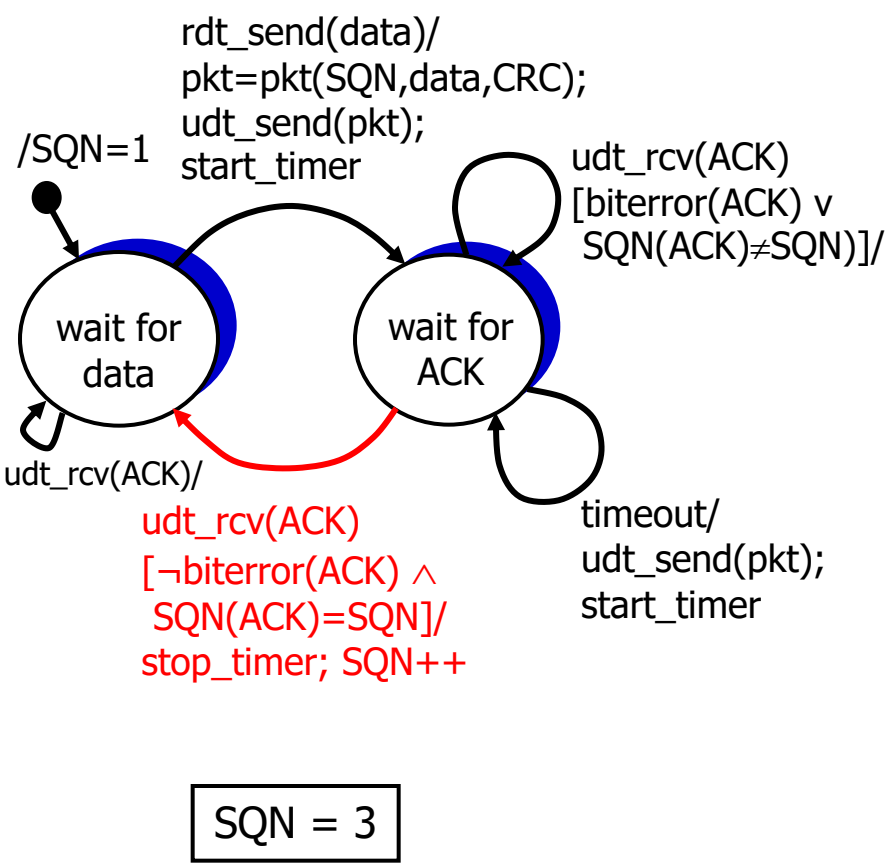



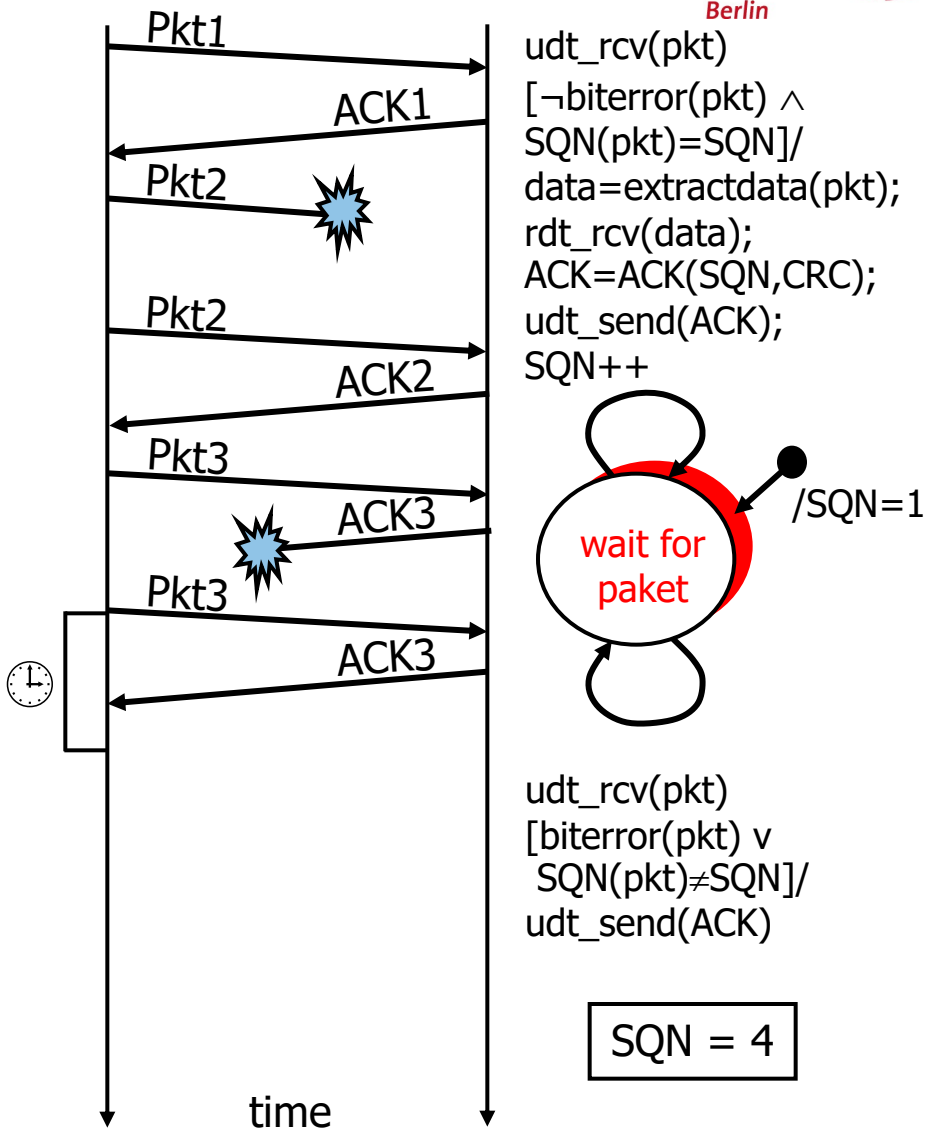
# Stop-and-Wait: loss of ACK



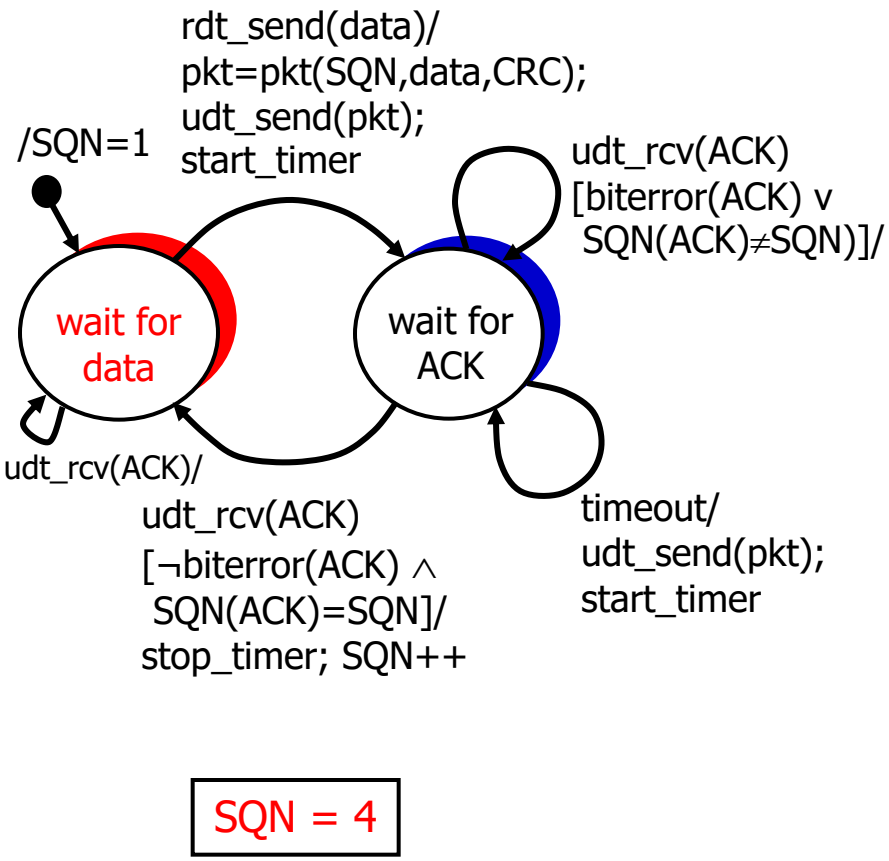



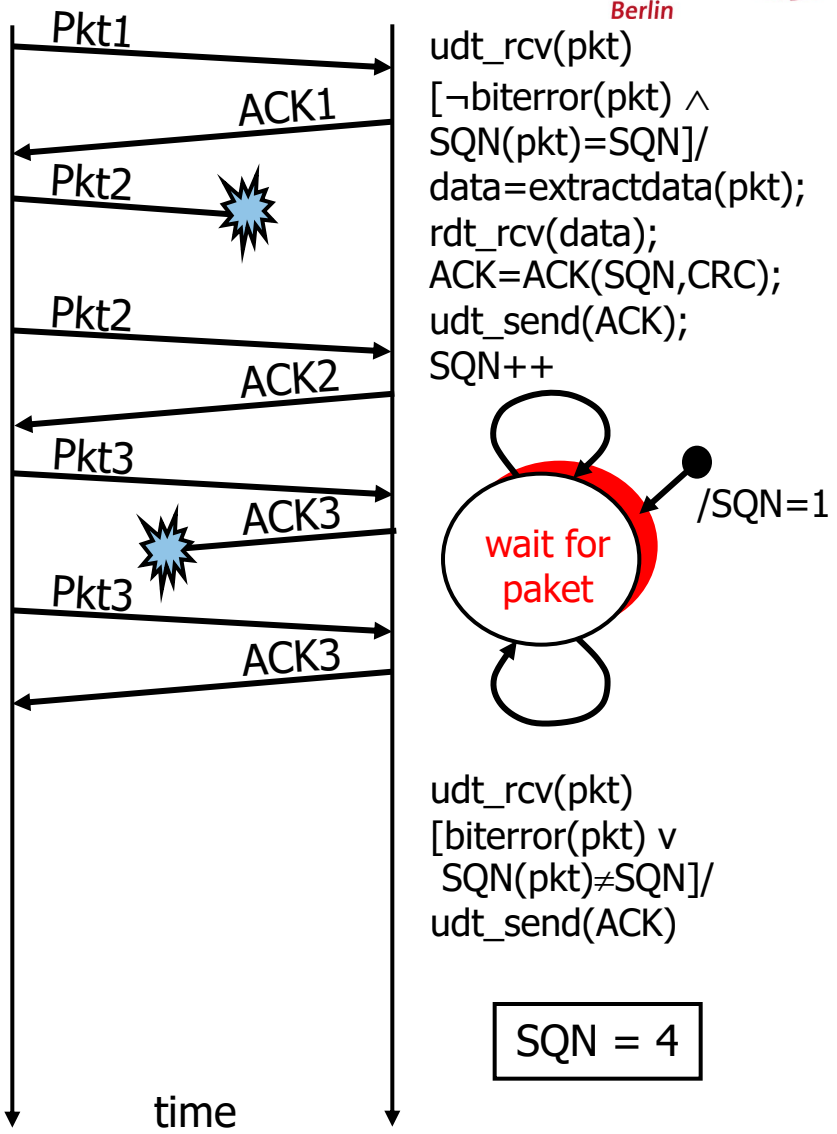
# Stop-and-Wait: loss of ACK



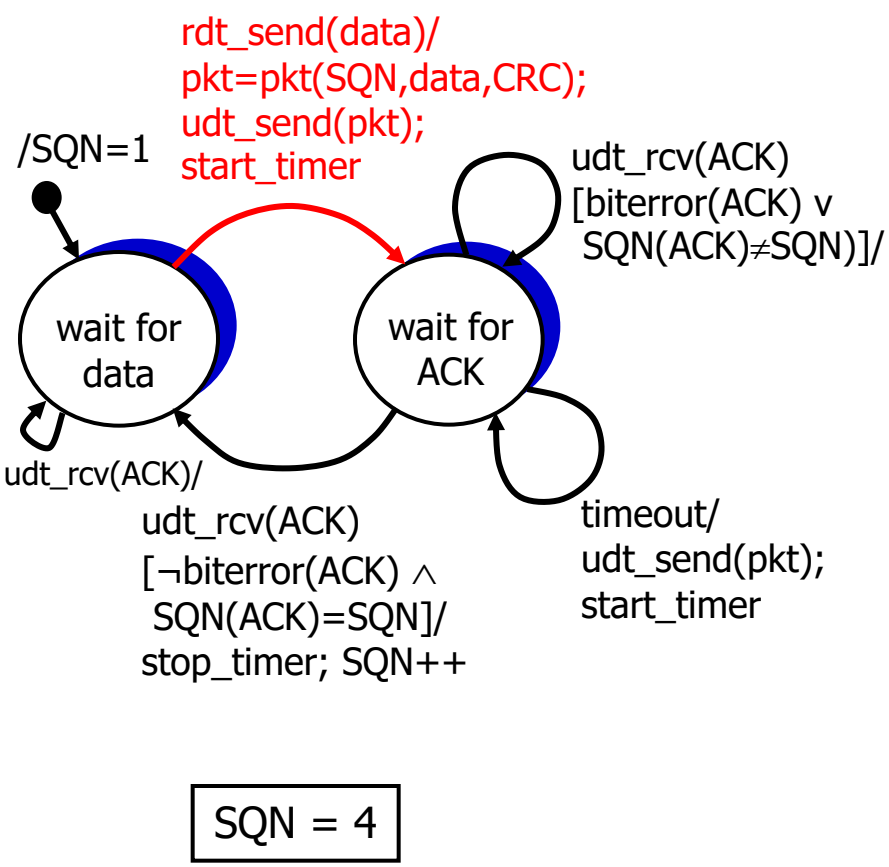



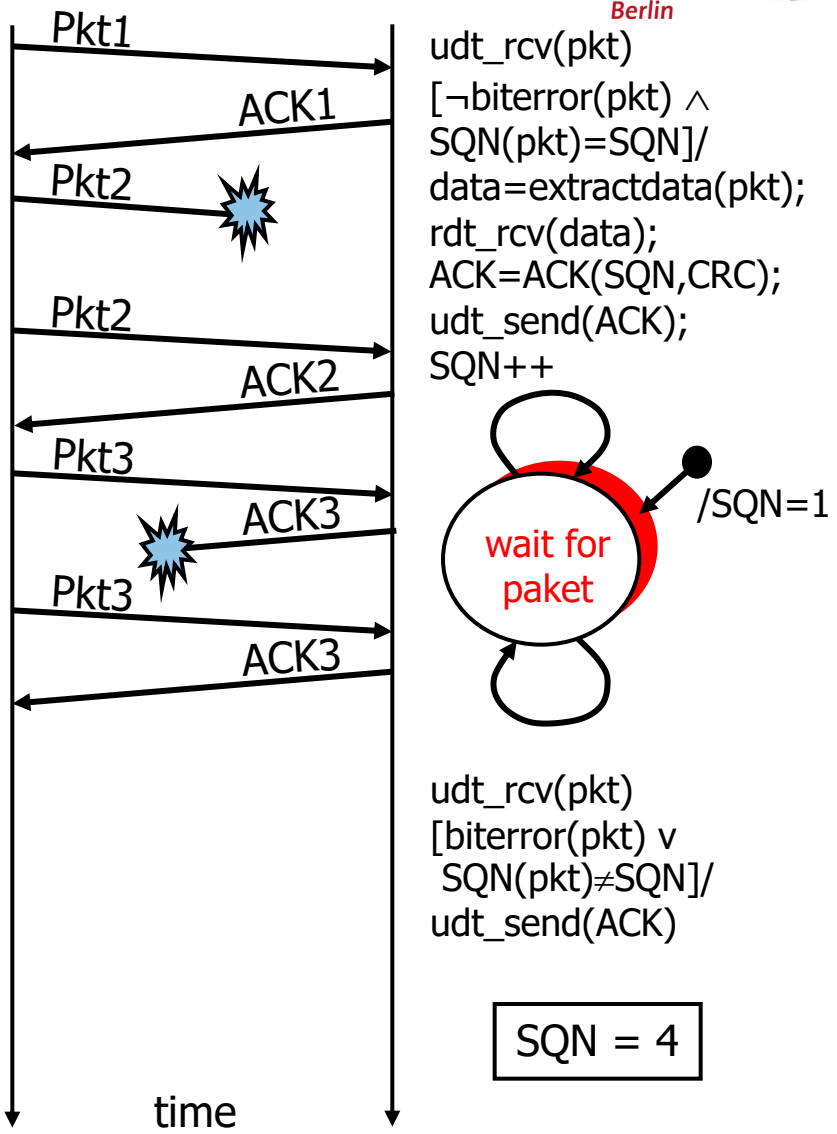
# Stop-and-Wait: loss of ACK



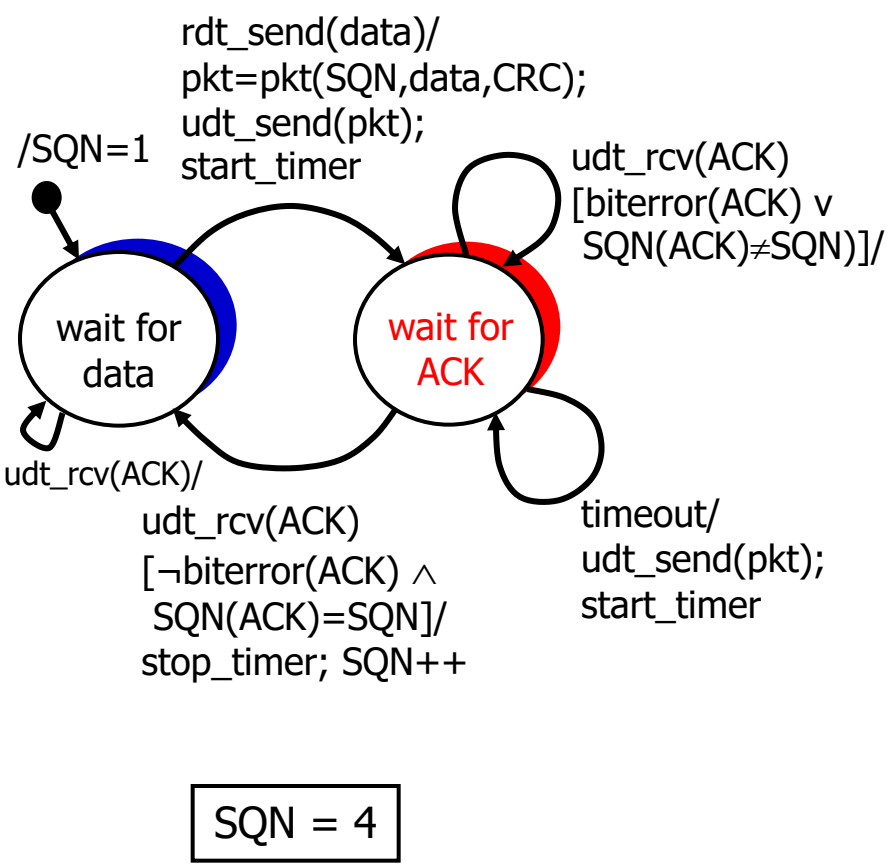



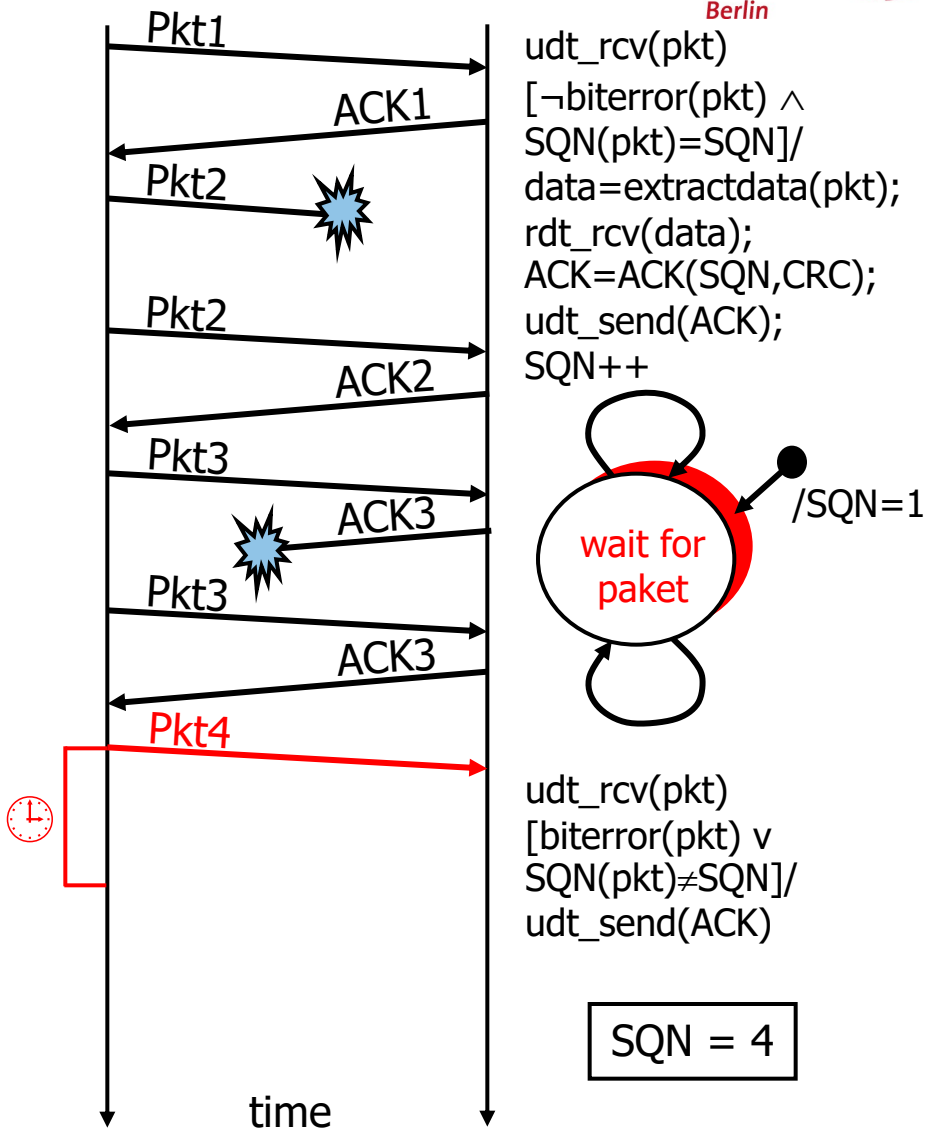
# Stop-and-Wait: delayed ACK



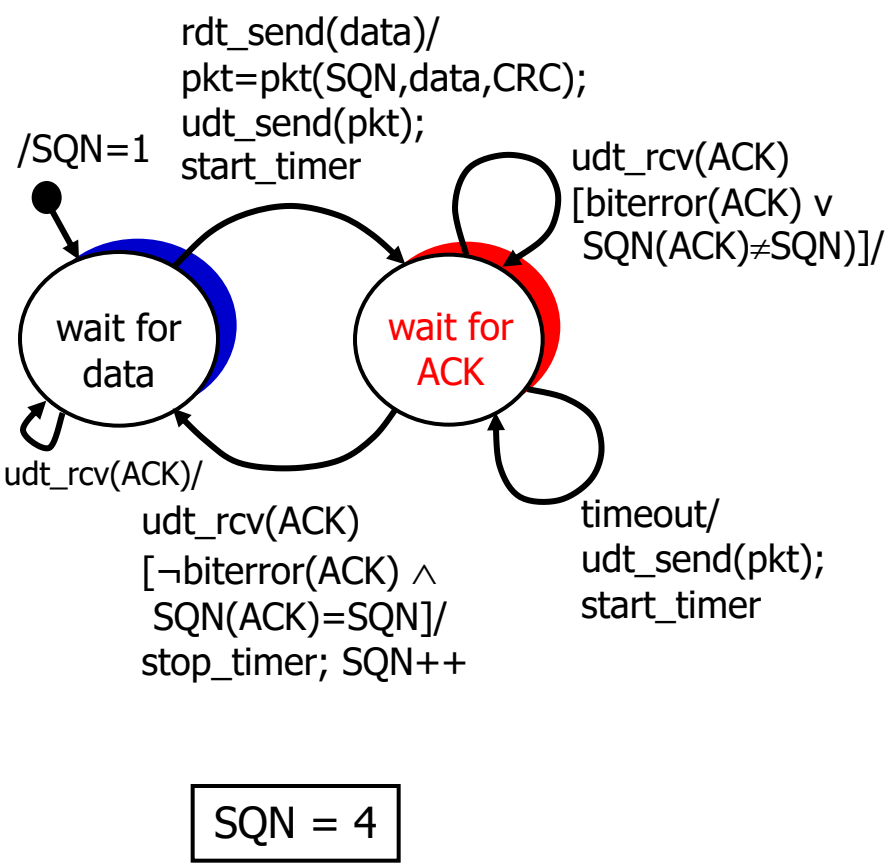



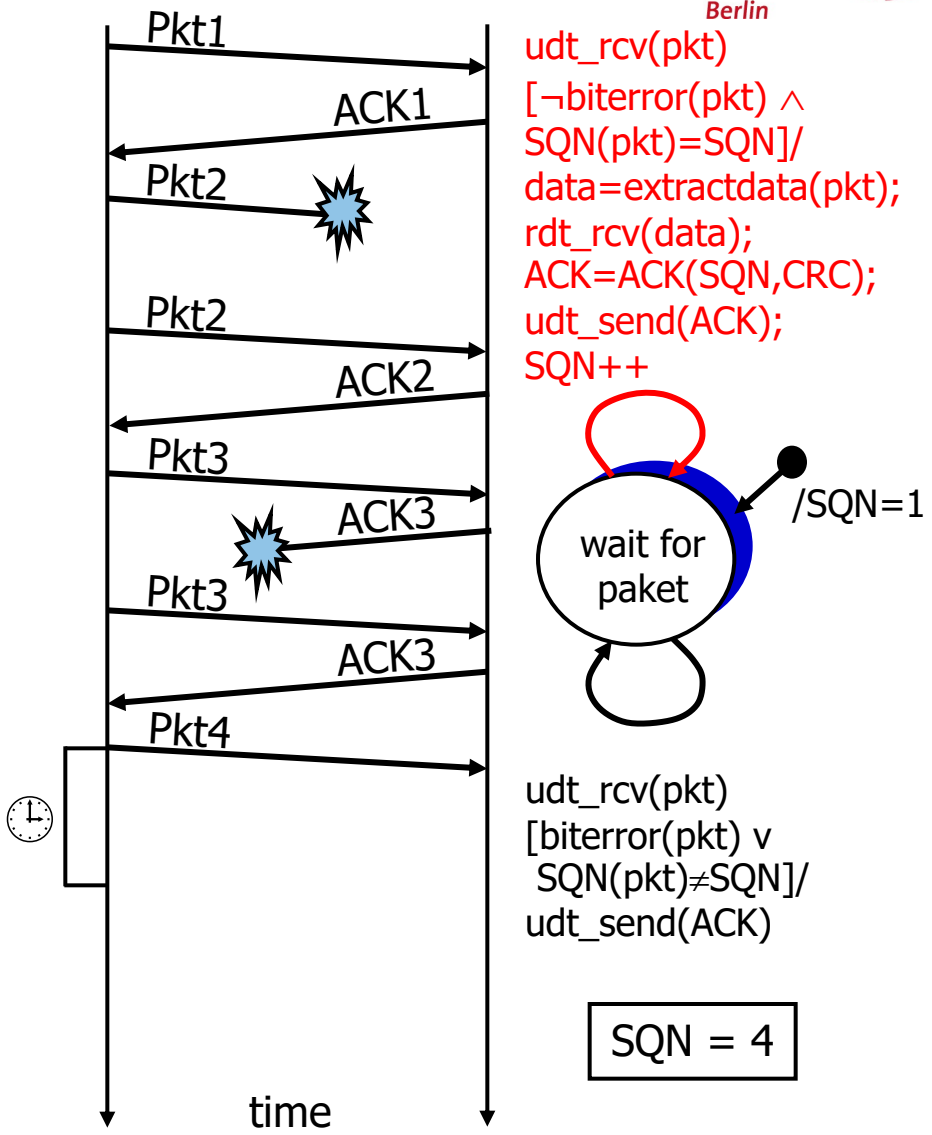
# Stop-and-Wait: delayed ACK



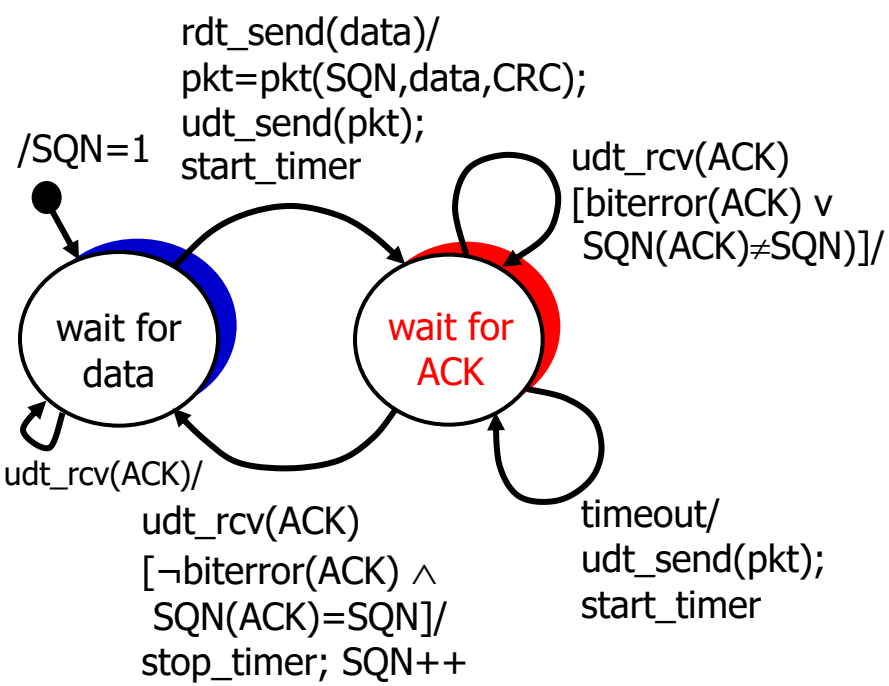



# Stop-and-Wait: delayed ACK

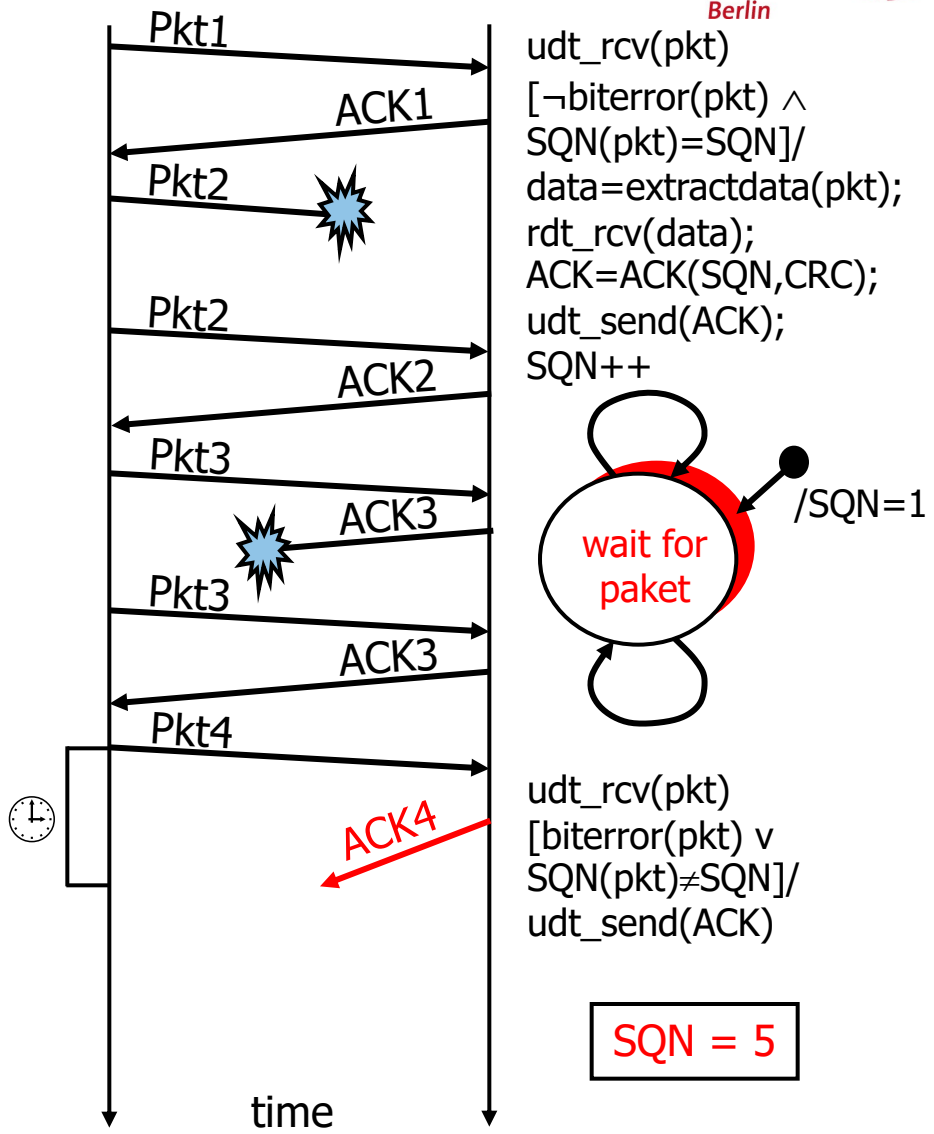


# Stop-and-Wait: delayed ACK

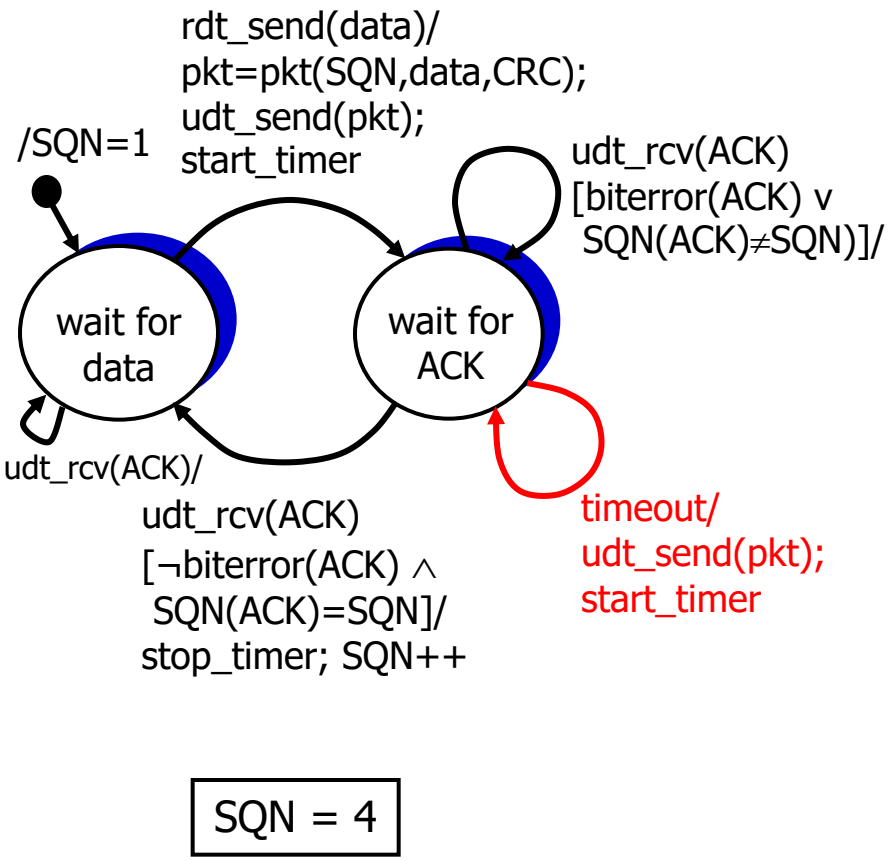



SQN = 4

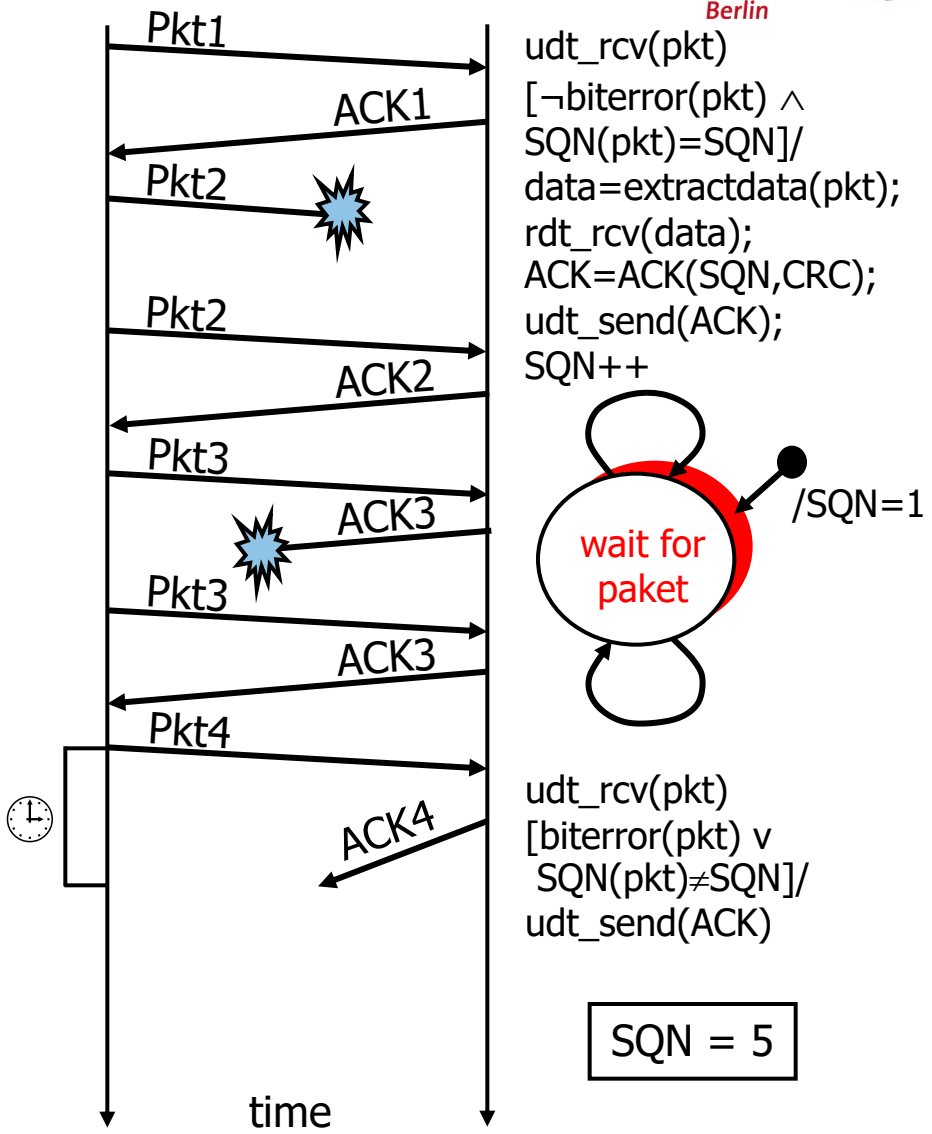



SQN = 5

# Stop-and-Wait: delayed ACK

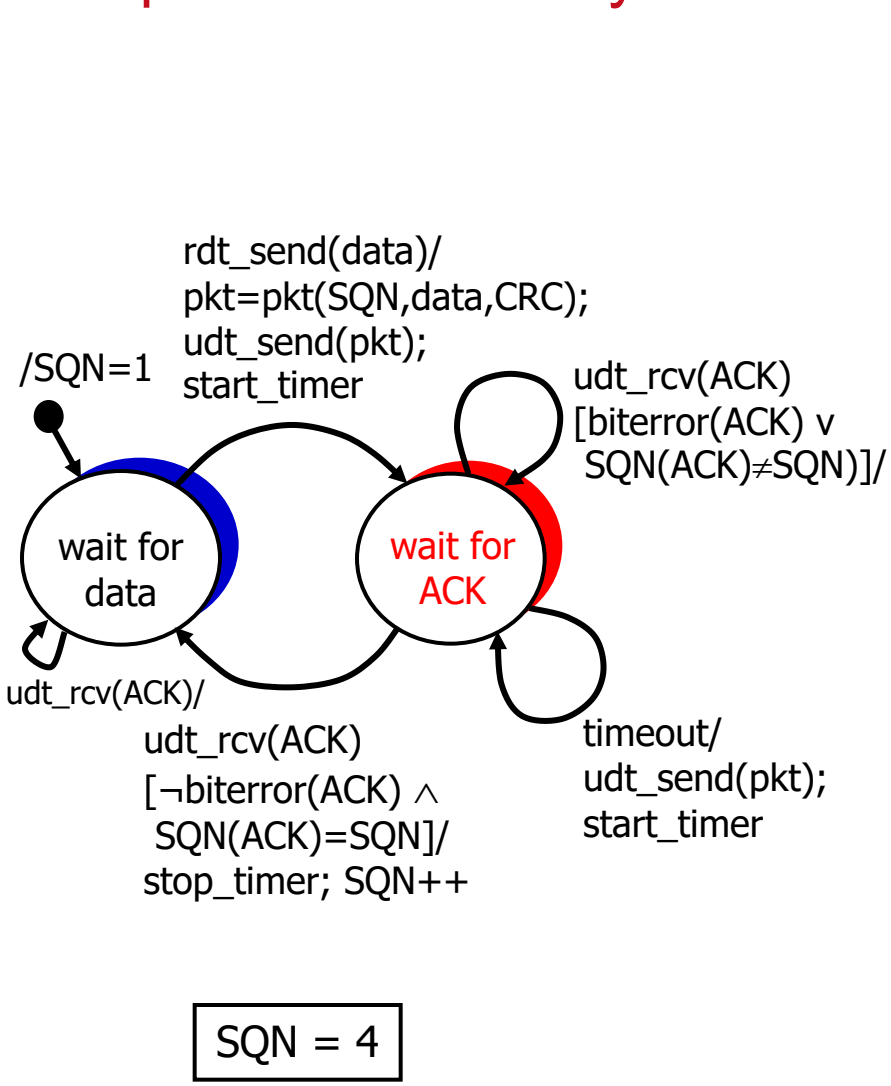


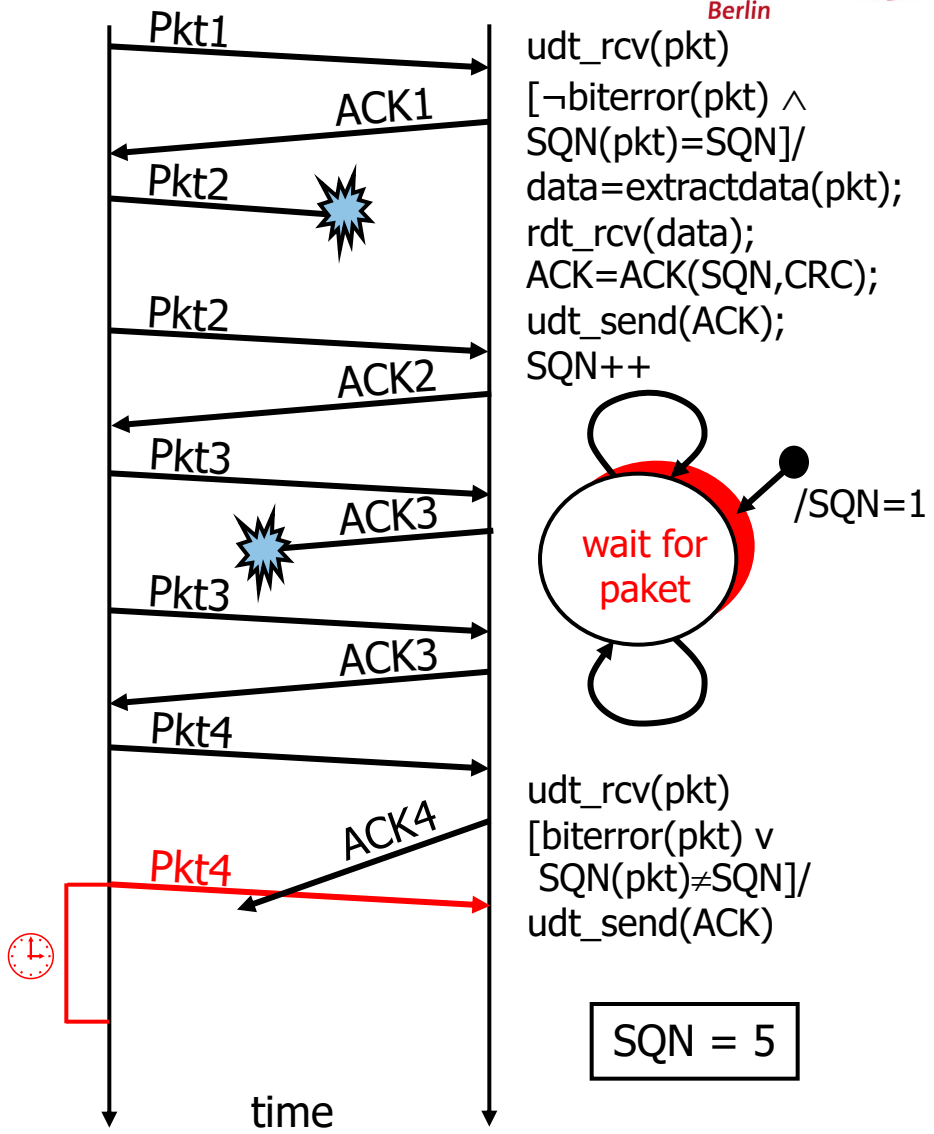

SQN = 4



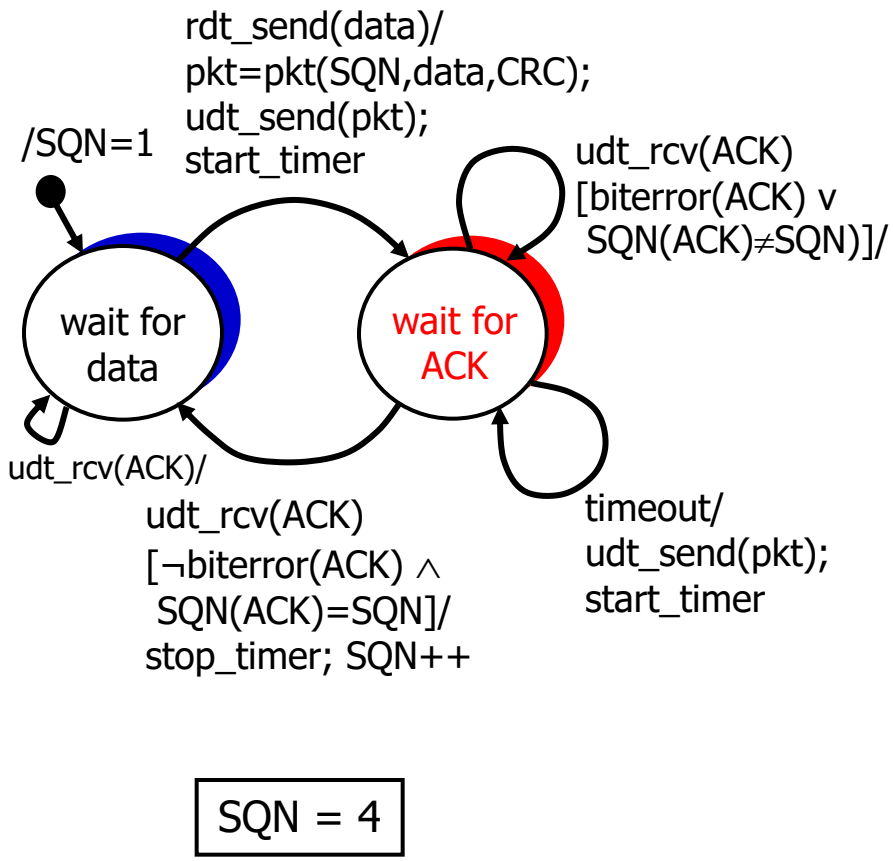

SQN = 5

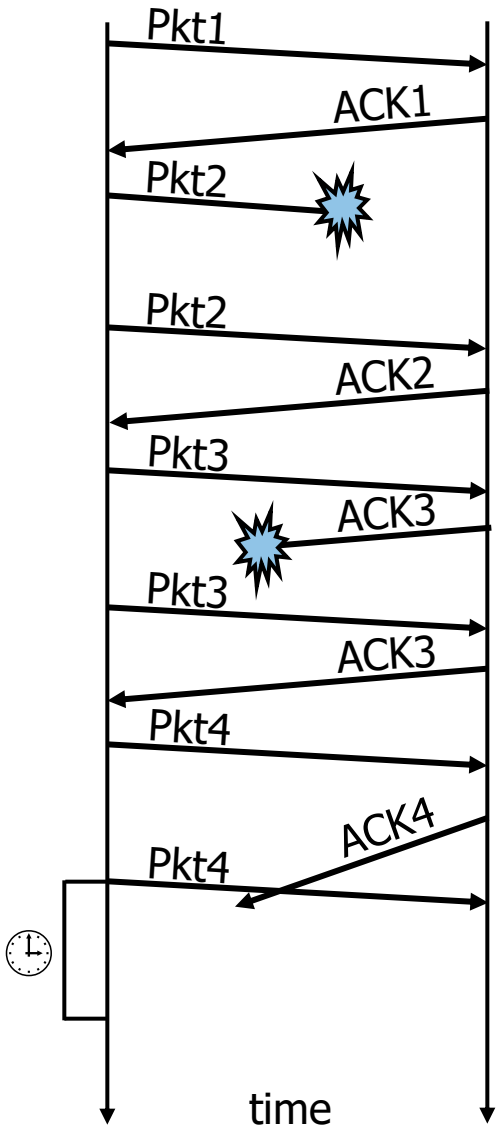
# Stop-and-Wait: delayed ACK

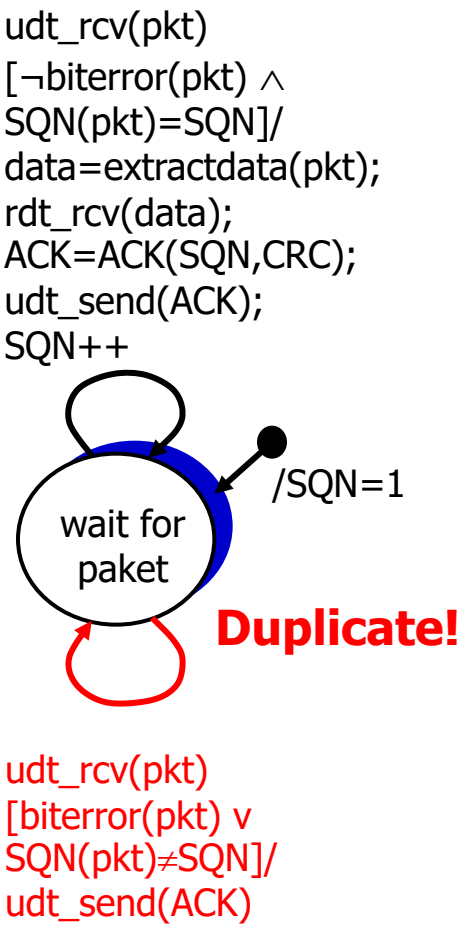


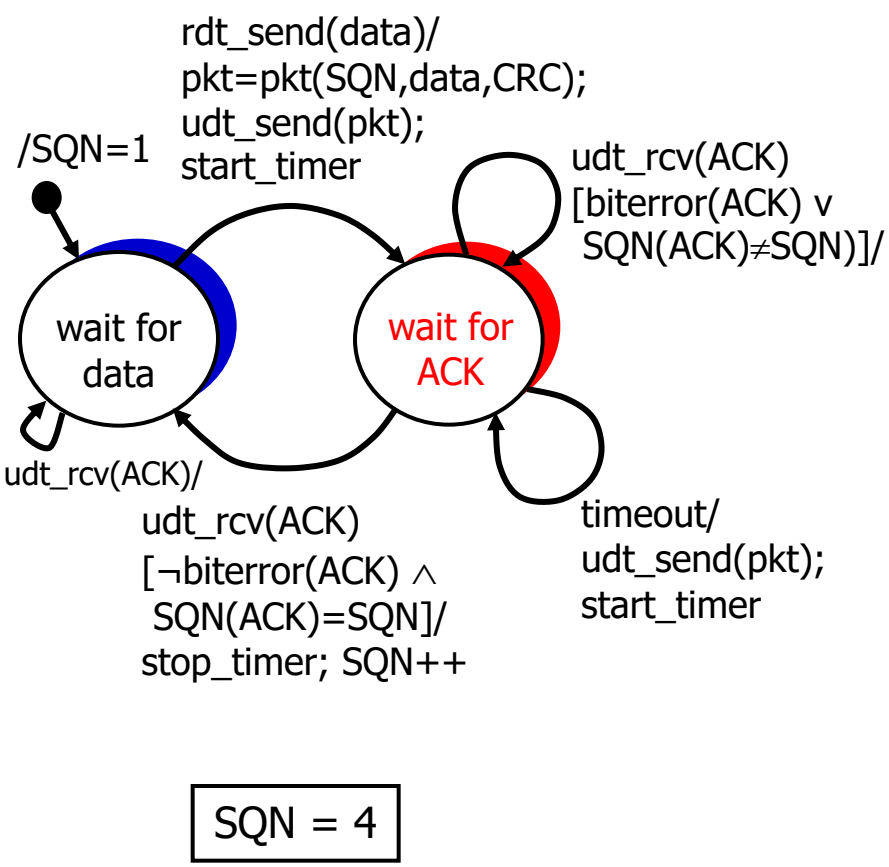
# Stop-and-Wait: delayed ACK

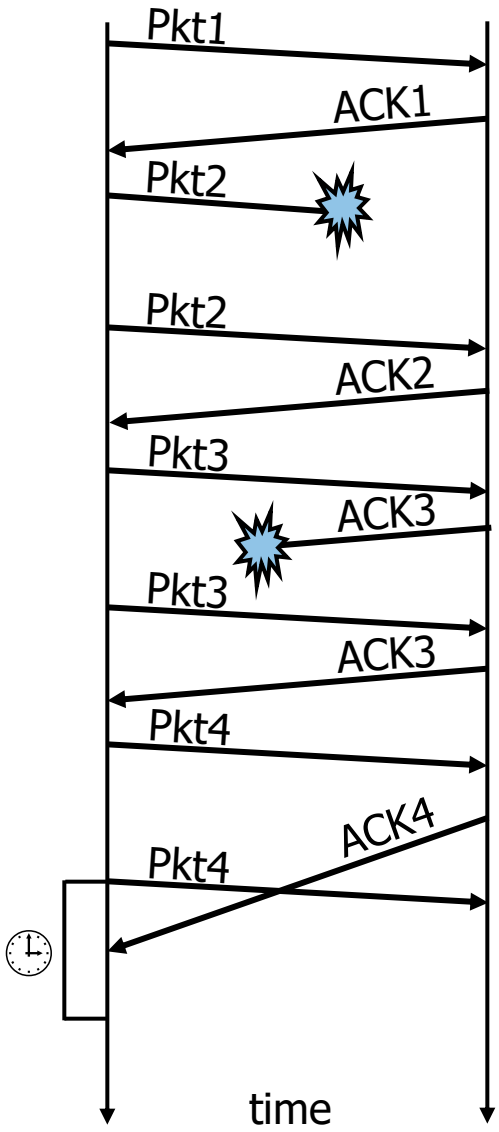


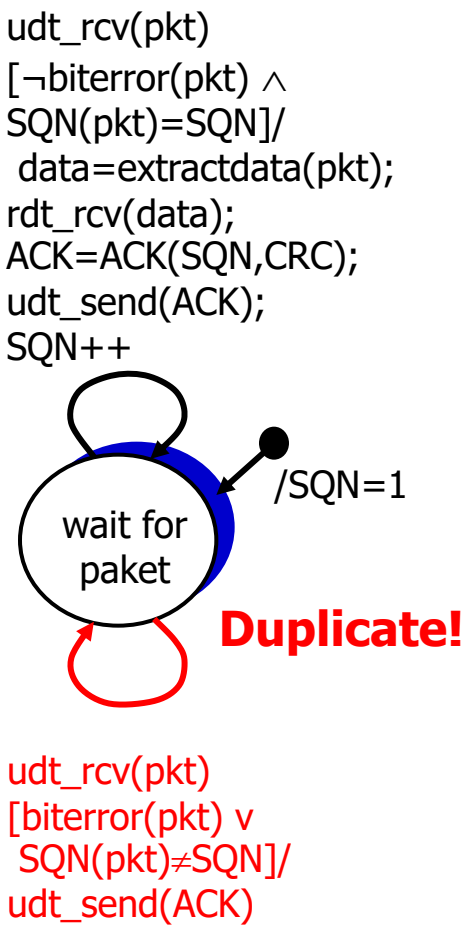



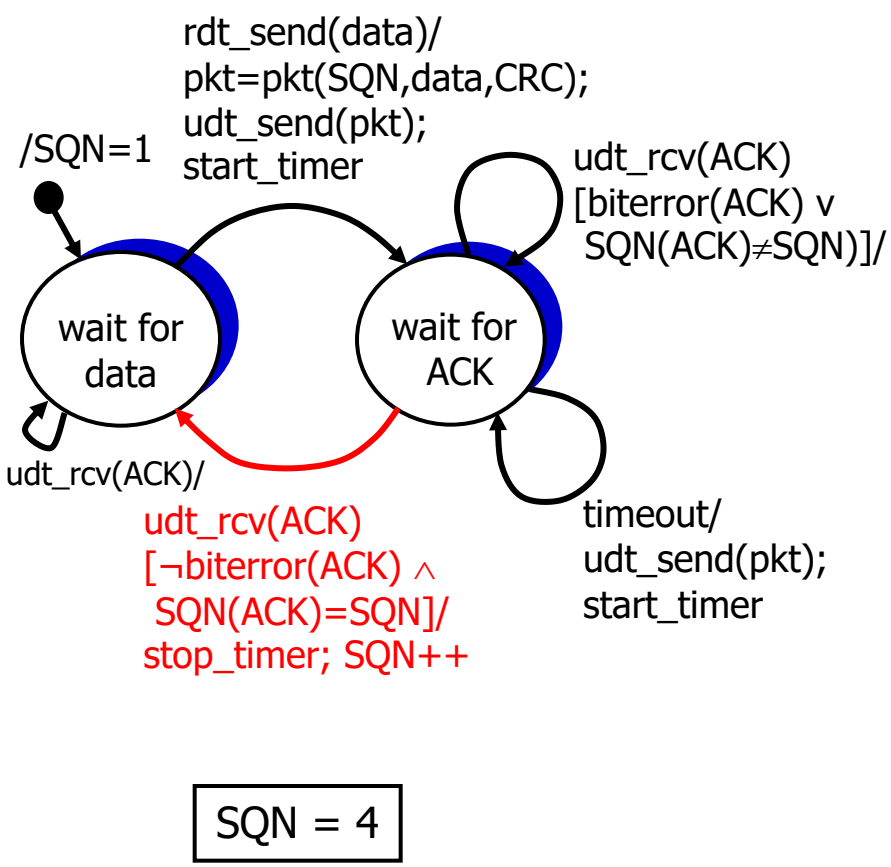
# Stop-and-Wait: delayed ACK



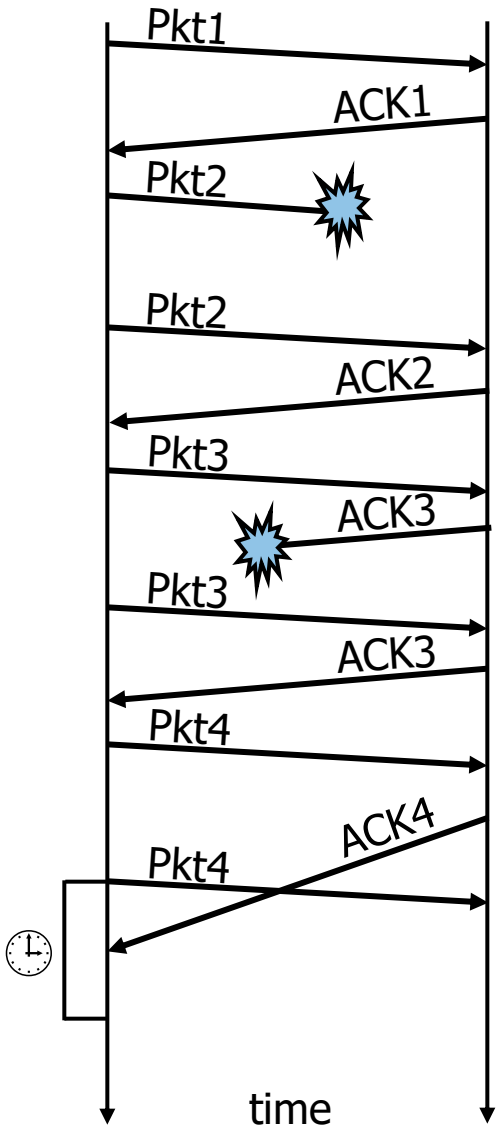



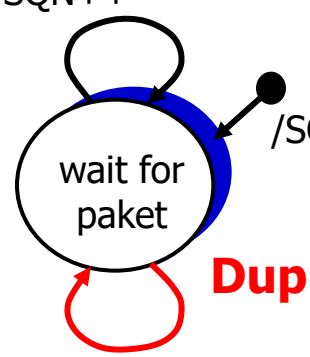
# Stop-and-Wait: delayed ACK




SQN = 4




udt_rcv(pkt)
[¬biterror(pkt) ∧
SQN(pkt)=SQN]/
data=extractdata(pkt);
rdt_rcv(data);
ACK=ACK(SQN,CRC);
udt_send(ACK);
SQN++

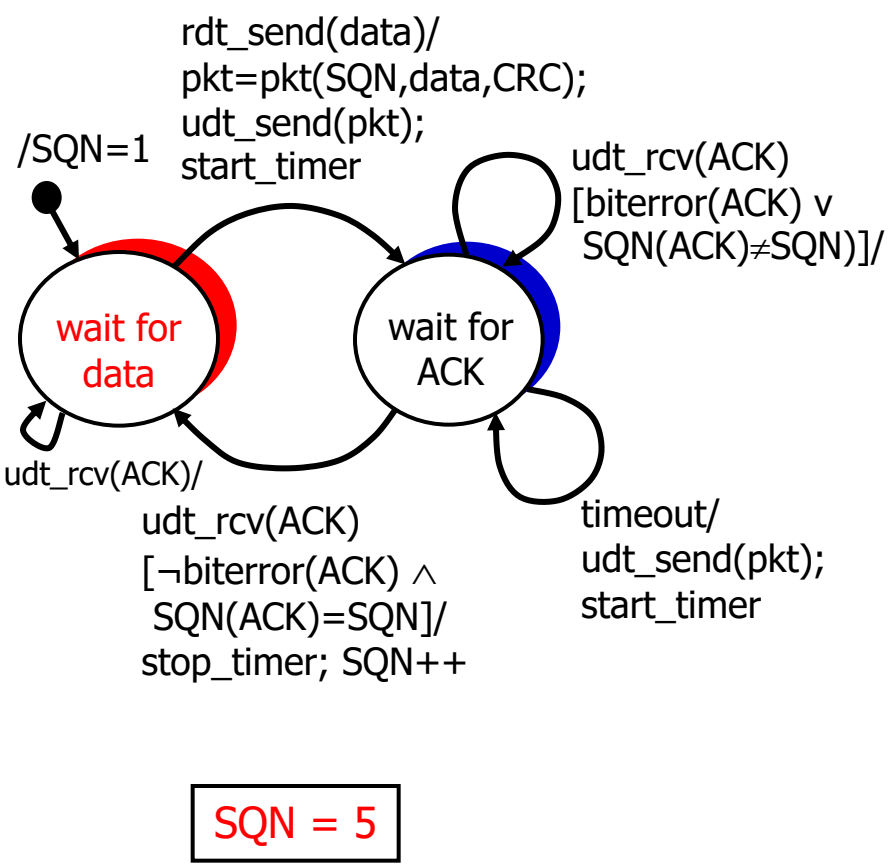


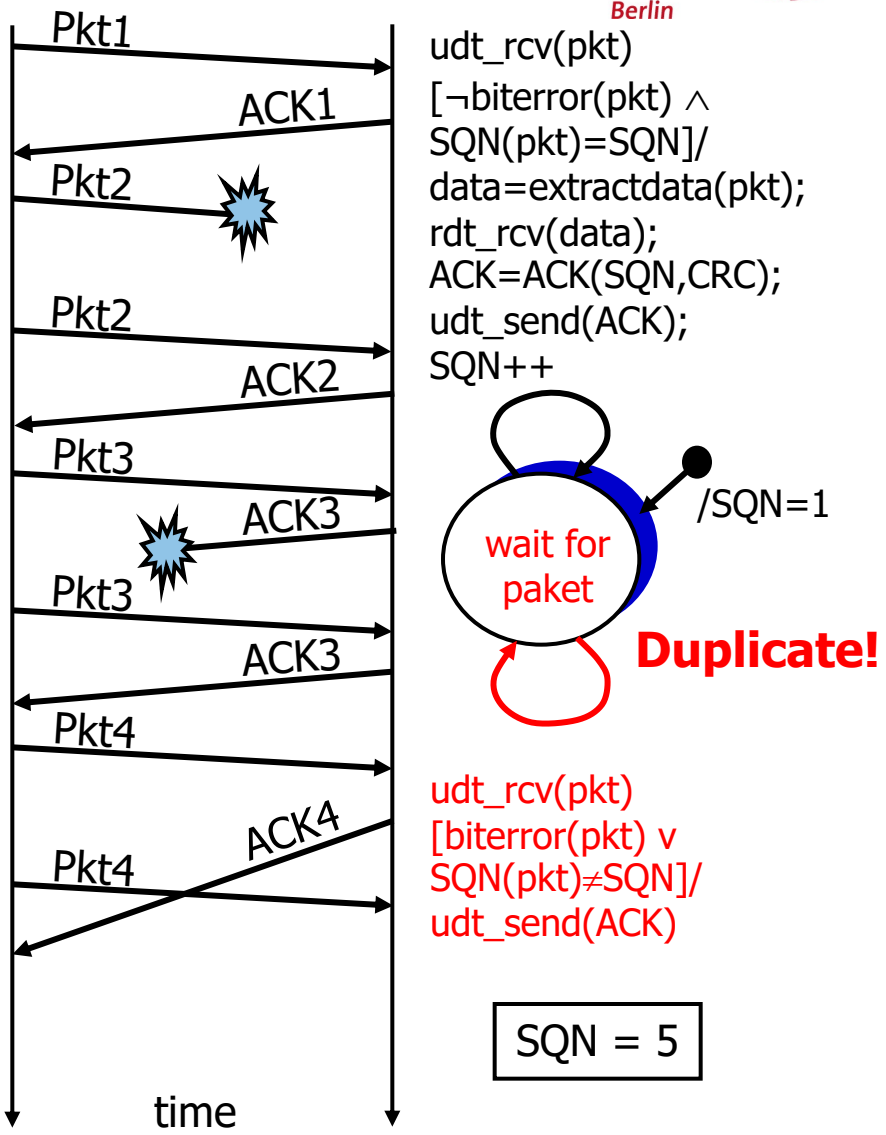

**Duplicate!**

udt_rcv(pkt)
[biterror(pkt) v
SQN(pkt)≠SQN]/
udt_send(ACK)

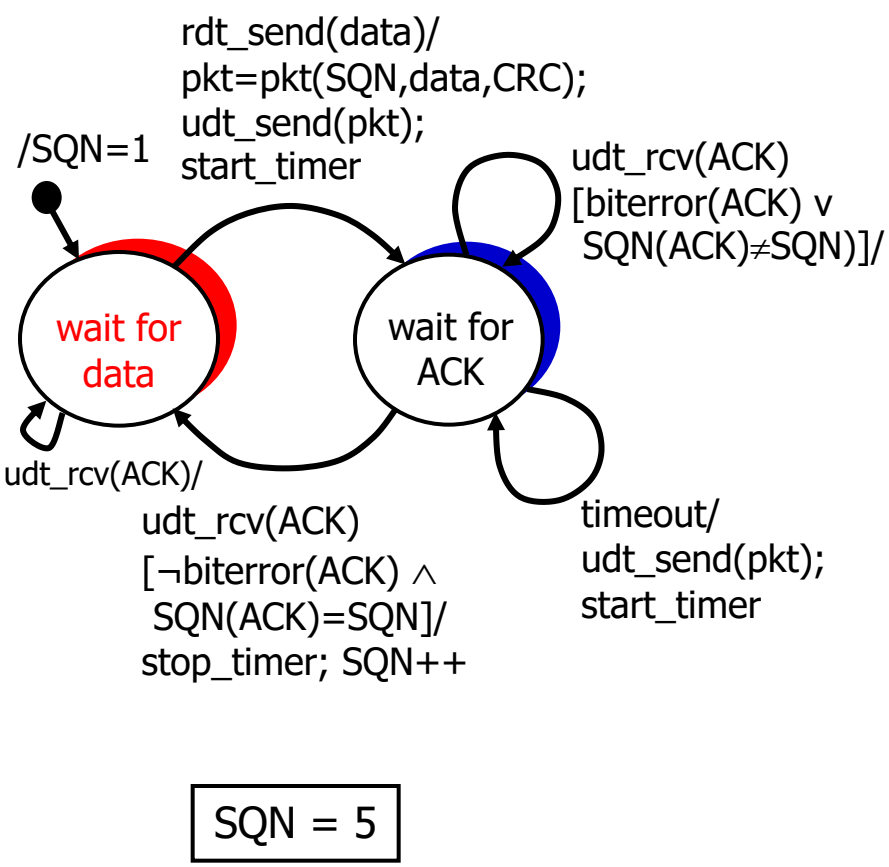
SQN = 5

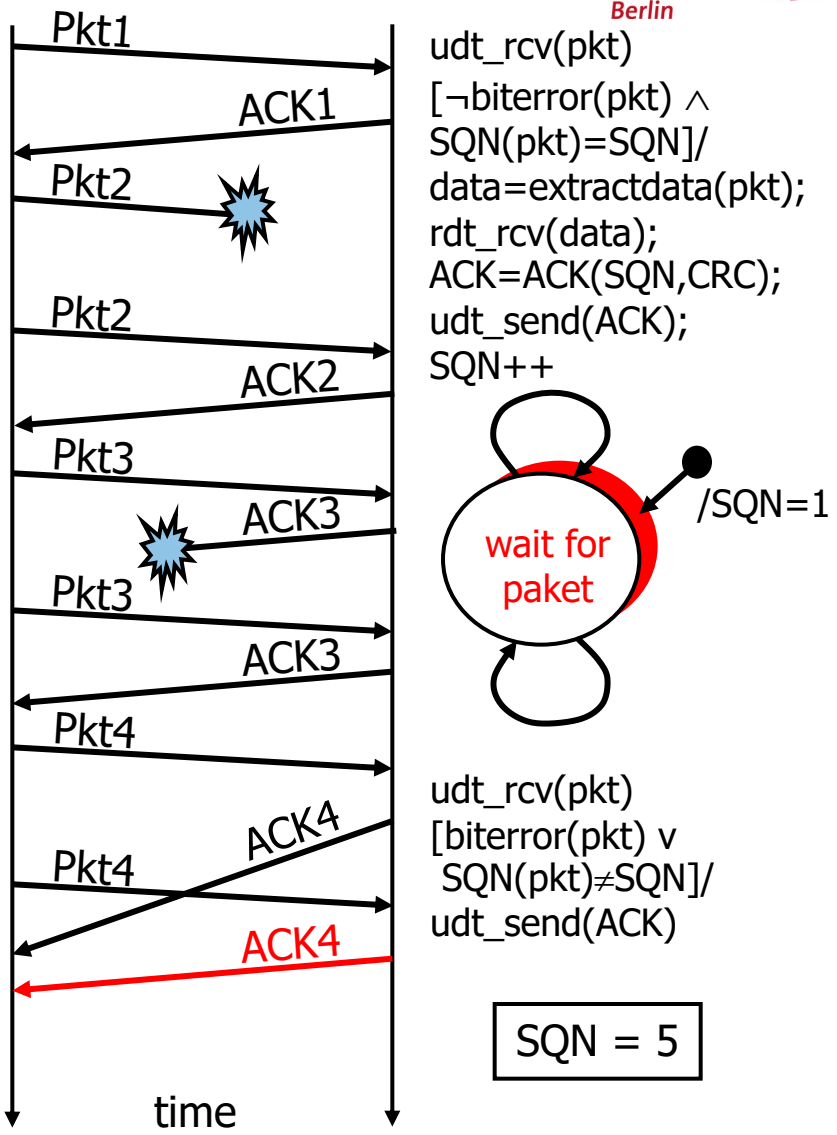
# Stop-and-Wait: delayed ACK



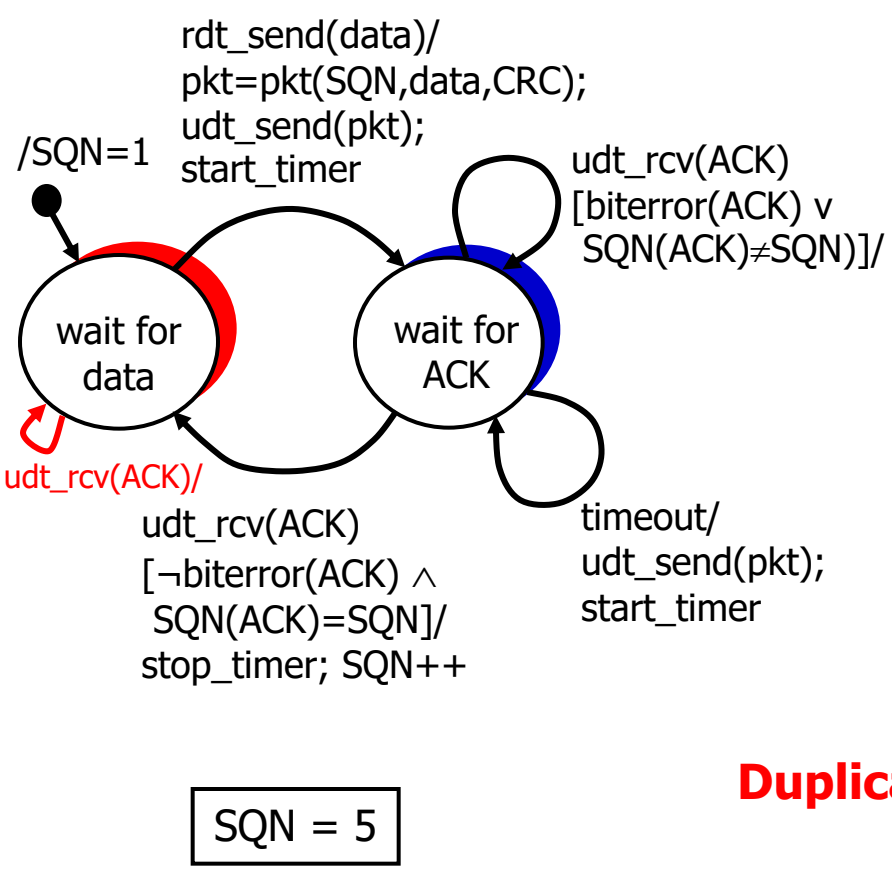



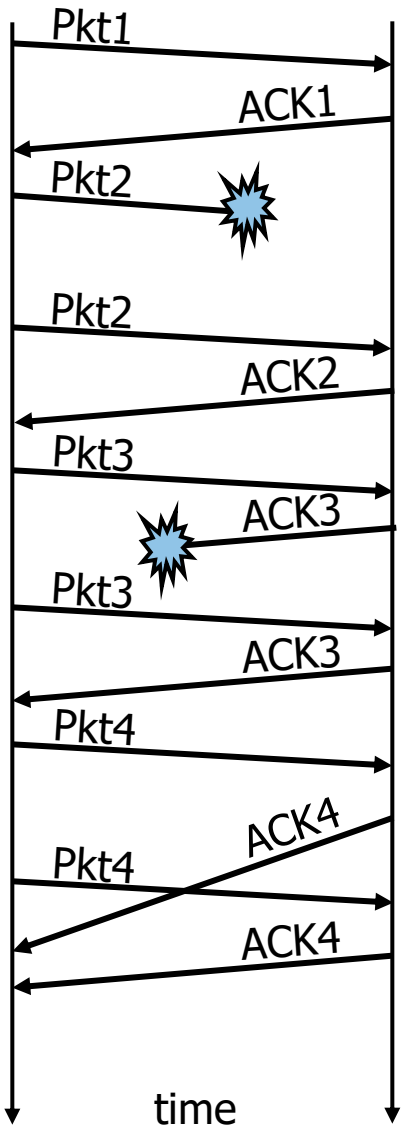
# Stop-and-Wait: delayed ACK



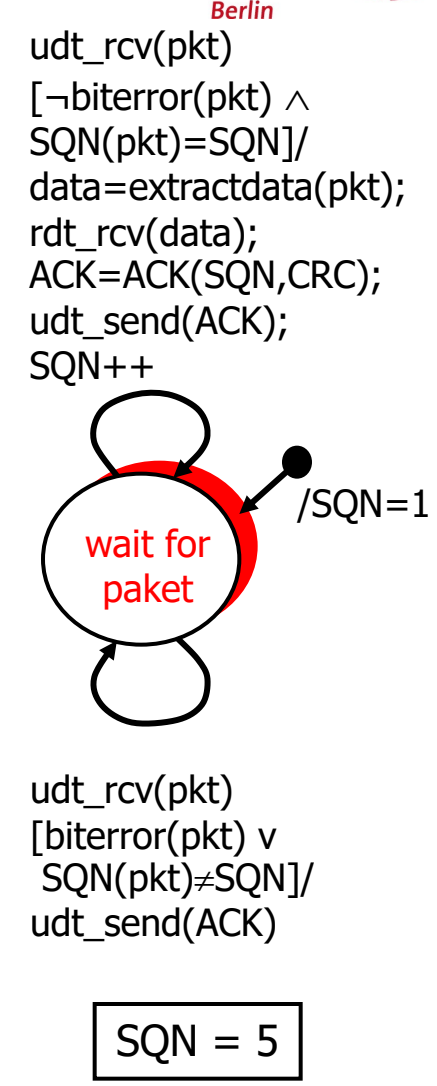



# Stop-and-Wait: delayed ACK




**Duplicate!**



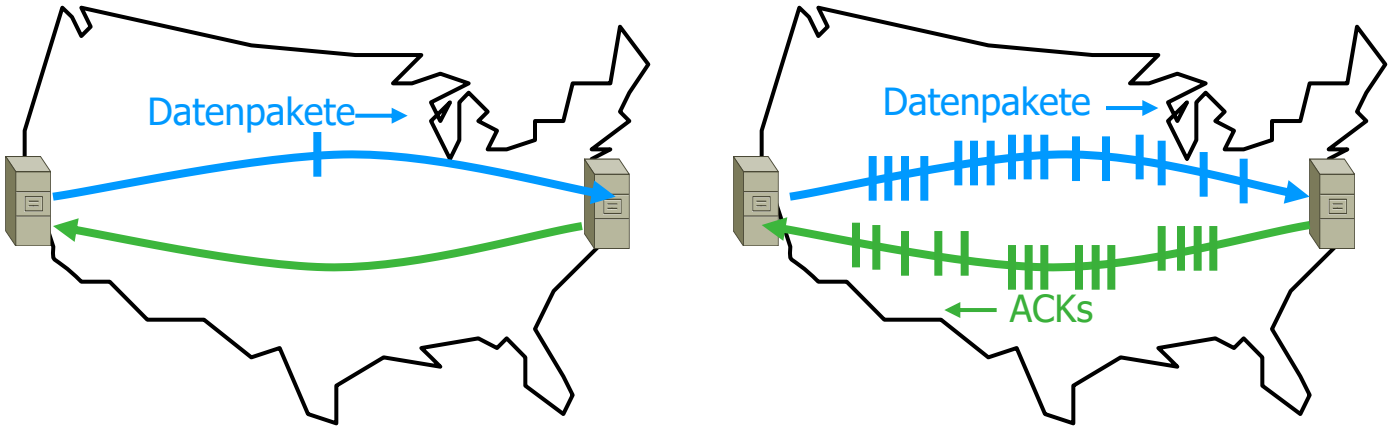
# Stop-and-Wait

- Sequence numbers
  - Every implementation uses finite sequence numbers: a variable of n bit allows the use of $2^n$ sequence numbers
  - Cyclic repeated use
  - For Stop-and-Wait, one bit is sufficient to represent two sequence numbers (0 and 1)
  - Stop-and–Wait using 0 and 1 as sequence numbers is also known as **alternating-bit-protocol**

# Sliding window protocols

# Go-Back-N and Selective Repeat

- Stop-and-Wait is inefficient for large bandwidth-delay-products
- **Sliding window protocols** send multiple packets before requiring an ACK




Stop-and-Wait

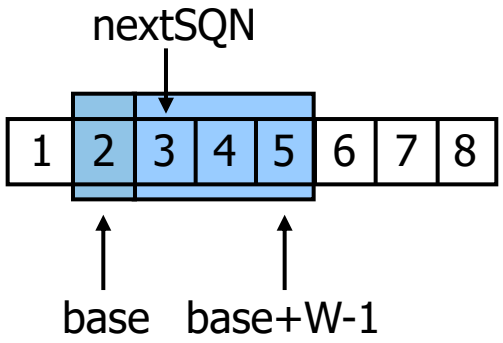
Sliding window protocol

# Go-Back-N

- Principles
  - Sender can send multiple packets (up to some maximum) before receiving the first ACK
  - A timer is started after the transmission of the first packet
  - All non-acknowledged packets are buffered
  - After timeout, **all unacknowledged packets are sent again**
  - The receiver sends **cumulative ACKs**: each ACK acknowledges all packets up to and including the current SQN
  - The receiver accepts packets only in the correct order and **does not need a buffer**

# Go-Back-N

- Send buffer

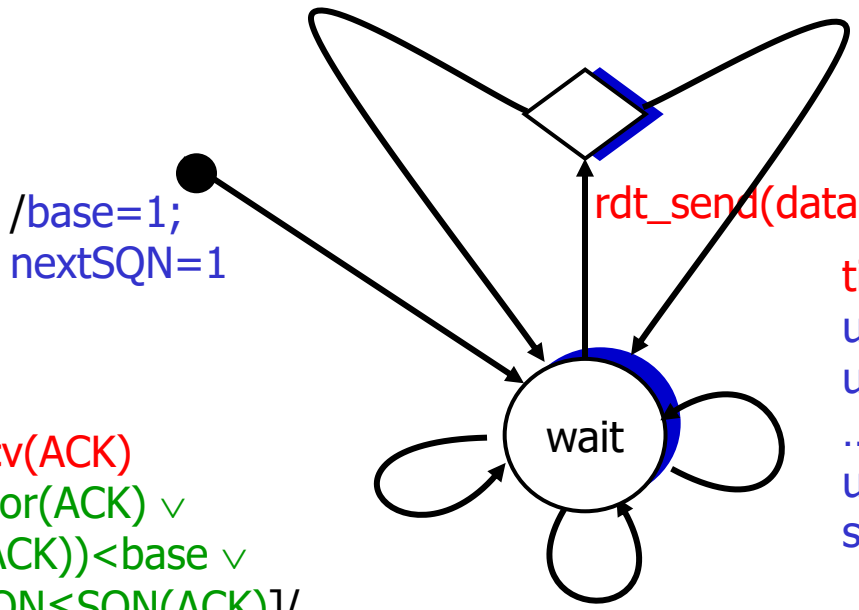



  - base: SQN of the oldest unacknowledged packet
  - nextSQN: SQN of the next packet to be transmitted
  - W: window size, number of packets that can be sent before ACK
  - Window [base, base+W-1] is constantly moved from left to right; thanks to cumulative ACKS, the structure remains unchanged:
    - [base, nextSQN-1]: packets that have been sent but not yet acknowledged
    - [nextSQN, base+W-1]: packets that have not yet been sent but can be sent before next ACK

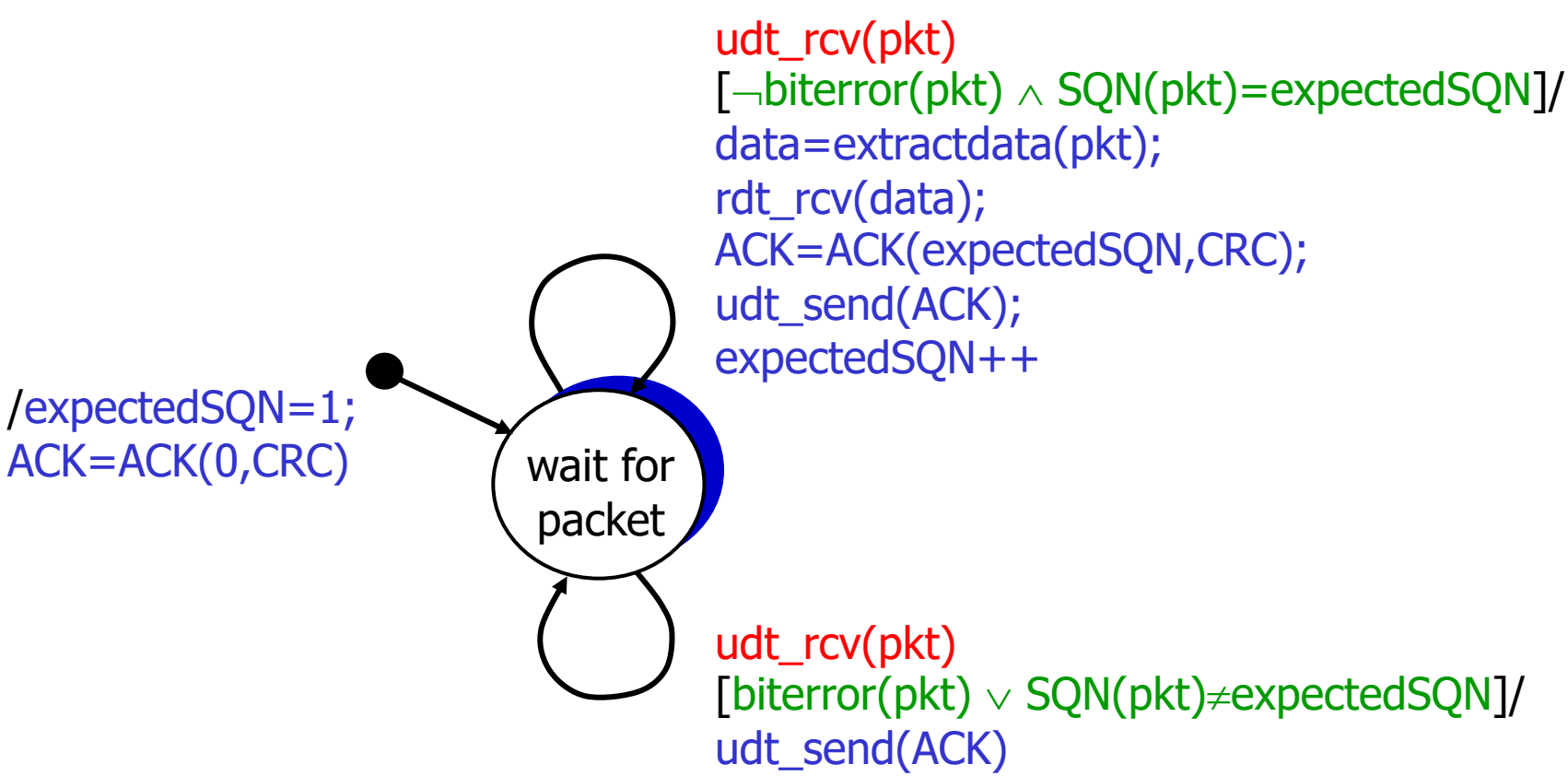
# Go-Back-N

- Informal description
  - Sender
    1. If data available and the window is not full: send packet with nextSQN, increment nextSQN, if this was the first packet in the window, start timer
    2. If ACK is in window, move window to this SQN; if the window is now empty, stop timer; otherwise, re-start timer
    3. If timeout, send all packets in window again, restart timer
  - Receiver
    - If packet is correct contains next expected SQN, send ACK with current SQN, increment SQN; otherwise re-send last ACK

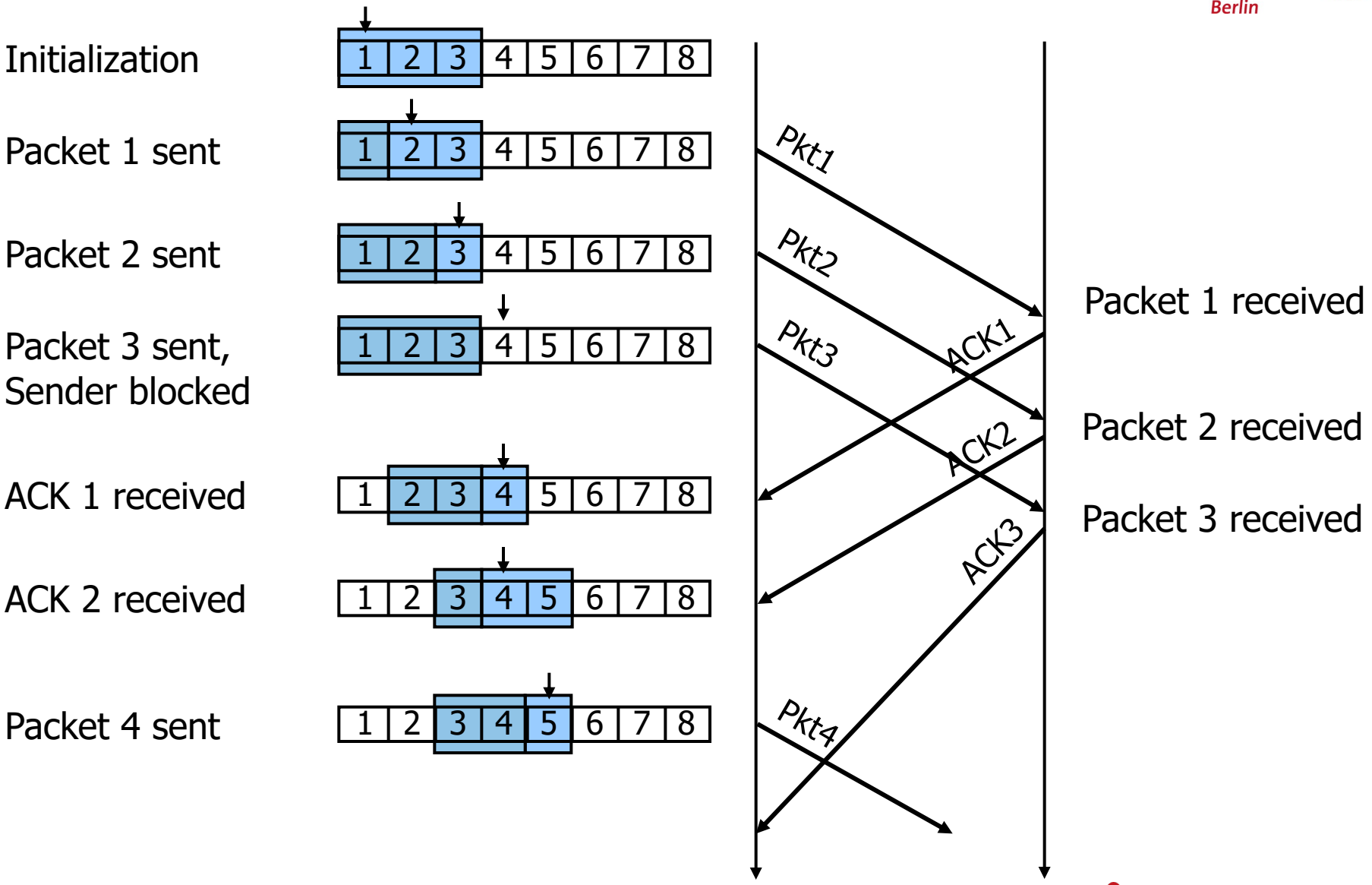
# Go-Back-N: Sender



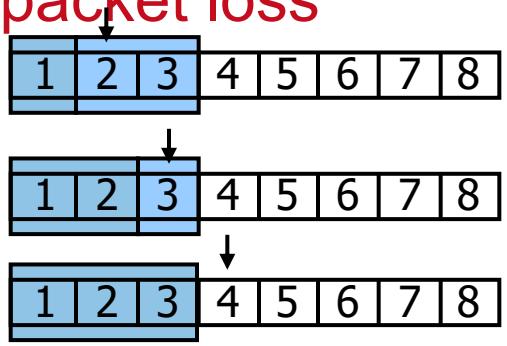
# Go-Back-N: Receiver



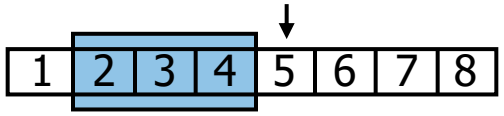
# Go-Back-N: normal execution



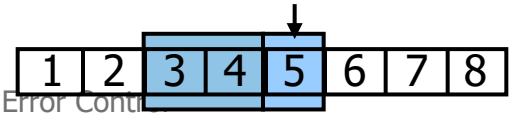
# Go-Back-N: packet loss

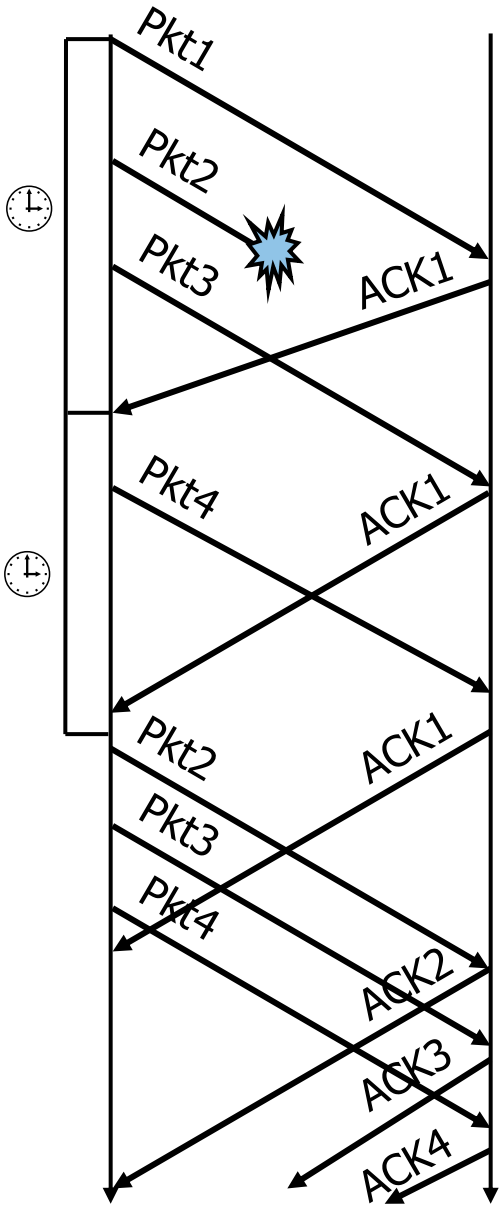



Timer neu starten




Timeout, all not yet acknowledged packets are re-transmitted



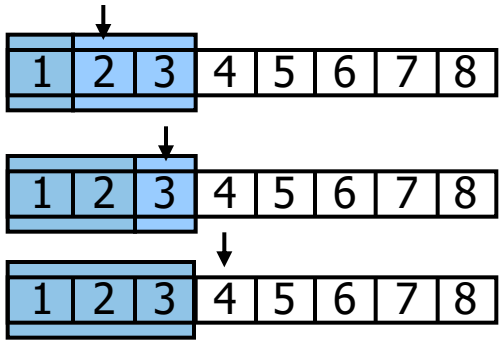

Packet 1 received

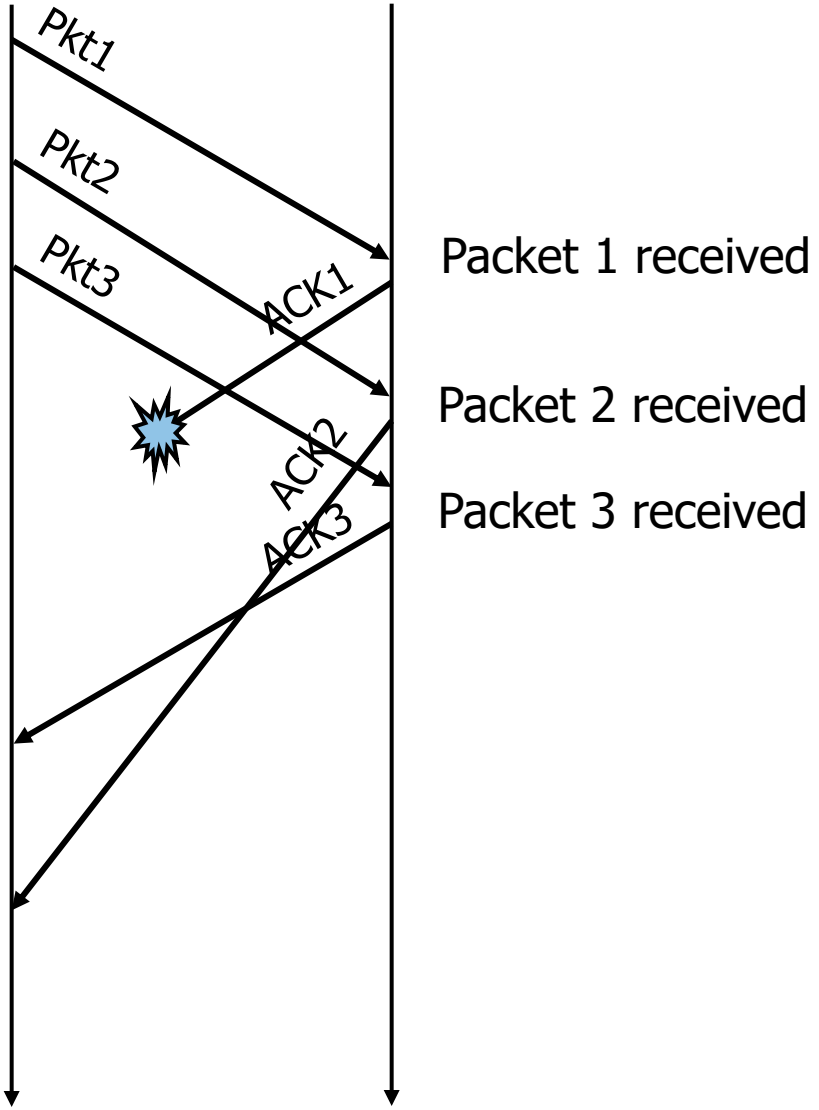
Packet 2 received

Packet 3 received

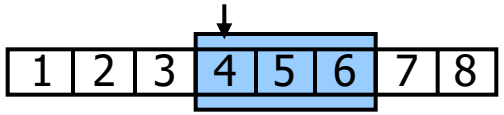
Packet 4 received

# Go-Back-N: lost and delayed ACKs



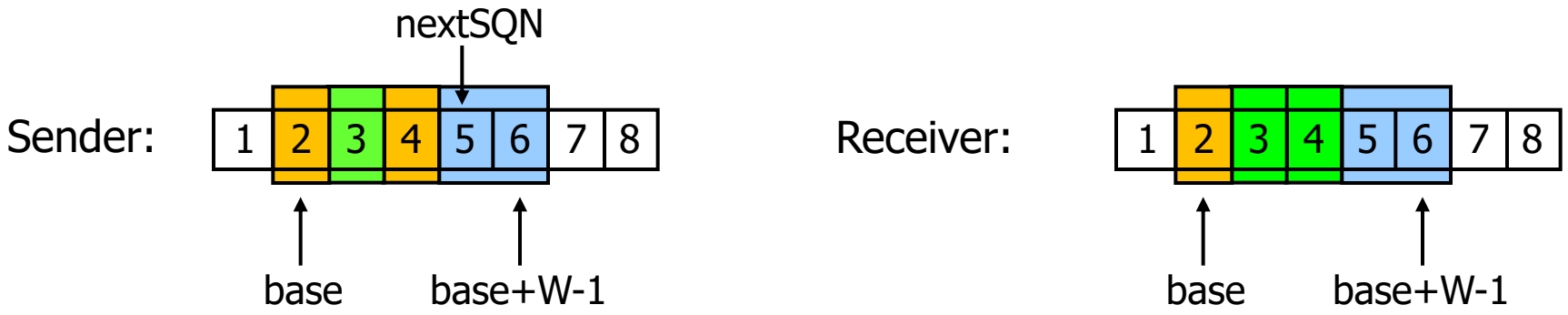

cumulative ACK to compensate loss and delay



# Selective Repeat

- Principles
  - Sender can send multiple packets (up to some maximum) before receiving the first ACK
  - A timer is started on a per-transmitted-packet-basis
  - All non-acknowledged packets are buffered
  - After timeout, **the associated packet is re-sent**
  - Receiver sends **selective ACKs**: the semantics is that the packet with the SQN in the ACK was successfully received
  - Receiver needs a **buffer** to compensate for (still) missing packets

# Selective Repeat: buffer management

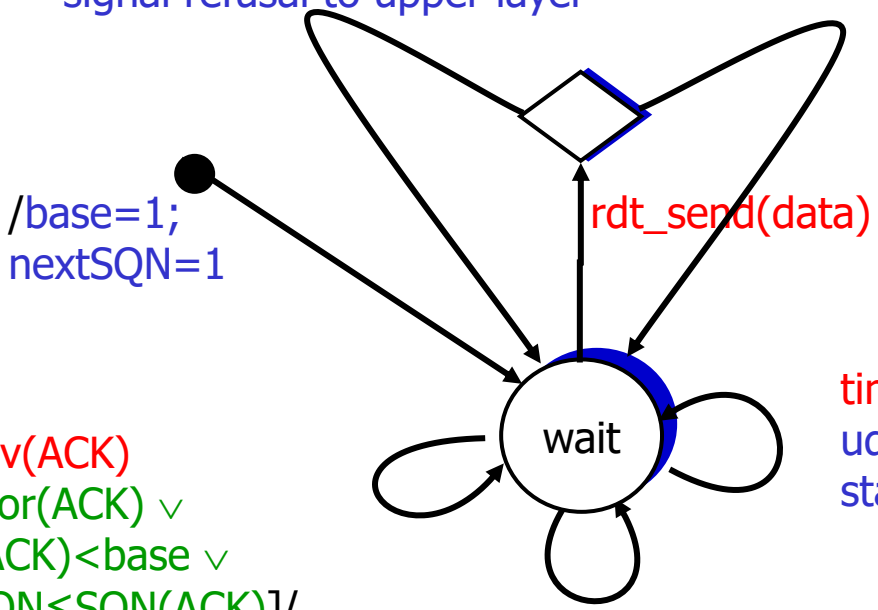



- base, nextSQN, W (as for Go-Back-N)
- Window at sender contains sent but not yet acknowledged, sent and acknowledged, and unsent packets
- Receiver buffers all received packets
- Window at receiver contains received packets, missing packets, and space for not yet received packets

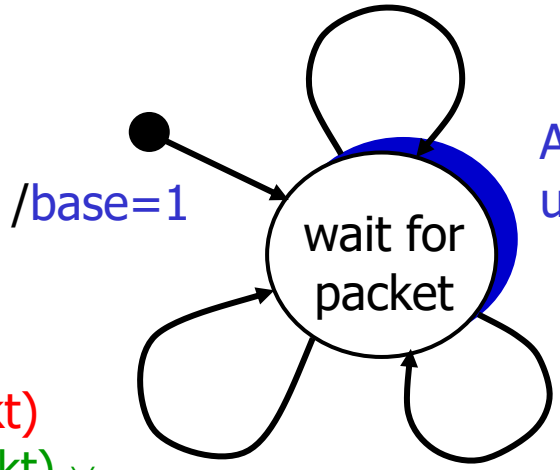
# Selective Repeat

- Informal description
  - Sender
    1. If data is available and there is space in the window: send packet, start timer for this packet, increment nextSQN
    2. If ACK is received with SQN within window, mark packet with this SQN as acknowledged, stop this timer; if possible, move sliding window
    3. If timeout, re-send the associated packet, restart timer
  - Receiver
    - If received packet is correct, SQN is within window, send ACK, buffer packet, and, if possible, move sliding window
    - If SQN of received packet is from within the previous window, re-send ACK

# Selective Repeat: Sender



# Selective Repeat: Receiver

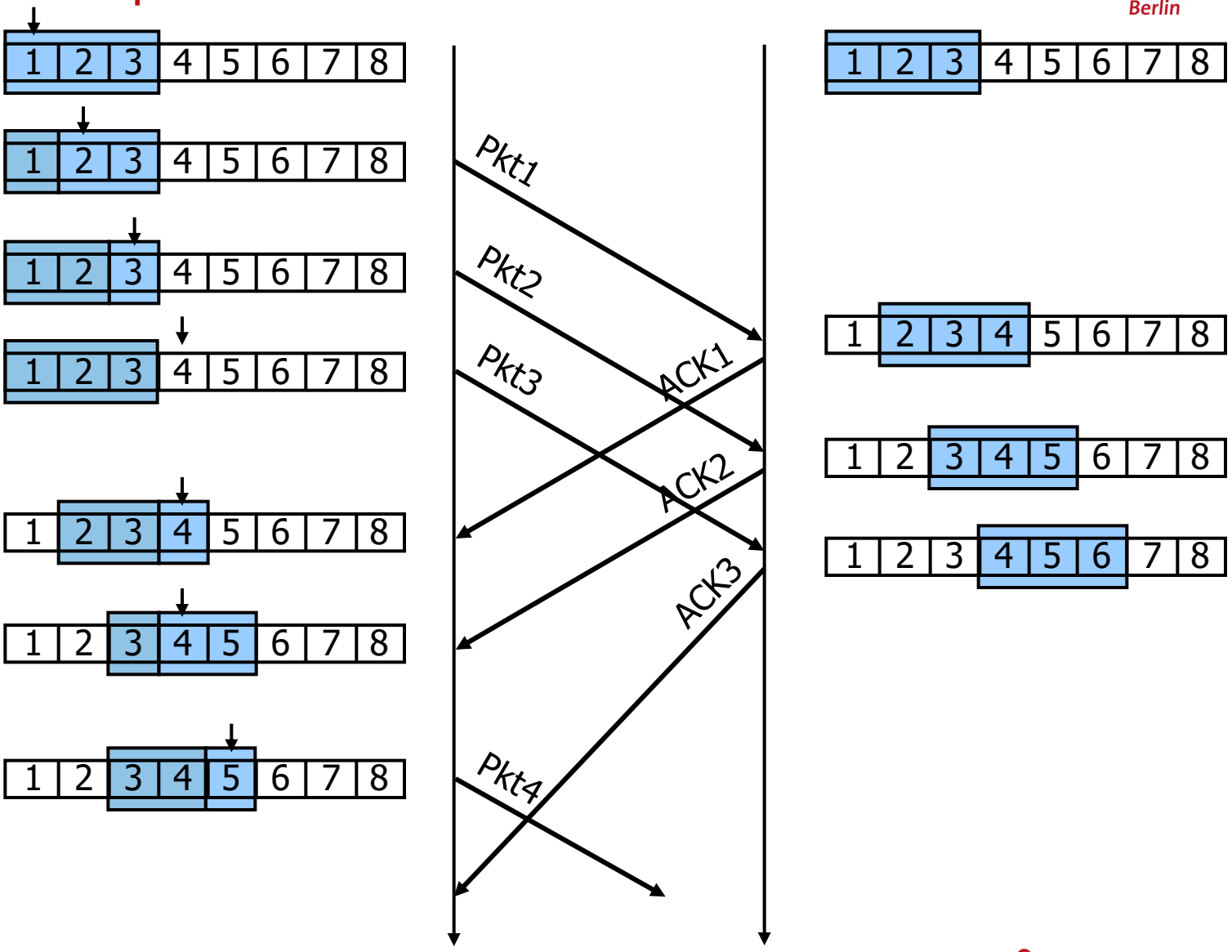



udt_rcv(pkt)
[¬biterror(pkt) ∧ SQN(pkt) in [base,base+W-1]]/
data=extractdata(pkt);
buffer data;
if SQN(pkt)=base
  base=lowest unrcvd sequence number;
  deliver all data until base-1;
ACK=ACK(SQN(pkt),CRC);
udt_send(ACK)

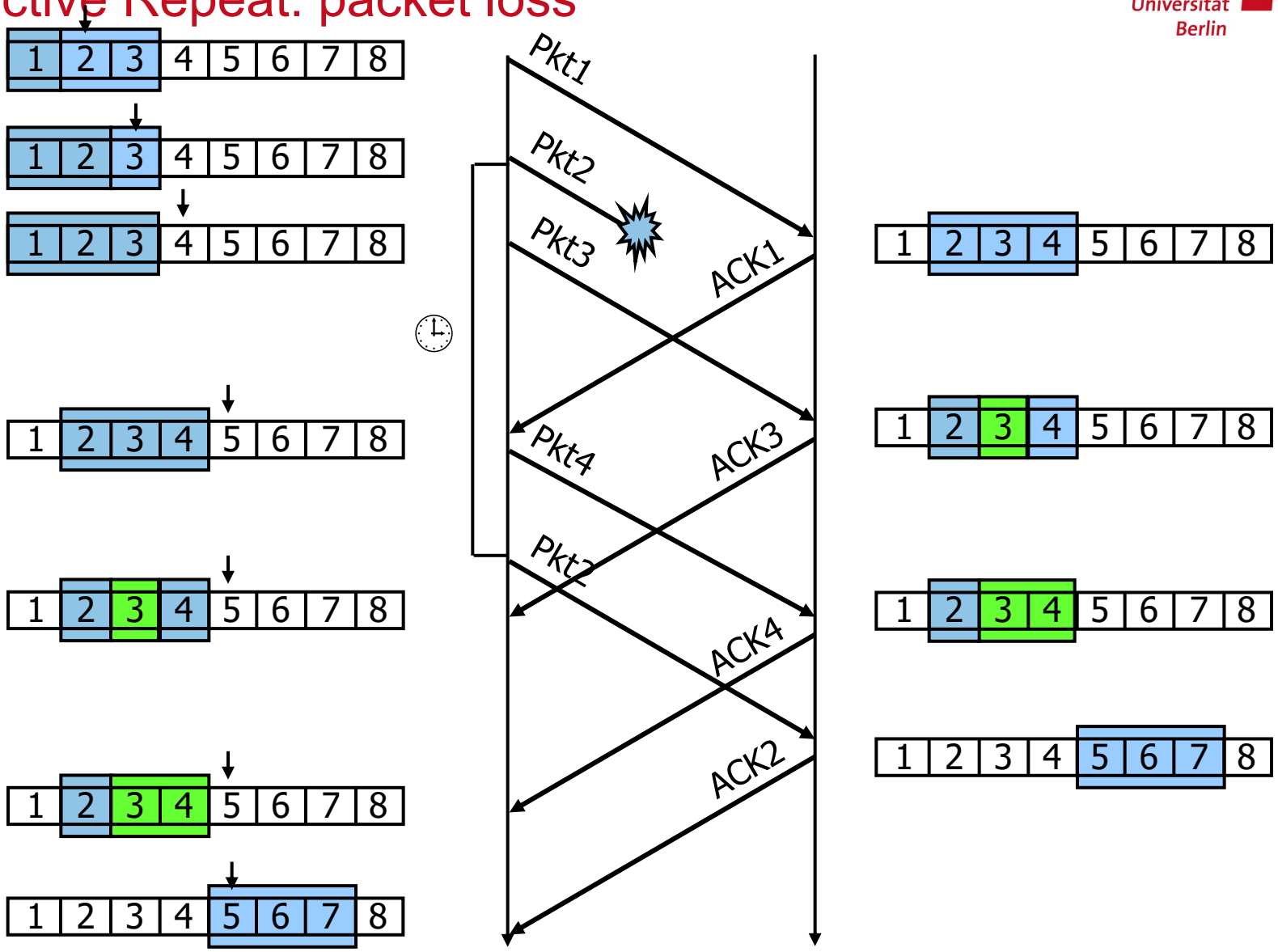
udt_rcv(pkt)
[biterror(pkt) ∨
 SQN(pkt) not in [base-W,base+W-1]/

udt_rcv(pkt)
[¬biterror(pkt) ∧ SQN(pkt) in [base-W,base-1]]/
ACK=ACK(SQN(pkt),CRC);
udt_send(ACK)

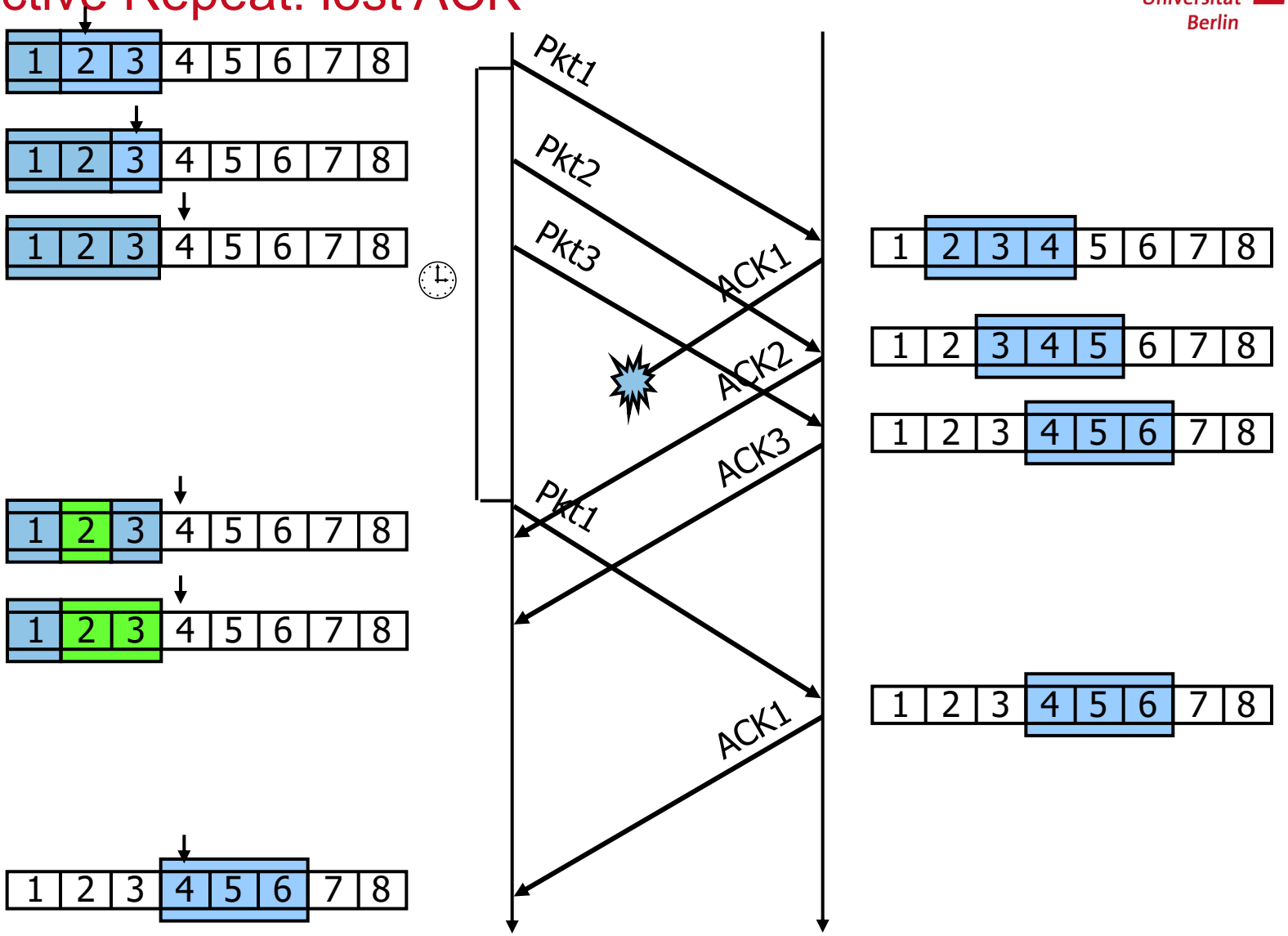
# Selective Repeat: normal execution



# Selective Repeat: packet loss



# Selective Repeat: lost ACK



# Sequence numbers for sliding window protocols

- Finite sequence numbers with m values
- Cyclic re-use of SQN
- Different packets with same SQN need to be distinguished
- Sufficient condition:
  - If receive window = 1: W < m
  - If send window = receive window = W > 1:
    W < (m+1)/2

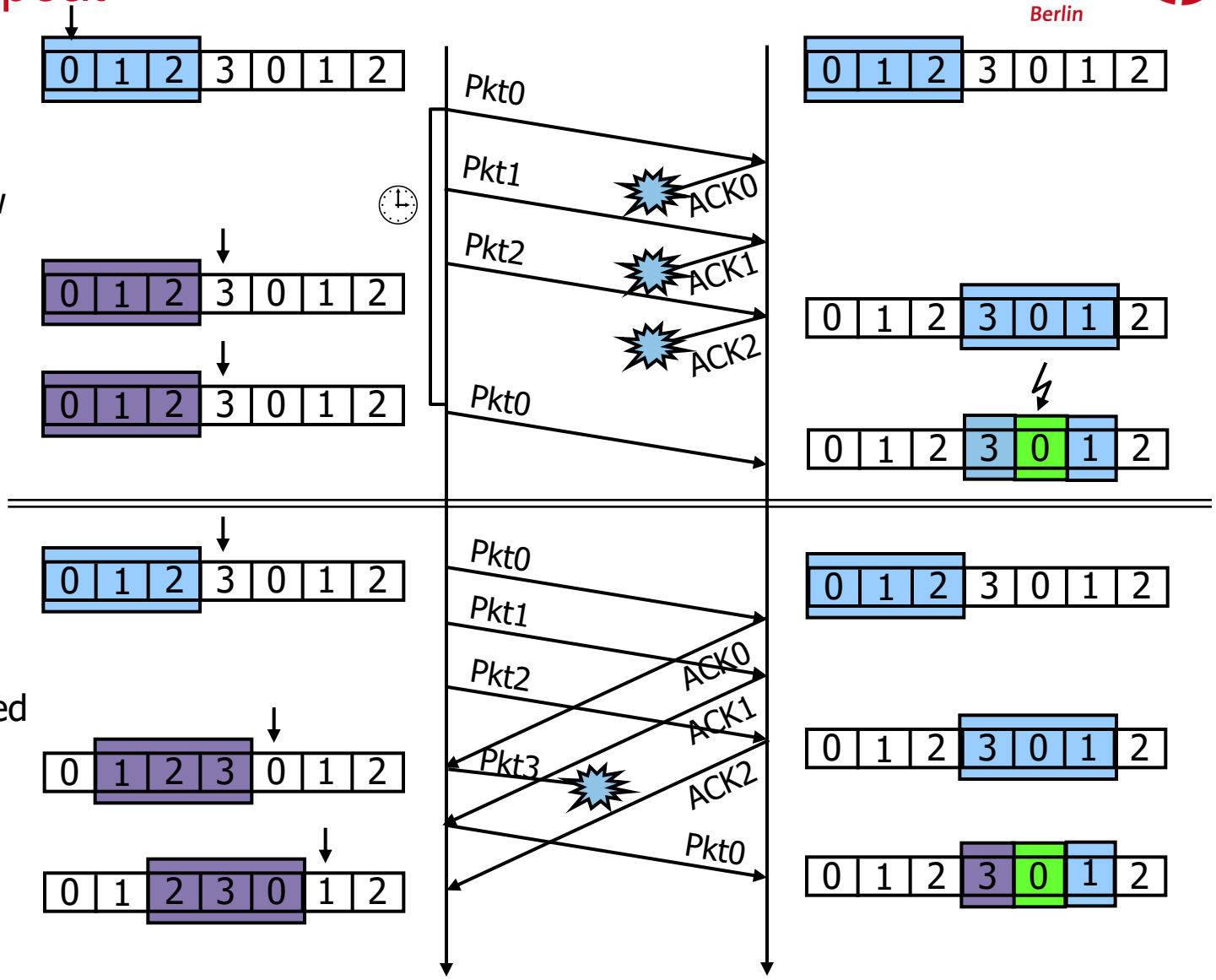
- Example for too small sequence number space
  - Sequence numbers m = 4, window size W = 3
  - W > (m+1)/2
  - Receiver cannot distinguish between old and new packets (see next slide)

# Selective Repeat

Two possible sequences for m=4 and W=3, which the receiver cannot distinguish and which lead to different results

Re-sent packet 0 is accepted as new packet

New packet 0 is correctly accepted as new



# Comparison Go-Back-N and Selective Repeat

- Advantages Go-Back-N
  - Cumulative ACKs compensate for lost and delayed ACKs, without the need to re-send packets
  - Sender only requires one timer
  - Receiver does not need buffer
  - Sender and receiver are easy to implement as no vacant spots in the window need to be maintained

- Advantages Selective Repeat
  - Less repeated packets because only really lost packets need to be retransmitted

# Performance analysis

# Performance analysis

- Possible questions
  - When does Stop-and-Wait block, what's the impact on throughput?
  - How large does the sliding window need to be to optimize throughput?
  - Which is more efficient: Go-Back-N or Selective Repeat?

- Following examples
  - W. Stallings: *Computer Networking with Internet Protocols and Technology*, Pearson Education, 2004 und W. Stallings: *High-Speed Networks, TCP/IP and ATM Design Principles*, Prentice Hall, 1998
  - Typical performance analysis in communication systems
  - Many assumptions to ease analysis
  - Mathematical terms are not that bad ☺
  - More detailed evaluation possible in computer simulation

# Bandwidth-delay-product

- A gentle reminder (see Chapter 1)

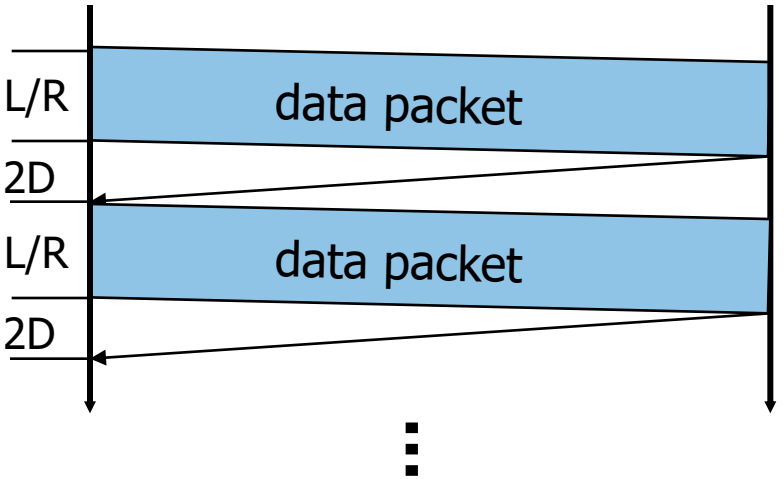
- **Storage capacity of a channel** (in packets, for packet size L)

  - $a = \frac{RD}{L} = \frac{d/v}{L/R} = \frac{\text{Propagation delay}}{\text{Packet transmission time}}$

    (Number of packets sent while the first bis travels from sender to receiver)

# Performance abalysis: Stop-and-Wait

- Stop-and-Wait without errors
  - (we ignore the time to send ACKs as well as all processing time – this assumption does not significantly change results and eases calculation)




  - Throughout in bit per second

$$\text{Throughout} = \frac{L}{\frac{L}{R} + 2D}$$

  - Normalized by bitrate (for better comparability)

$$\text{Normalized throughout} = S = \frac{L}{\frac{L}{R} + 2D}\frac{1}{R} = \frac{1}{1 + 2RD/L} = \frac{1}{1 + 2a}$$

  - $\boldsymbol{S = \frac{1}{1+2a}}$ $\Rightarrow$ **low throughput for large a (channel cannot be filled with packets)**

# Performance analysis: Stop-and-Wait

- Stop-and-Wait with errors
  - Retransmission after error (timeout)
  - Assumption (again): errors appear with probability p, errors are independent
  - Timeout = 2D
  - N is the mean number of transmissions of each packet

  - Throughout in bit per second

$$\text{Throughout} = \frac{L}{N\left(\frac{L}{R} + 2D\right)}$$

  - Normalized by bitrate (for better comparability)

$$\text{Normalized throughout} = S = \frac{L/R}{N\left(\frac{L}{R} + 2D\right)} = \frac{1}{N(1 + 2RD/L)} = \frac{1}{N(1 + 2a)}$$

# Performance analysis: Stop-and-Wait

- Calculation of N:
  - Probability that a packet needs to be sent i times is equal to the probability that a transmission failed i-1 times followed by a successful attempt
  - Pr[i transmission attempts] = $p^{i-1} \cdot (1-p)$
  - This is a **geometric distribution**, expected value:

    $N = E[\text{transmission attemps}] = \sum_{i=1}^{\infty} i \cdot \Pr[i \text{ transmission attemps}]$

    $N = \sum_{i=1}^{\infty} i \cdot p^{i-1} \cdot (1-p) = (1-p) \sum_{i=1}^{\infty} i \cdot p^{i-1} = (1-p) \frac{1}{(1-p)^2} = \frac{1}{1-p}$

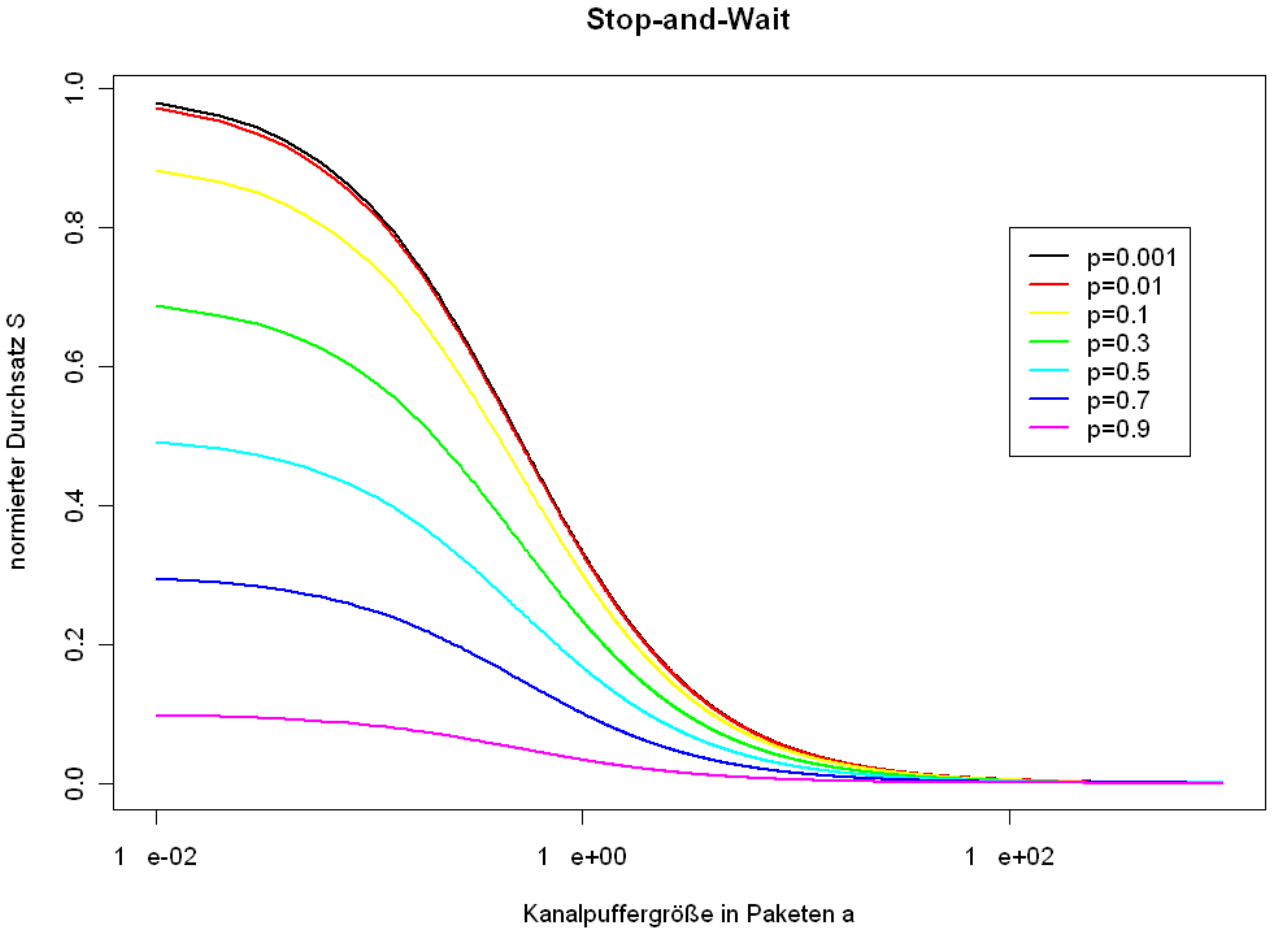
    using $\sum_{i=1}^{\infty} iX^{i-1} = \frac{1}{(1-X)^2}$ for (-1 < X < 1)

  - Normalized throughout:

    $\boldsymbol{S = \frac{1-p}{1+2a}}$ $\Rightarrow$ **low throughput for large a and large p**

# Performance analysis: Stop-and-Wait

- Normalized throughput of Stop-and-Wait as a function of a:




For large a and for large p, the normalized throughput gets smaller

# Performance analysis: sliding window protocols

- Sliding window protocols without error
  - Window size W and packet length L
  - Case 1: window is large enough to continuously send before first ACK:

$$W \geq \frac{L/R + 2D}{L/R} = 1 + 2a$$

$$S = \frac{W \cdot L}{W \cdot L/R} \cdot \frac{1}{R} = 1$$

  - Case 2: window is too small:

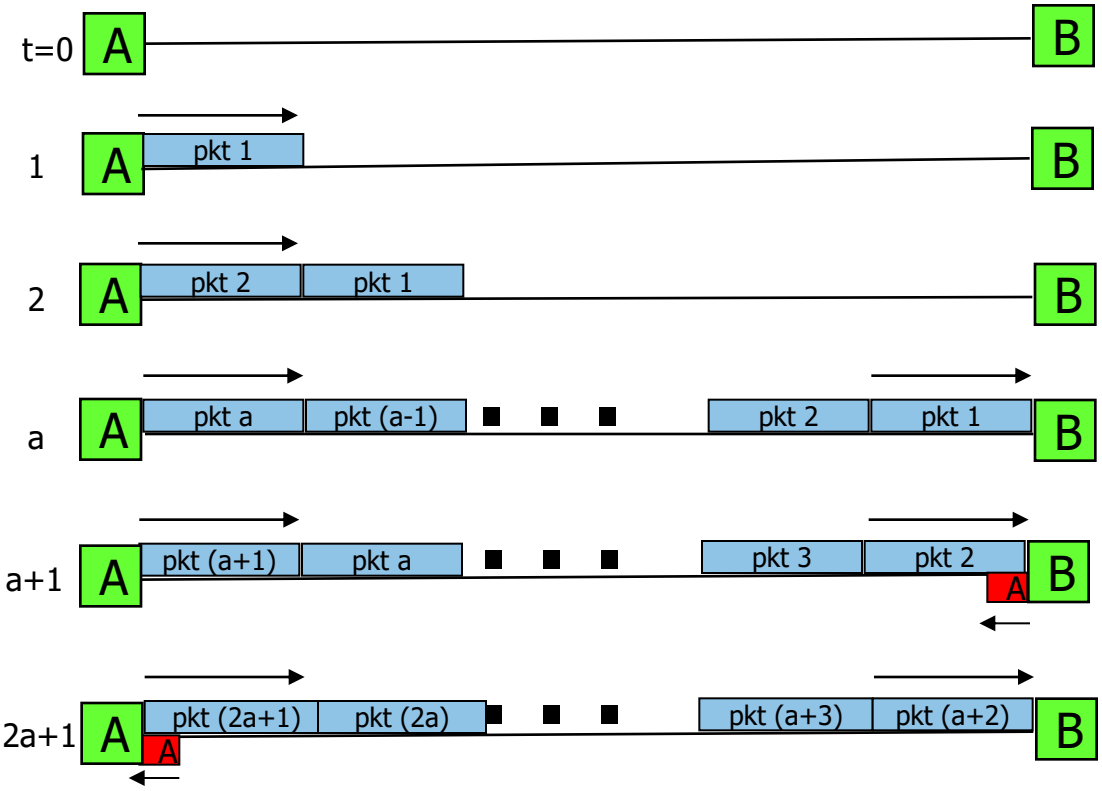
$$W < 1 + 2a$$

$$S = \frac{W \cdot L}{L/R + 2D} \cdot \frac{1}{R} = \frac{W}{1 + 2a}$$

$$S = \begin{cases} 1 & W \geq 1 + 2a \\ \dfrac{W}{1 + 2a} & W < 1 + 2a \end{cases}$$

# Performance analysis: sliding window protocols

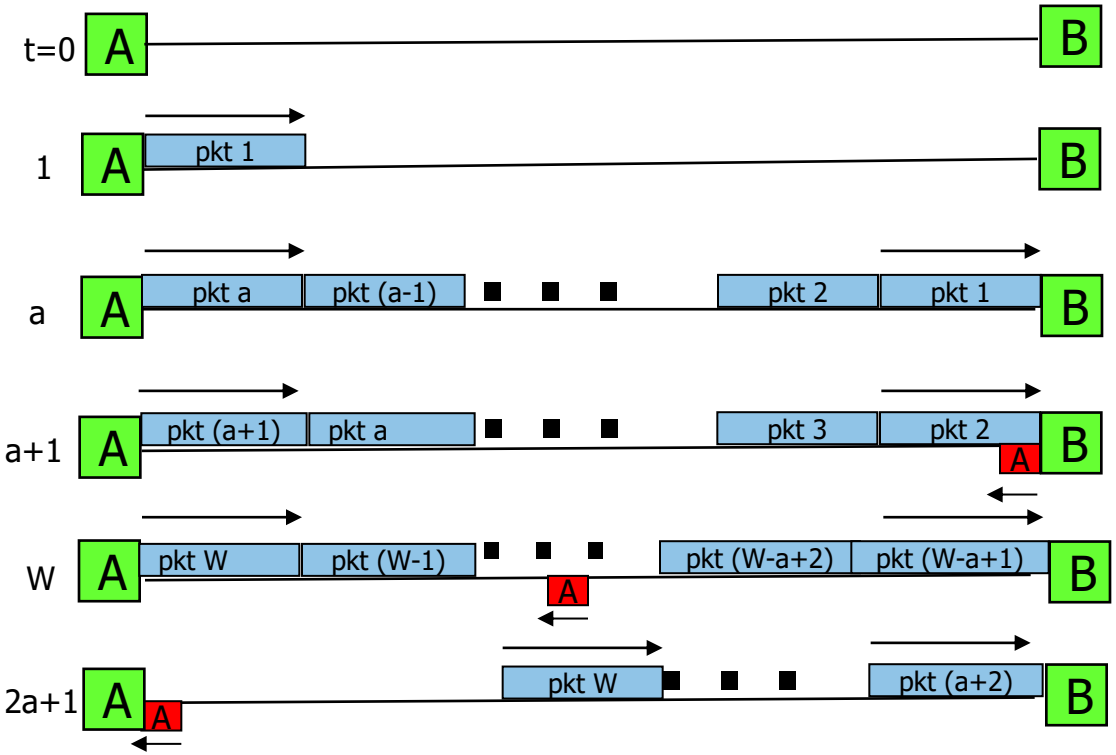
- Schematic procedure




$$W > 2a + 1$$

# Performance analysis: sliding window protocols

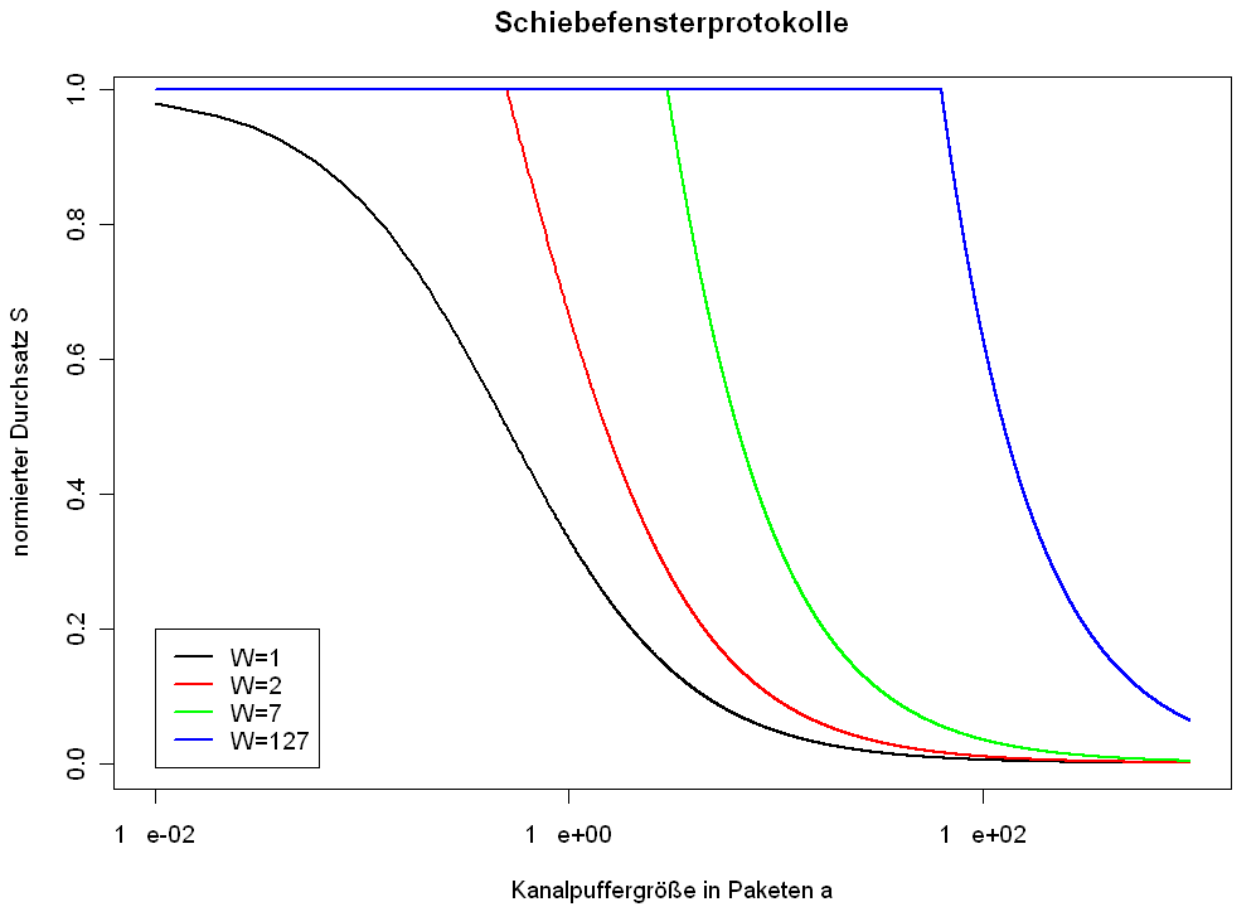
- Schematic procedure




$$W < 2a + 1$$

# Performance analysis: sliding window protocols

- Normalized throughput of sliding window protocols as a function of a:




→ Window size W Is sufficient to fill the pipe, if a = (W-1)/2

# Performance analysis: sliding window protocols

- Selective Repeat with errors
  - Assumption: errors with probability p are independent
  - N = E[transmission attempts] = 1/(1-p)
  - Throughput in error-free case needs to be divided by N:

$$S = \begin{cases} \frac{1}{N} = \frac{1}{1/(1-p)} = 1-p & W \geq 1+2a \\ \frac{W}{N \cdot (1+2a)} = \frac{W}{1/(1-p) \cdot (1+2a)} = \frac{W(1-p)}{1+2a} & W < 1+2a \end{cases}$$

$$S = \begin{cases} 1-p & W \geq 1+2a \\ \frac{W(1-p)}{1+2a} & W < 1+2a \end{cases}$$

# Performance analysis: sliding window protocols

- Go-back-N with errors
  - Every error requires retransmission of K packets
  - Assumption: in case of error, the pipe is filled and all packets in the window needs to be retransmitted:

$$K = \begin{cases} 1+2a & W \geq 1+2a \\ W & W < 1+2a \end{cases}$$

  - If the erroneous packet needs to be retransmitted i times, in total 1+(i-1)K = (1-K)+Ki packets need to be sent

$$N = \sum_{i=1}^{\infty} ((1-K)+Ki) \cdot p^{i-1} \cdot (1-p) = (1-K)(1-p)\sum_{i=1}^{\infty} p^{i-1} + K(1-p)\sum_{i=1}^{\infty} i \cdot p^{i-1}$$

$$= (1-K)(1-p)\frac{1}{(1-p)} + K(1-p)\frac{1}{(1-p)^2}$$

$$= 1-K+\frac{K}{1-p} = \frac{1-p+Kp}{1-p}$$

$$\text{using } \sum_{i=1}^{\infty} X^{i-1} = \frac{1}{1-X} \text{ for } (-1 < X < 1)$$

# Performance analysis: sliding window protocols

- Given K, we get:

$$N = \begin{cases} \frac{1-p+Kp}{1-p} = \frac{1-p+(1+2a)p}{1-p} = \frac{1+2ap}{1-p} & W \geq 1+2a \\ \frac{1-p+Kp}{1-p} = \frac{1-p+Wp}{1-p} & W < 1+2a \end{cases}$$

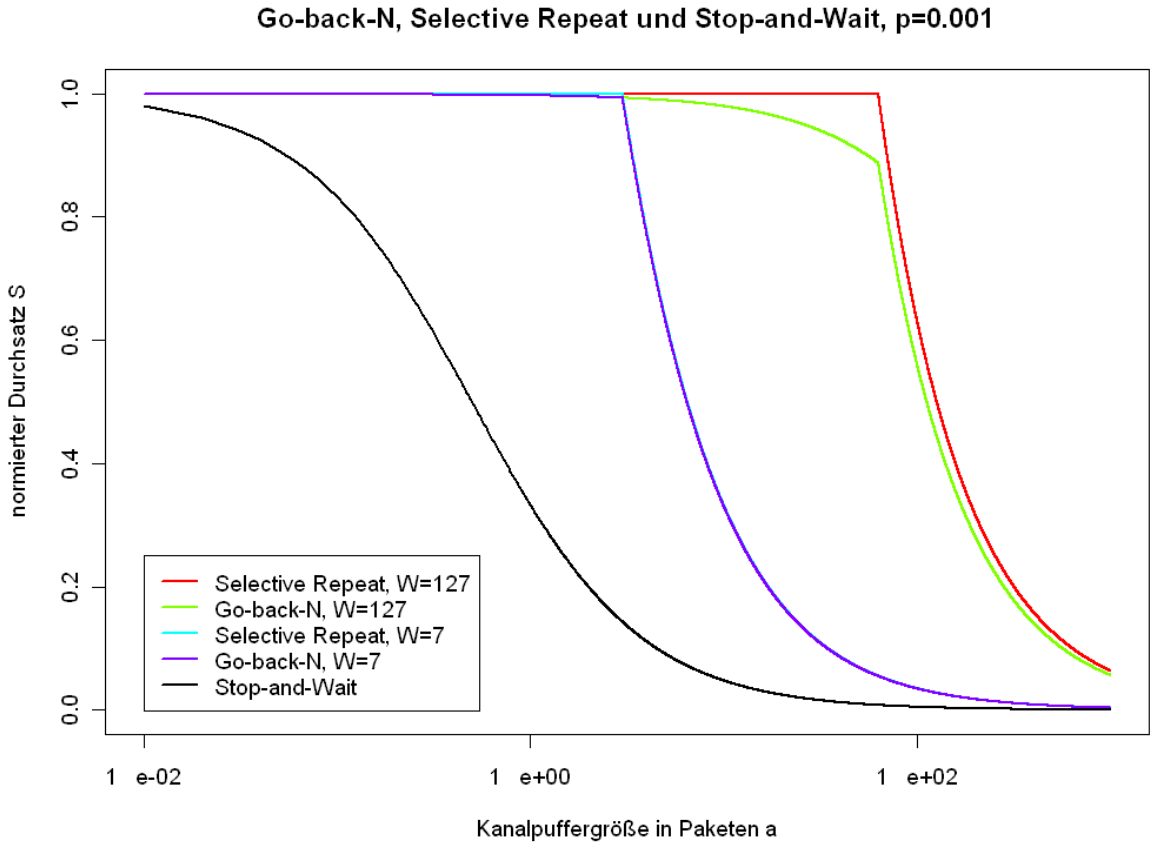
- Normalized throughout (dividing throughput w/o errors by N):

$$S = \begin{cases} \frac{1}{N} = \frac{1-p}{1+2ap} & W \geq 1+2a \\ \frac{W}{N \cdot (1+2a)} = \frac{W(1-p)}{(1-p+Wp) \cdot (1+2a)} & W < 1+2a \end{cases}$$

$$S = \begin{cases} \frac{1-p}{1+2ap} & W \geq 1+2a \\ \frac{W(1-p)}{(1-p+Wp) \cdot (1+2a)} & W < 1+2a \end{cases}$$

# Performance analysis: sliding window protocols

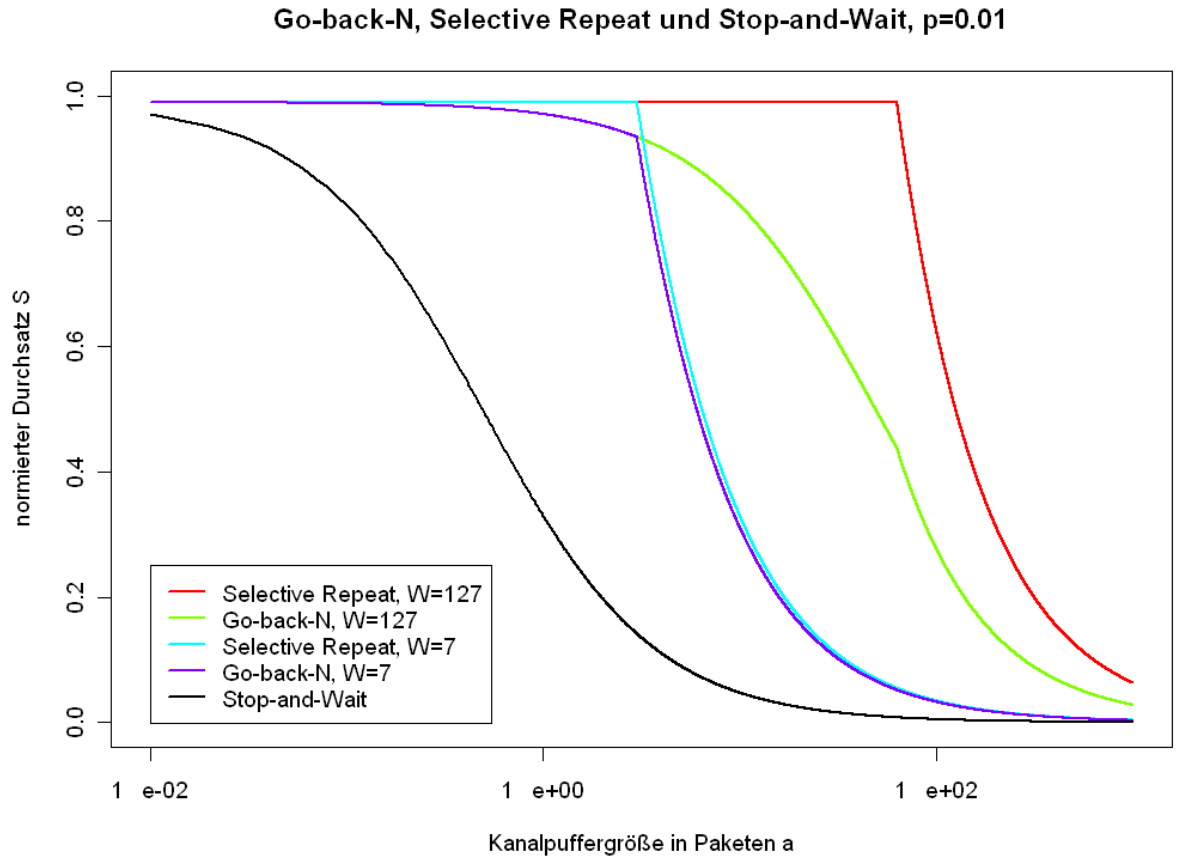
- Normalized throughput of G-Back-N and Selective Repeat as a function of a, p = $10^{-3}$:




→ only for large window sizes, we notice an advantage of Selective Repeat

# Performance analysis: sliding window protocols

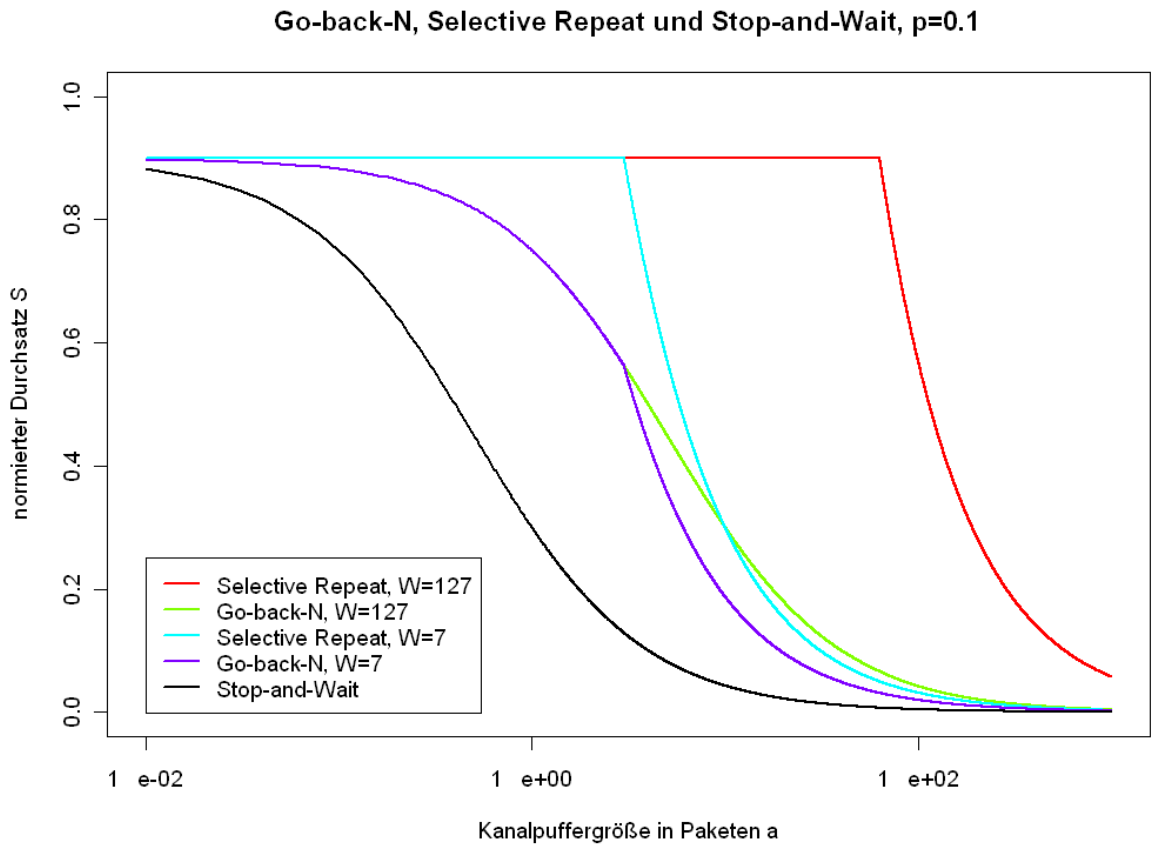
- Normalized throughput of G-Back-N and Selective Repeat as a function of a, p = $10^{-2}$:




→ due to the higher loss rate, we notice the advantage of Selective Repeat also for smaller window sizes

# Performance analysis: sliding window protocols

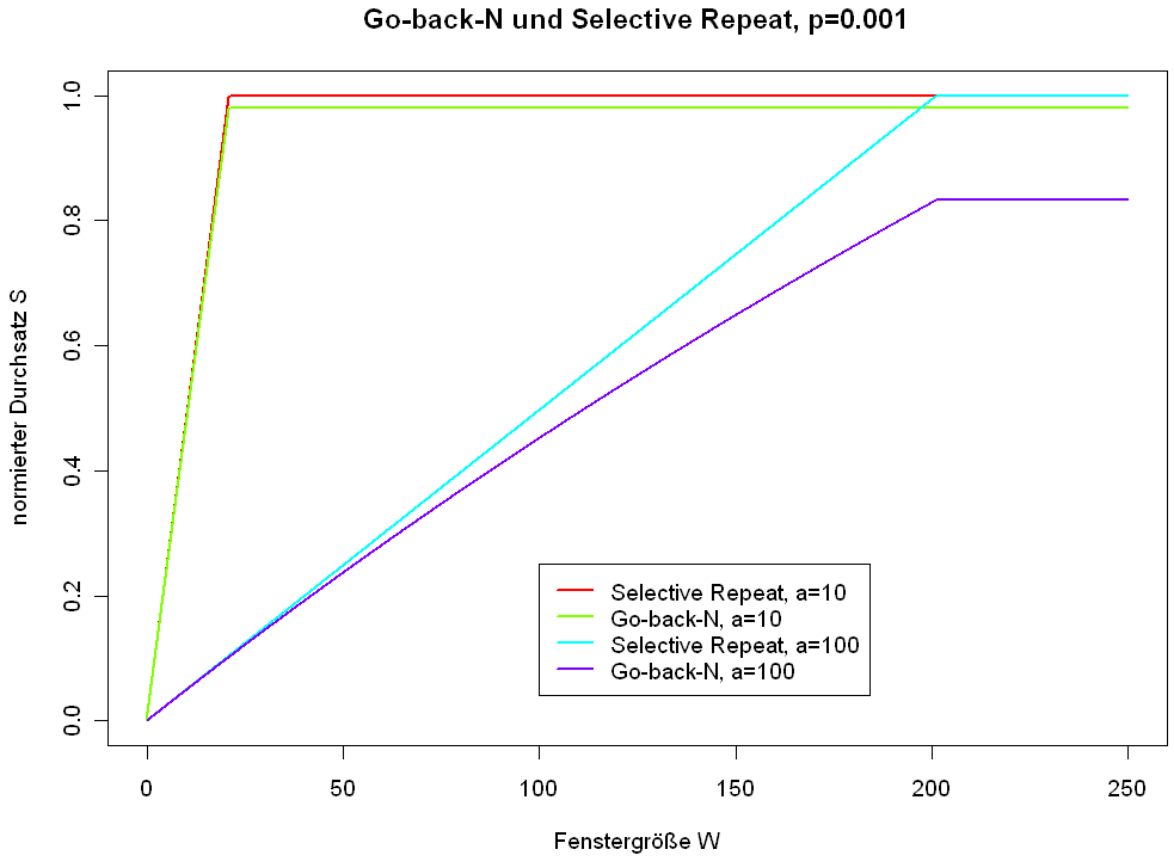
- Normalized throughput of G-Back-N and Selective Repeat as a function of a, p = $10^{-1}$:




→ for high loss rates, the advantage of Selective Repeat is very clearly visible

# Performance analysis: sliding window protocols

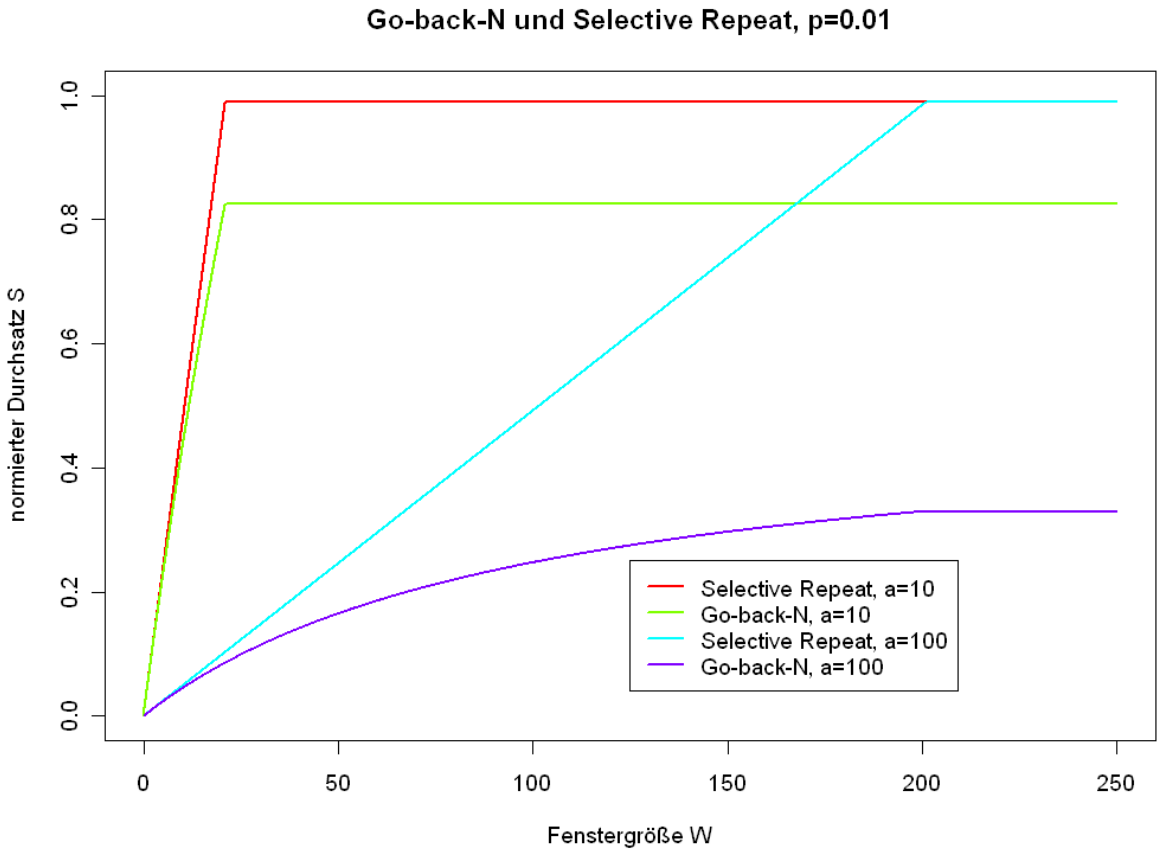
- Normalized throughput of G-Back-N and Selective Repeat as a function of W, p = $10^{-3}$:




→ for large a and for large window sizes, there is a significant difference between Go-Back-N and Selective Repeat

# Performance analysis: sliding window protocols

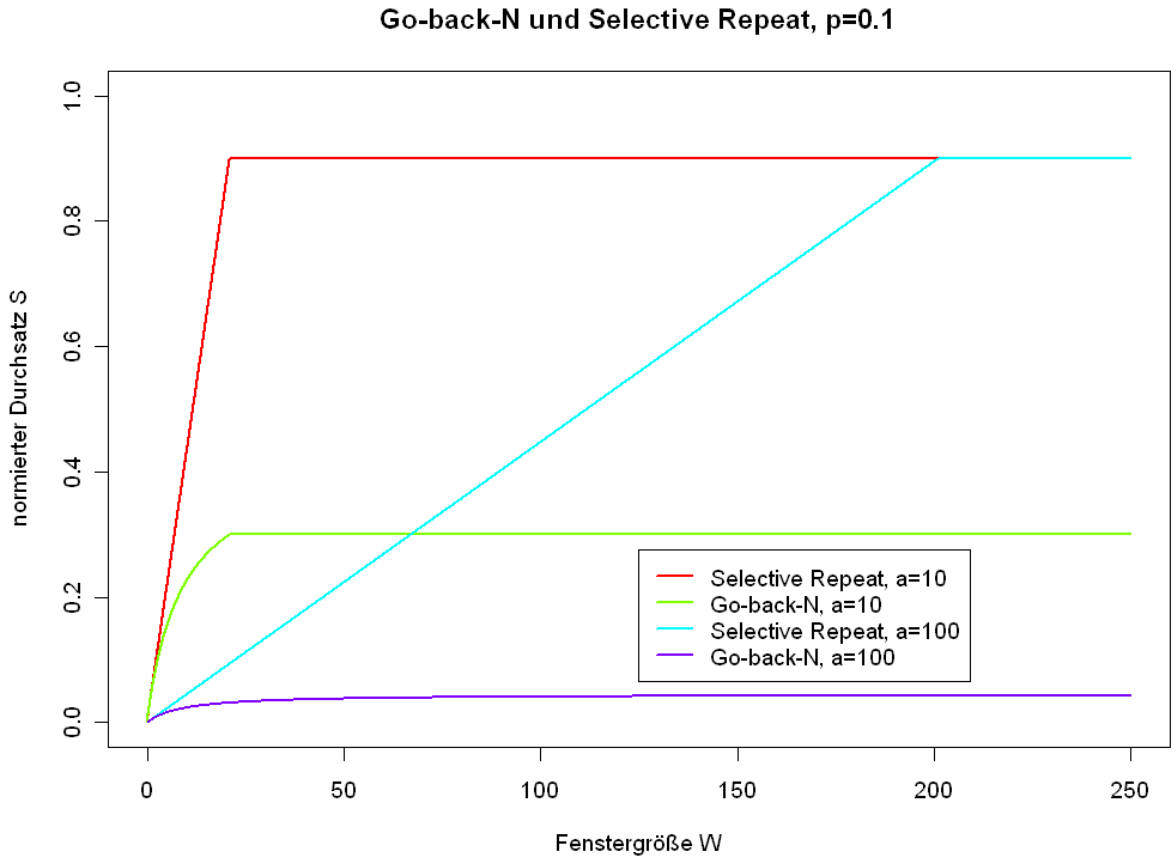
- Normalized throughput of G-Back-N and Selective Repeat as a function of W, p = $10^{-2}$:




→ the difference grows for higher loss rates

# Performance analysis: sliding window protocols

- Normalized throughput of G-Back-N and Selective Repeat as a function of W, p = $10^{-1}$:




→ and even more