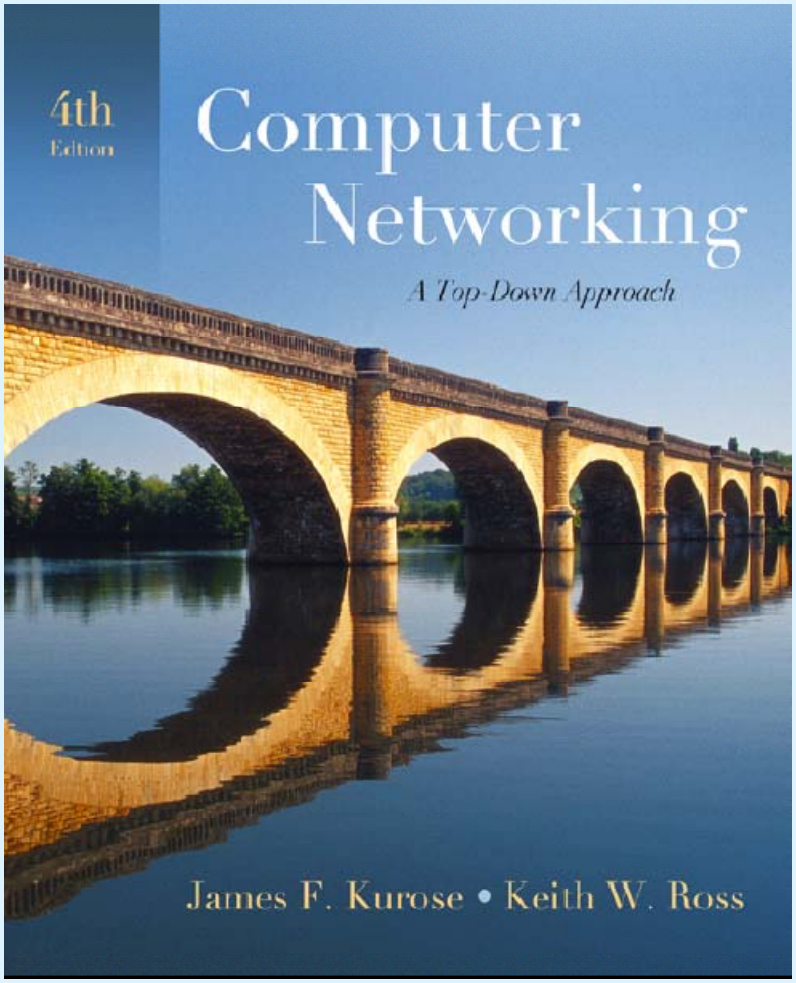
Transport Layer

Felövervakning

Övningar på Internet-checksumman

- UDP
  - Förbindelselöst
  - "Best effort"
- TCP
  - Förbindelseorientat
  - Tillförlitligt
  - Flödesreglering
  - Trafikstockningsreglering

Bildspelet omfattar till stor del bilder som hör till följande bok:



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

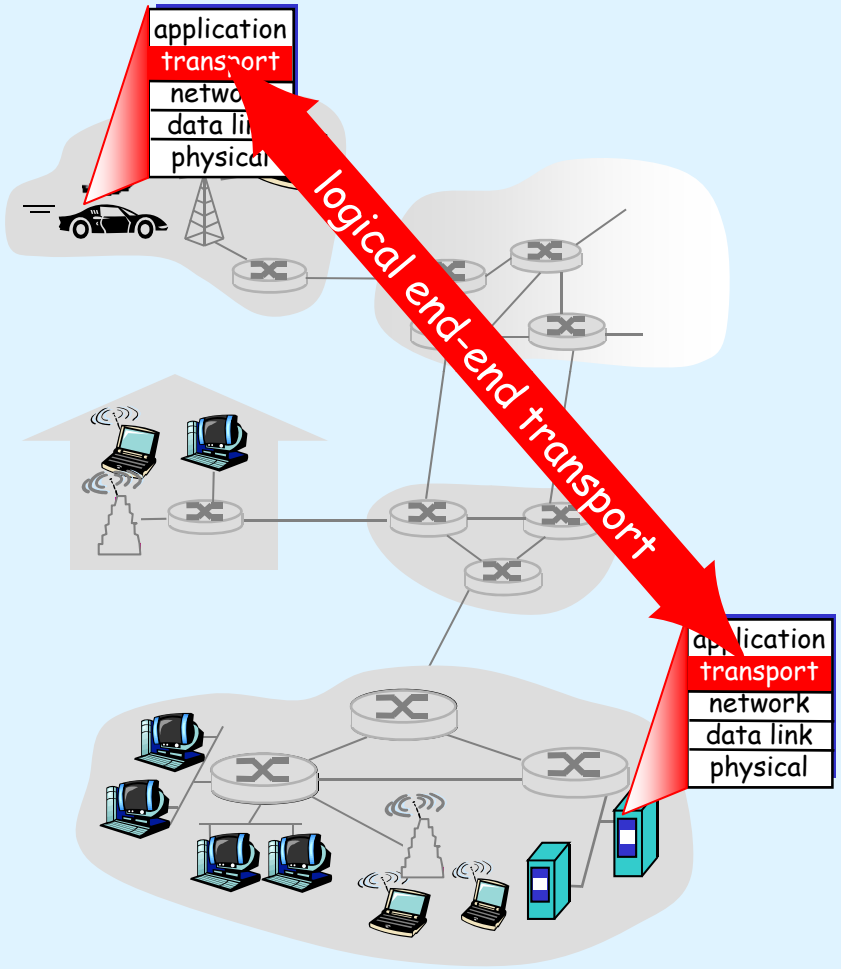
- ❑ If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- ❑ If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach, 4th edition.
Jim Kurose, Keith Ross, Addison-Wesley, July 2007.

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - receiver side: reassembles segments into messages, passes to app layer



Transport vs. network layer

- *Transport layer:*

  logical communication between processes

  - relies on, enhances, network layer services

- *Network layer:*

  logical communication between hosts

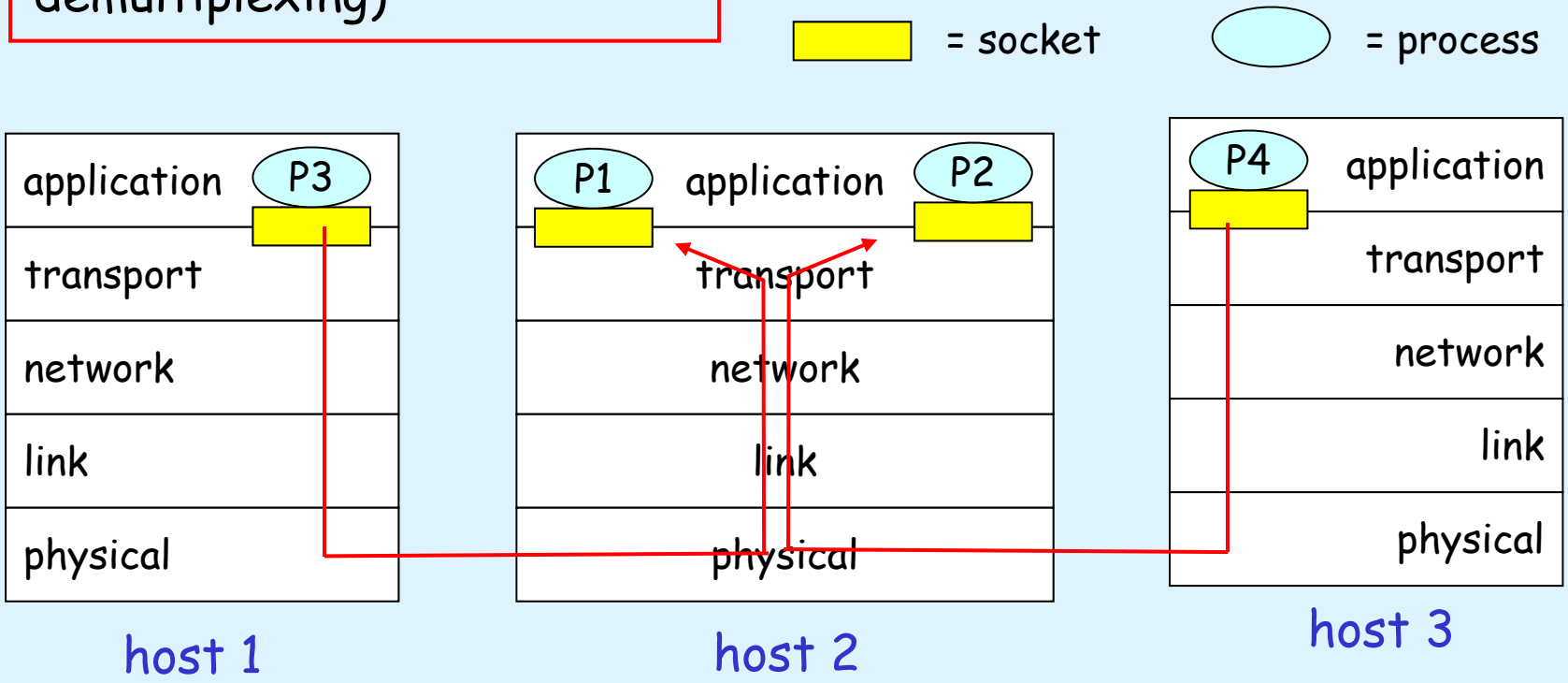
Multiplexing/demultiplexing

Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

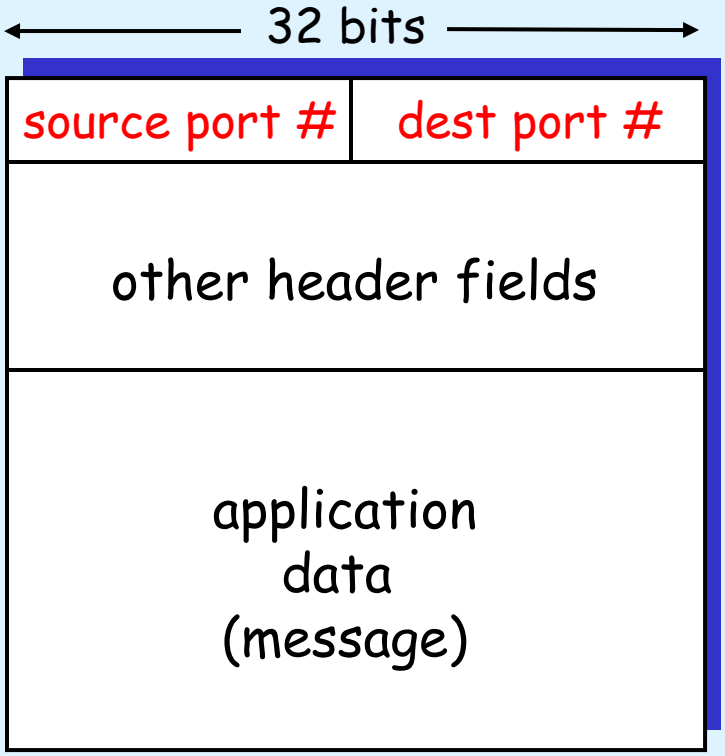
Demultiplexing at rcv host:

delivering received segments to correct socket



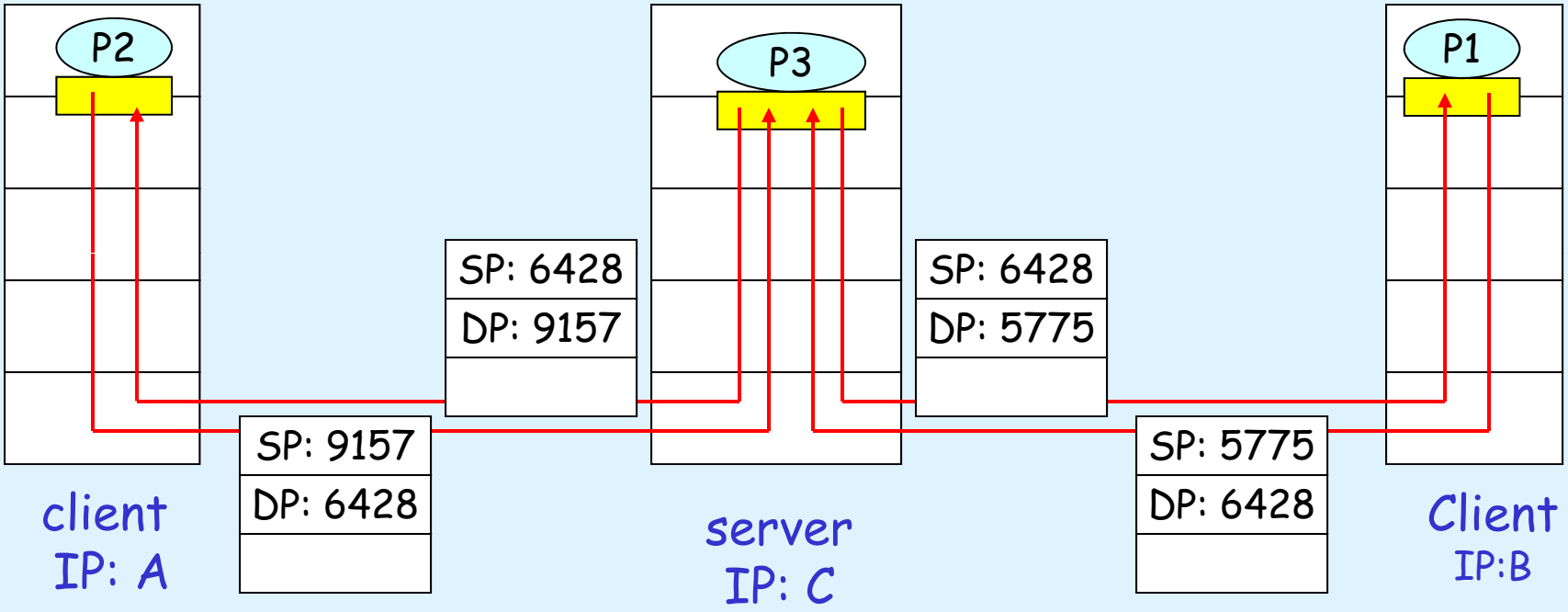
How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demux (cont)

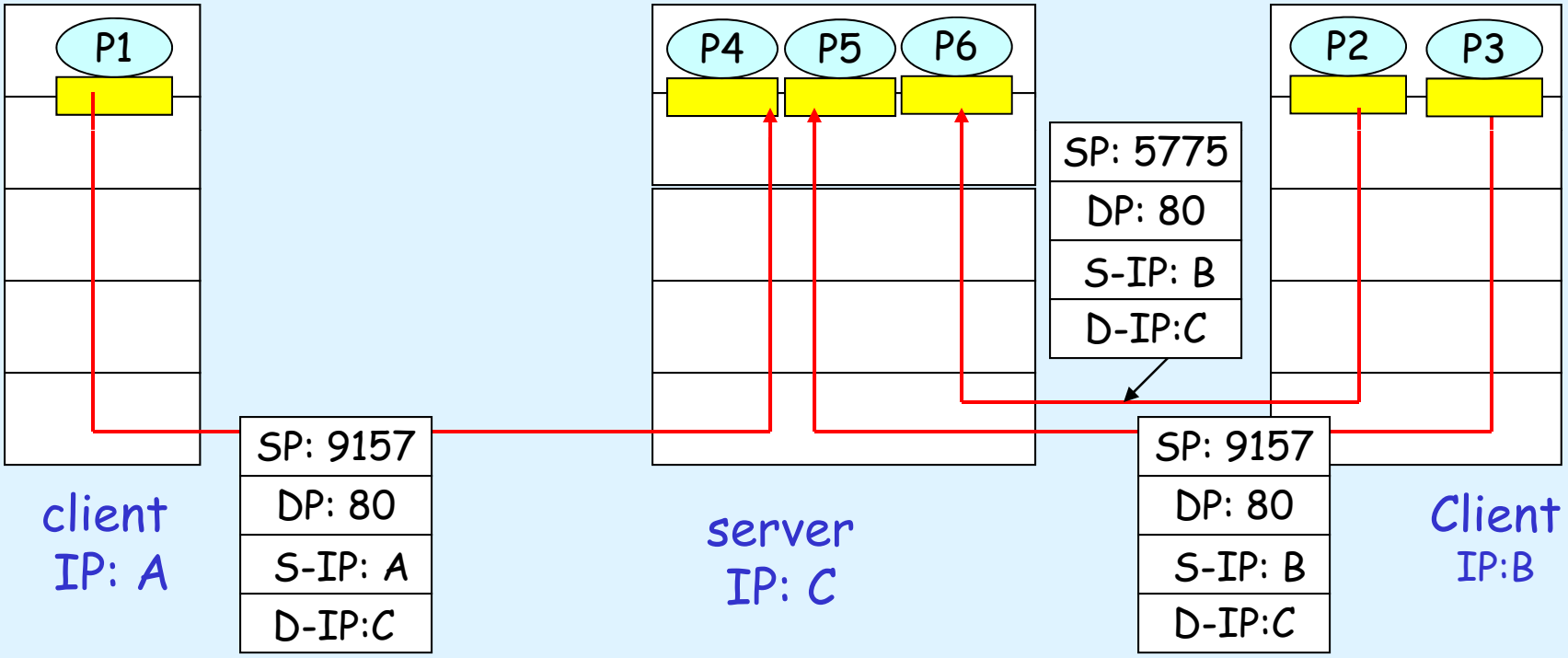


SP provides "return address"

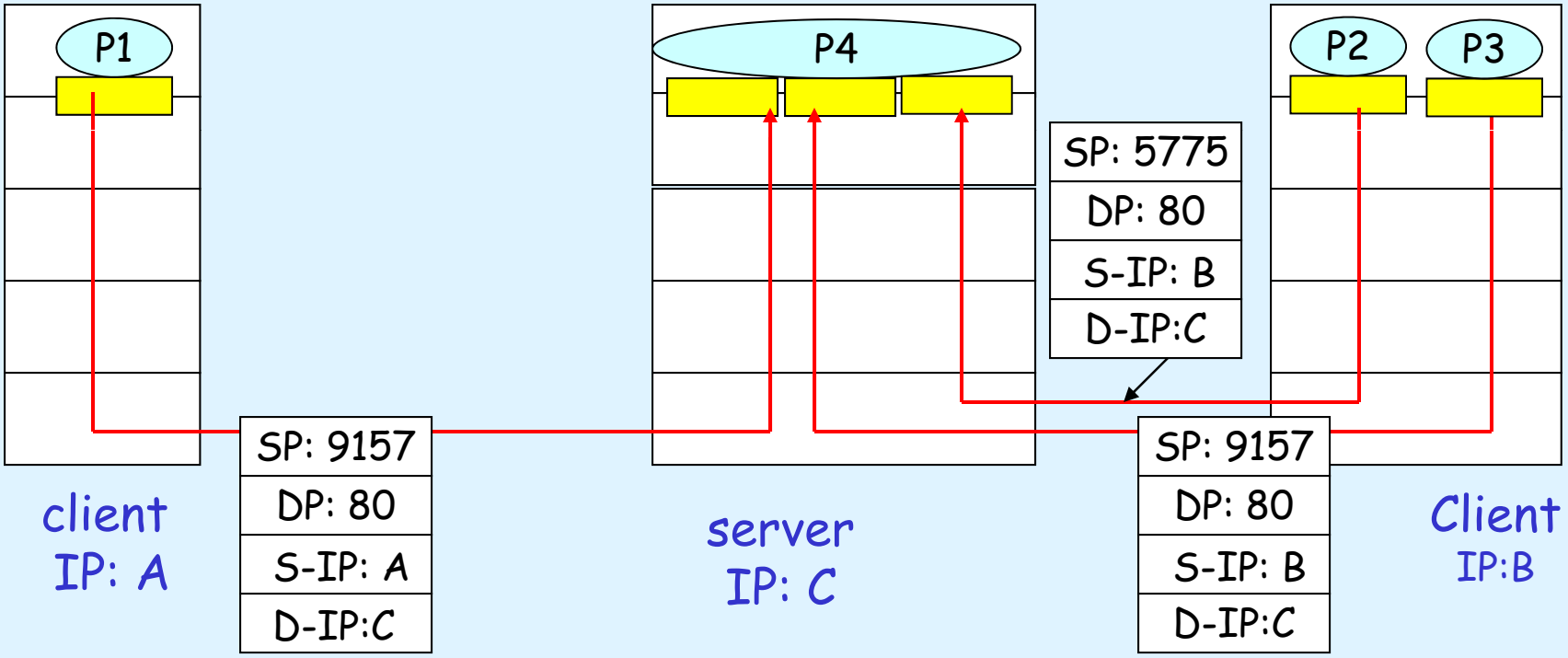
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux (cont)



Connection-oriented demux: Threaded Web Server



Felövervakning (1)

- Sekvensnummer
- Checksumma
- Omsändningsmetod

Felövervakning (2)

- Manuell

  Exempelvis Echo checking.

- Automatisk
  (Automatic Repeat Request, ARQ)

  Upptäcka bitfel (feedback error control)

  Korrigera bitfel (forward error control)

Kvittenser för ARQ

- Positiva kvittenser

  (Acknowledgement, ACK)

- Negativa kvittenser

  (Negative Acknowledgement, NAK)

Idle repeat request

- Implicit retransmission

  Endast ACK.

- Explicit request

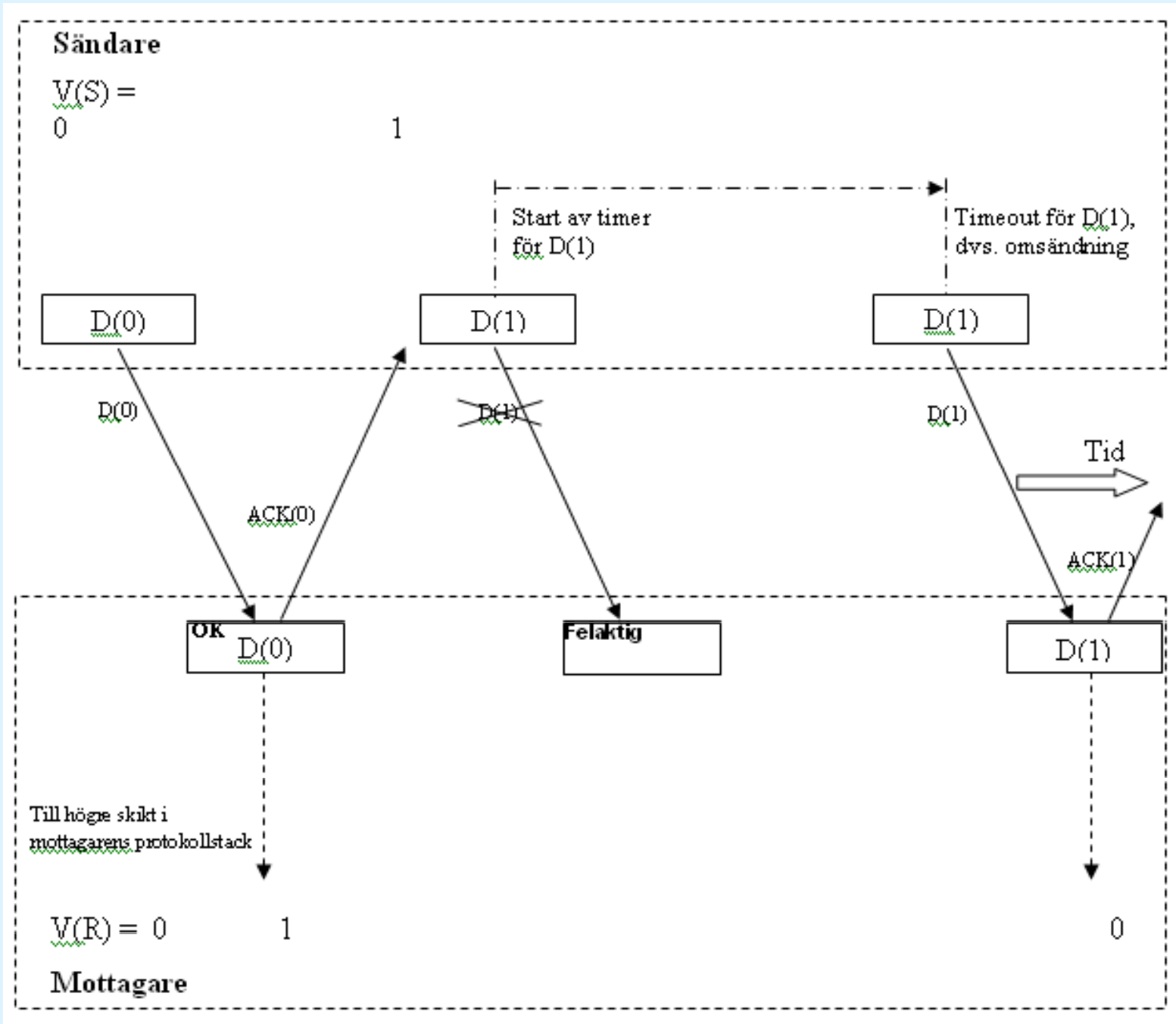
  ACK och NAK.

Implicit retransmission

- Sänd – vänta på ACK eller timeout

  Om ACK, sänd nästa – ...

  Om timeout, repetera föregående – ...


Sändare
V(S) =
0
1
Start av timer
för D(1)
Timeout för D(1),
dvs. omsändning
D(0)
D(1)
D(1)
D(0)
D(1)
D(1)
Tid
ACK(0)
ACK(1)
OK
D(0)
Felaktig
D(1)
Till högre skikt i
mottagarens protokollstack
V(R) = 0
1
0
Mottagare

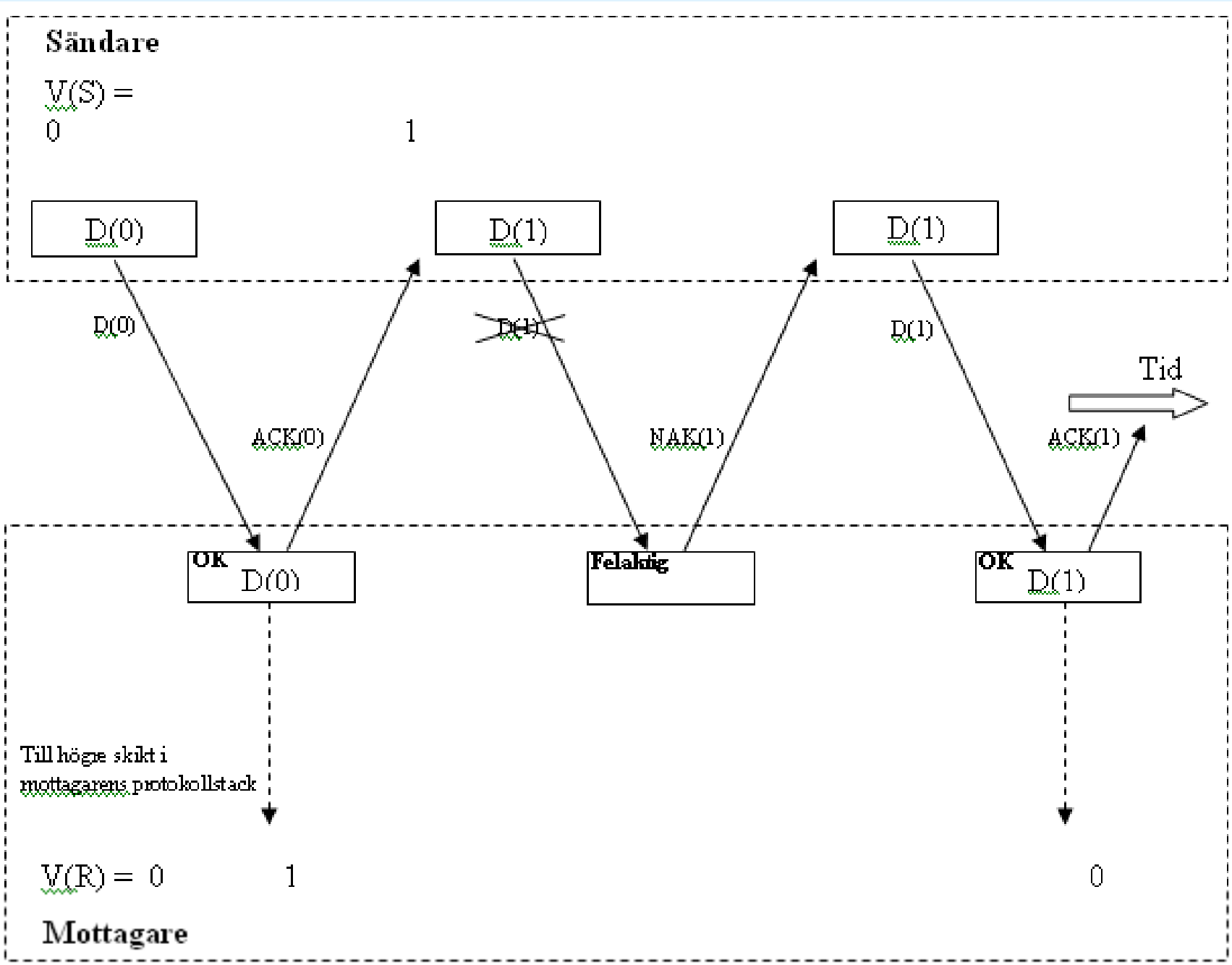
Explicit request

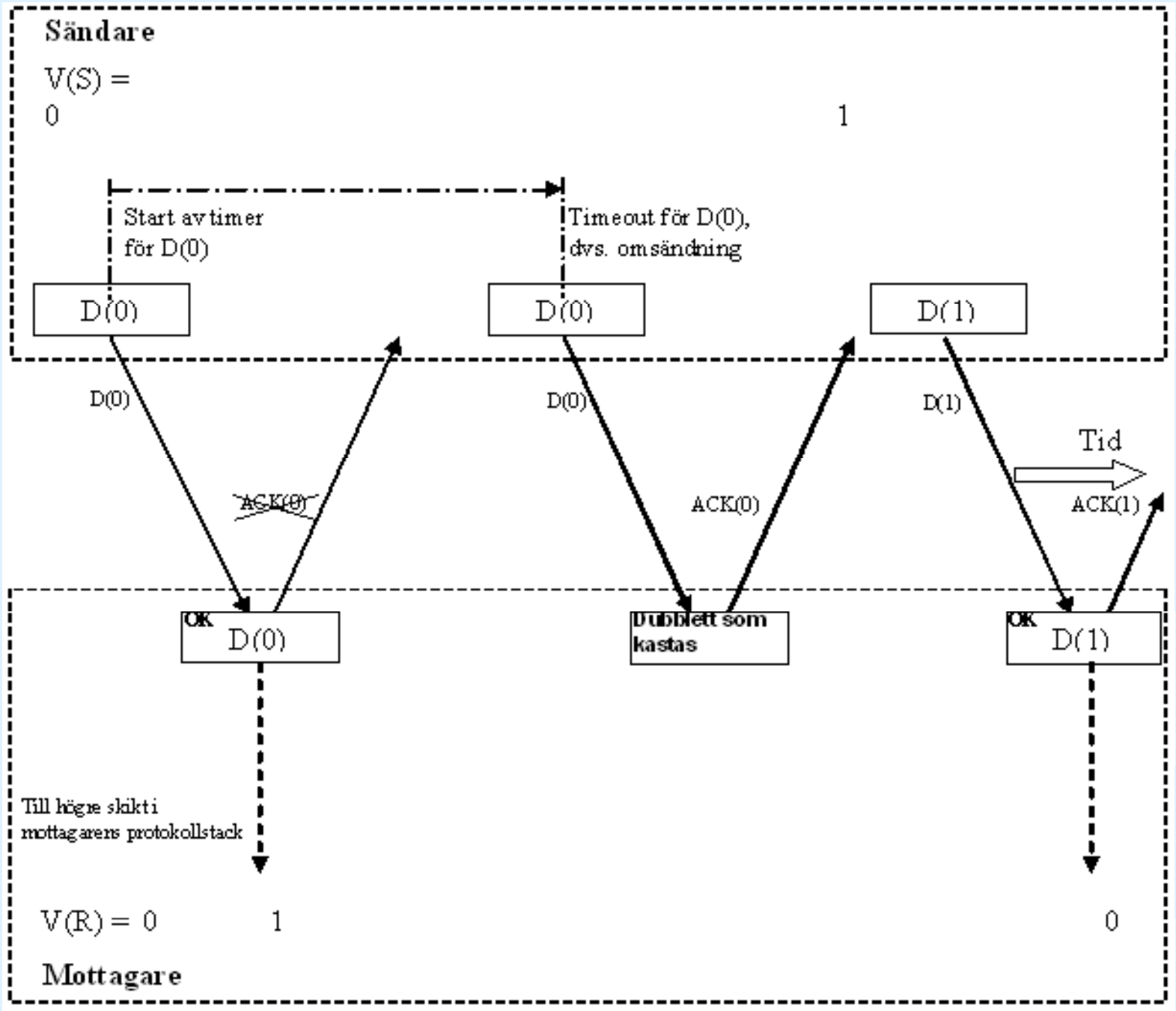
- Sänd – vänta på ACK, NAK eller timeout

  Om ACK, sänd nästa – ...

  Om NAK eller timeout, repetera föregående – ...

- NAK används av mottagaren för att snabbt få ett meddelande (ram, paket, segment etc.) repeterat.


Sändare
V(S) =
0
1
D(0)
D(1)
D(1)
D(0)
D(1)
D(1)
Tid
ACK(0)
NAK(1)
ACK(1)
OK
D(0)
Felaktig
OK
D(1)
Till högre skikt i
mottagarens protokollstack
V(R) = 0
1
0
Mottagare


Sändare
V(S) =
0
1
Start av timer
för D(0)
Timeout för D(0),
dvs. omsändning
D(0)
D(0)
D(1)
D(0)
D(0)
D(1)
Tid
ACK(0)
ACK(0)
ACK(1)
OK
D(0)
Dubblett som
kastas
OK
D(1)
Till högre skikt i
mottagarens protokollstack
V(R) = 0
1
0
Mottagare

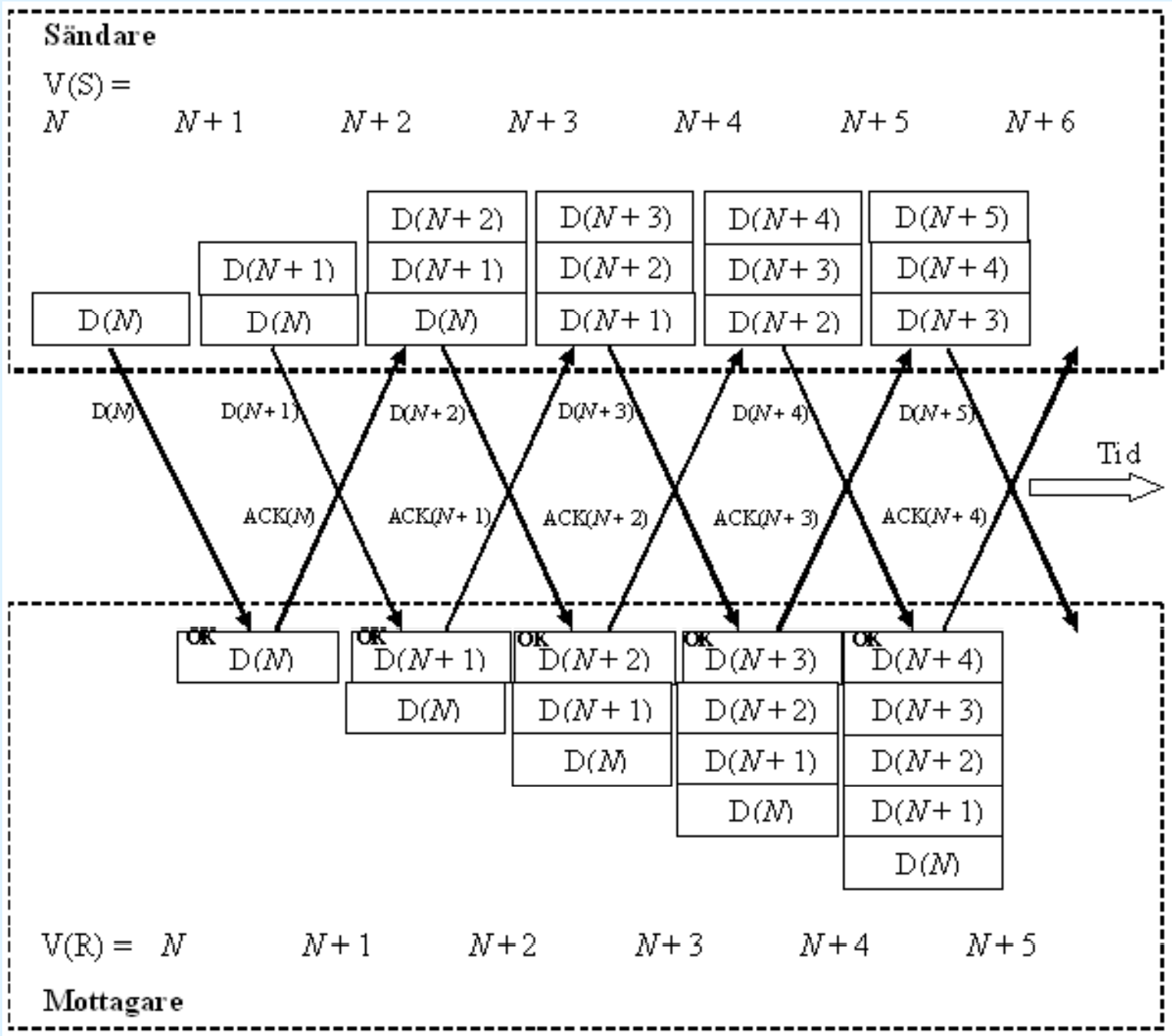
Continous repeat request (1)

- Omsändningsmetoderna med stora sändnings- och mottagningsbuffertar

- Skicka meddelanden (ramar, paket, segment etc.) som i strömmar utan att behöva invänta varje ACK, NAK eller timeout.

Continous repeat request (2)

- Selective repeat with implicit retransmission
- Selective repeat with explicit request
- Go-back-N


Sändare
V(S) =
N
N+1
N+2
N+3
N+4
N+5
N+6
D(N)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
D(N+5)
Tid
ACK(N)
ACK(N+1)
ACK(N+2)
ACK(N+3)
ACK(N+4)
OK
V(R) =
Mottagare

Selective repeat with implicit retransmission (1)

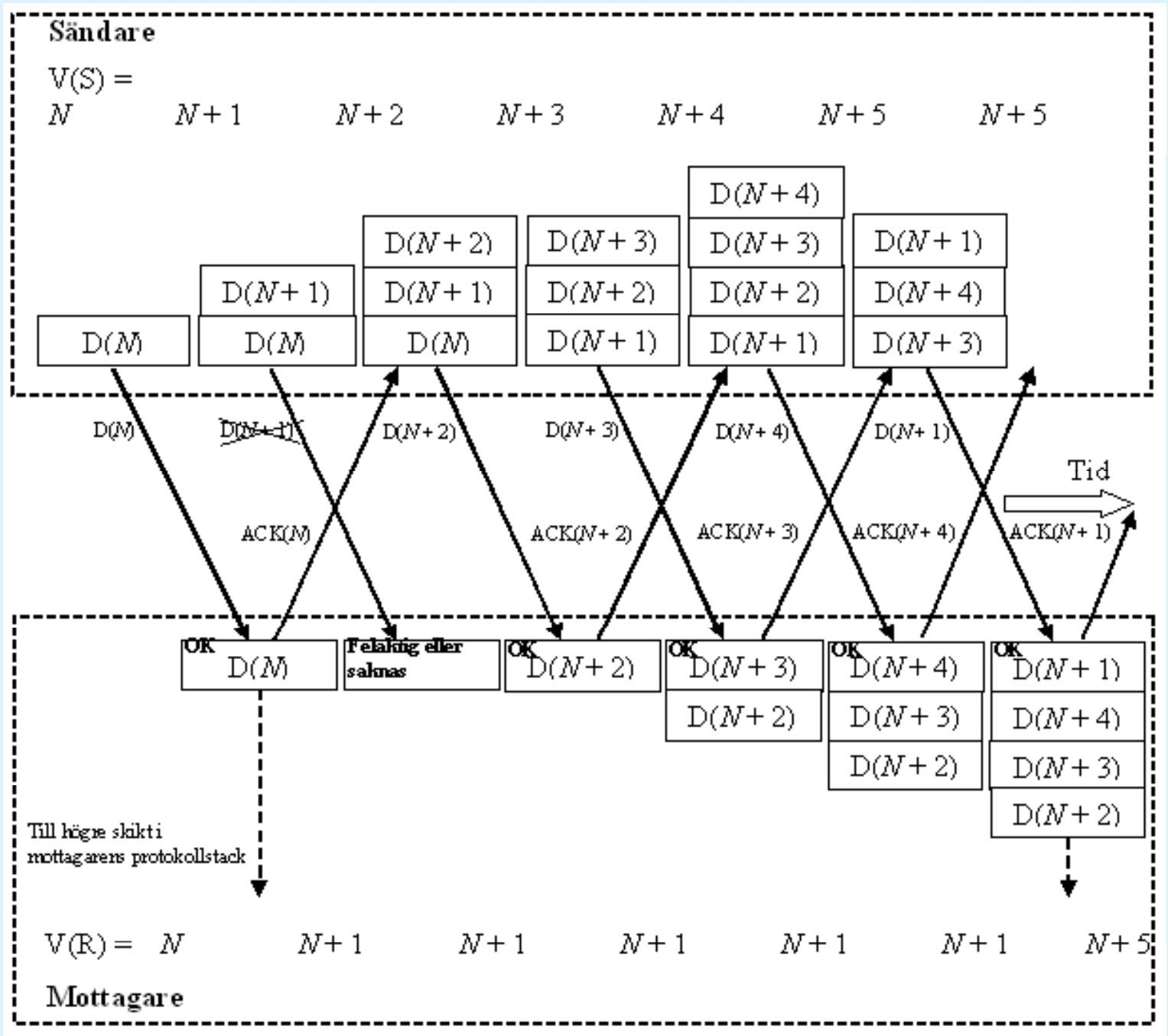
- Enbart felaktiga meddelanden (ramar, paket, segment etc.) sänds om vid begäran.
- Implicit retransmission betyder att NAK inte används.

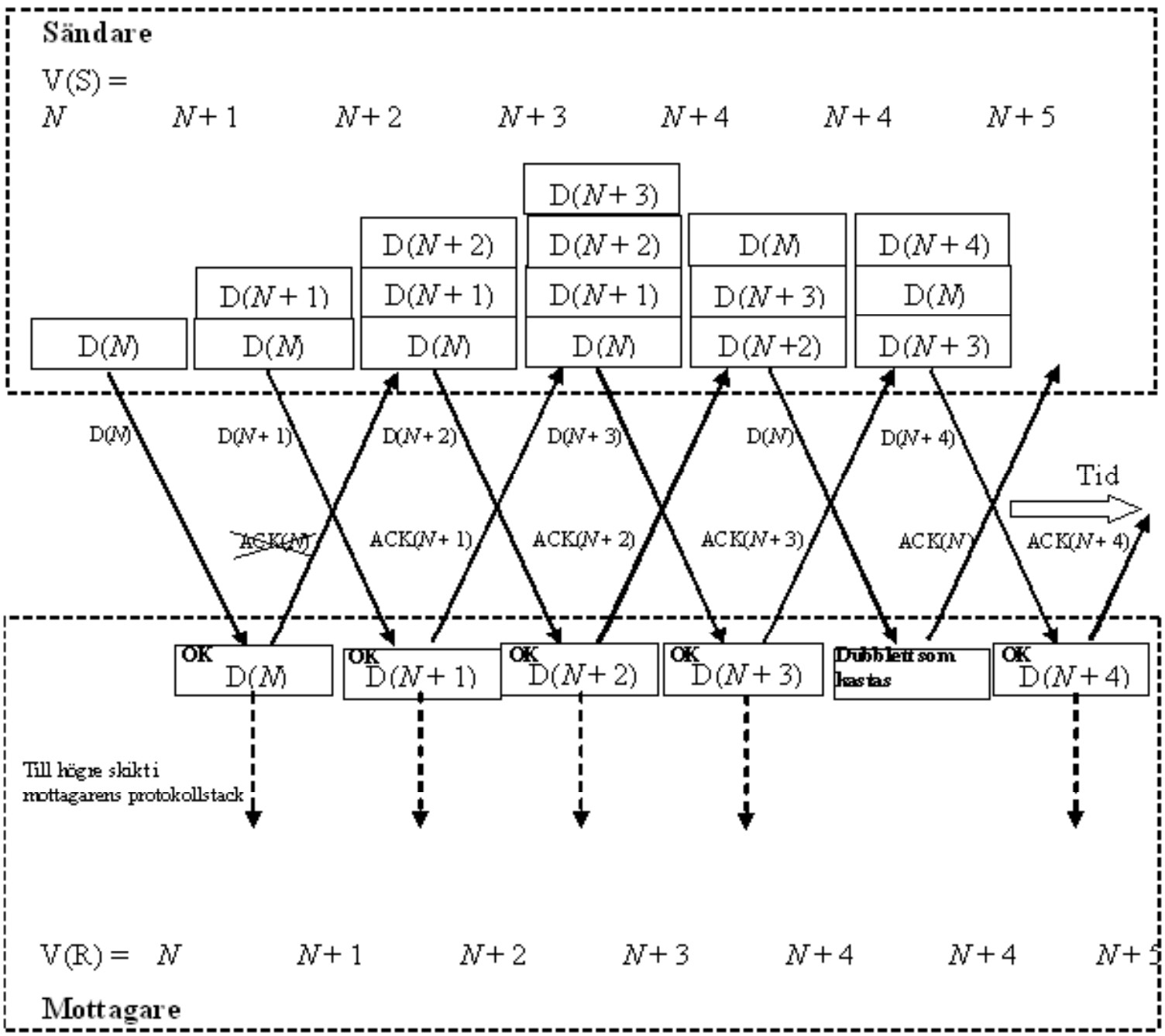
Selective repeat with implicit retransmission (2)

Speciella egenskaper:

- Om meddelande *N* + 1 kvitteras innan meddelande *N*, sänds meddelande *N* på nytt.
- Fel på ACK uppfattas av sändaren som ingen ACK.
- Dubbla försändelser kan hanteras av mottagaren.
- Det behövs inga timers för att kontrollera ACK:ar.

  (Däremot kan timers användas för kunna att upptäcka avbrutna sändningar.)


Sändare
V(S) =
N
N+1
N+2
N+3
N+4
N+5
N+5
D(M)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
D(N+3)
D(N+1)
D(N+2)
D(N+4)
D(N+1)
D(N+2)
D(N+3)
D(M)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
D(N+1)
Tid
ACK(M)
ACK(N+2)
ACK(N+3)
ACK(N+4)
ACK(N+1)
OK
D(M)
Felaktig eller saknas
OK
D(N+2)
OK
D(N+3)
D(N+2)
OK
D(N+4)
D(N+3)
D(N+2)
OK
D(N+1)
D(N+4)
D(N+3)
D(N+2)
Till högre skikt i mottagarens protokollstack
V(R) = N
N+1
N+1
N+1
N+1
N+1
N+5
Mottagare


Sändare
V(S) =
N
N+1
N+2
N+3
N+4
N+4
N+5
D(N+3)
D(N+2)
D(N+2)
D(M)
D(N+4)
D(N+1)
D(N+1)
D(N+1)
D(N+3)
D(M)
D(M)
D(M)
D(M)
D(M)
D(N+2)
D(N+3)
D(M)
D(N+1)
D(N+2)
D(N+3)
D(M)
D(N+4)
Tid
ACK(M)
ACK(N+1)
ACK(N+2)
ACK(N+3)
ACK(N)
ACK(N+4)
OK
D(M)
OK
D(N+1)
OK
D(N+2)
OK
D(N+3)
Dubblett som kastas
OK
D(N+4)
Till högre skikt i mottagarens protokollstack
V(R) = N
N+1
N+2
N+3
N+4
N+4
N+5
Mottagare

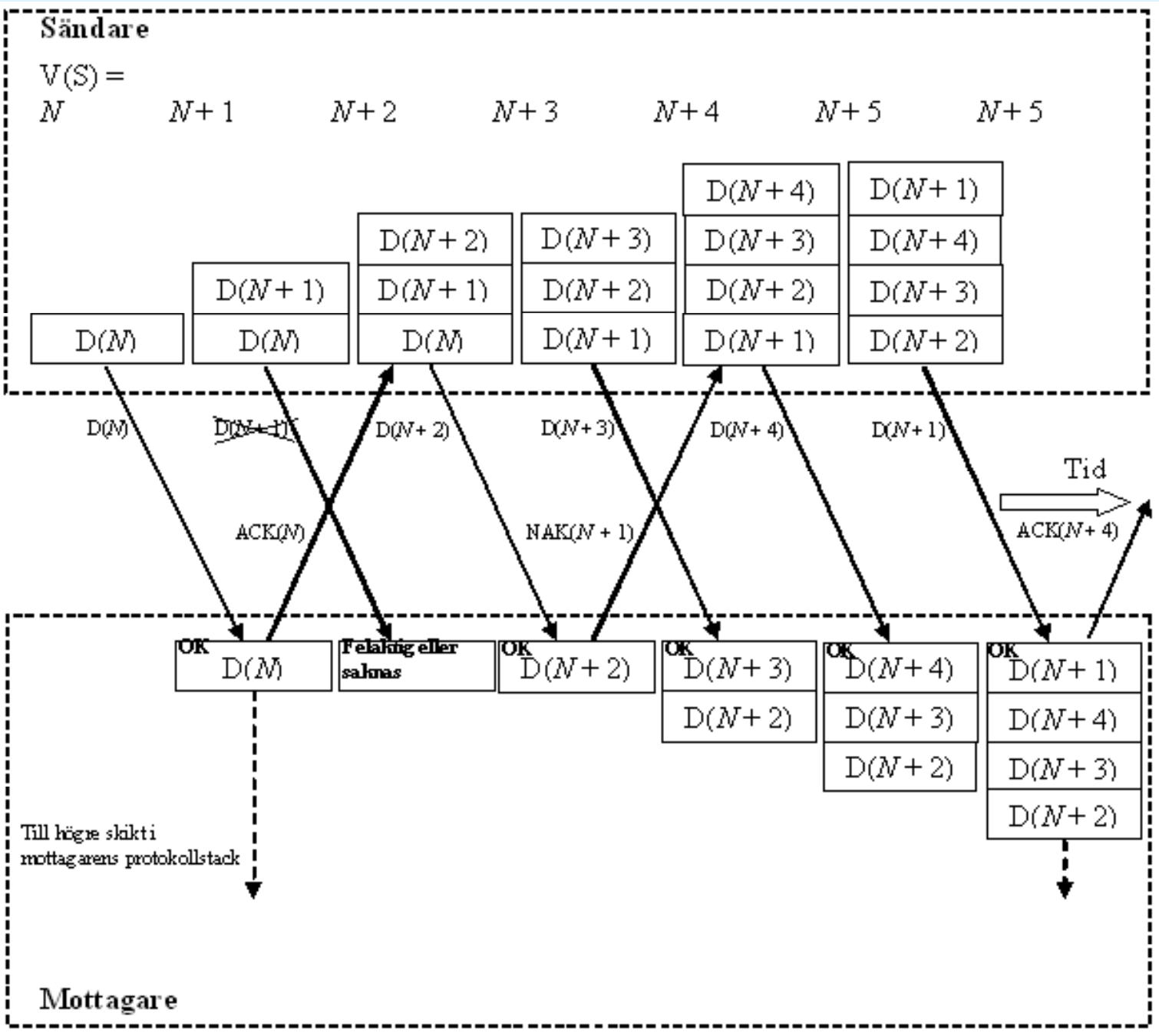
Selective repeat with explicit request (1)

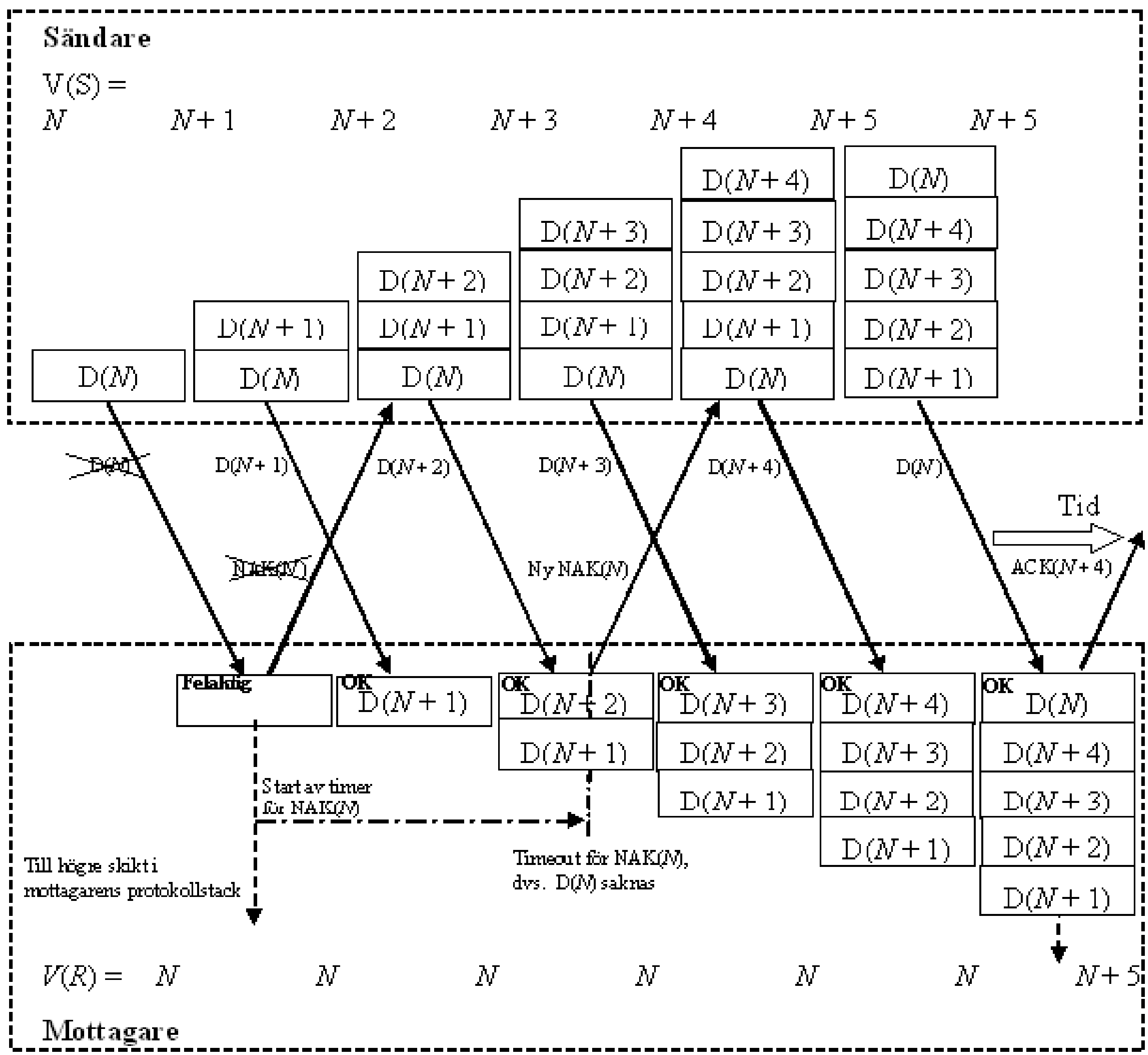
- Enbart felaktiga meddelanden (ramar, paket, segment etc.) sänds om vid begäran.
- Explicit request betyder att NAK används.

Selective repeat with explicit request (2)

Speciella egenskaper:

- ACK(N) kvitterar samtliga meddelanden till och med meddelande N.
- Mottagaren har timer för att förhindra uteblivet meddelande vid ev. fel på NAK.


Sändare
V(S) =
N
N+1
N+2
N+3
N+4
N+5
N+5
D(N)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
D(N+1)
D(N+2)
D(N+3)
D(N+1)
D(N)
D(N)
D(N)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
D(N+1)
ACK(N)
NAK(N+1)
ACK(N+4)
Tid
OK
D(N)
Felaktig eller saknas
OK
D(N+2)
OK
D(N+3)
D(N+2)
OK
D(N+4)
D(N+3)
D(N+2)
OK
D(N+1)
D(N+4)
D(N+3)
D(N+2)
Till högre skikt i mottagarens protokollstack
Mottagare

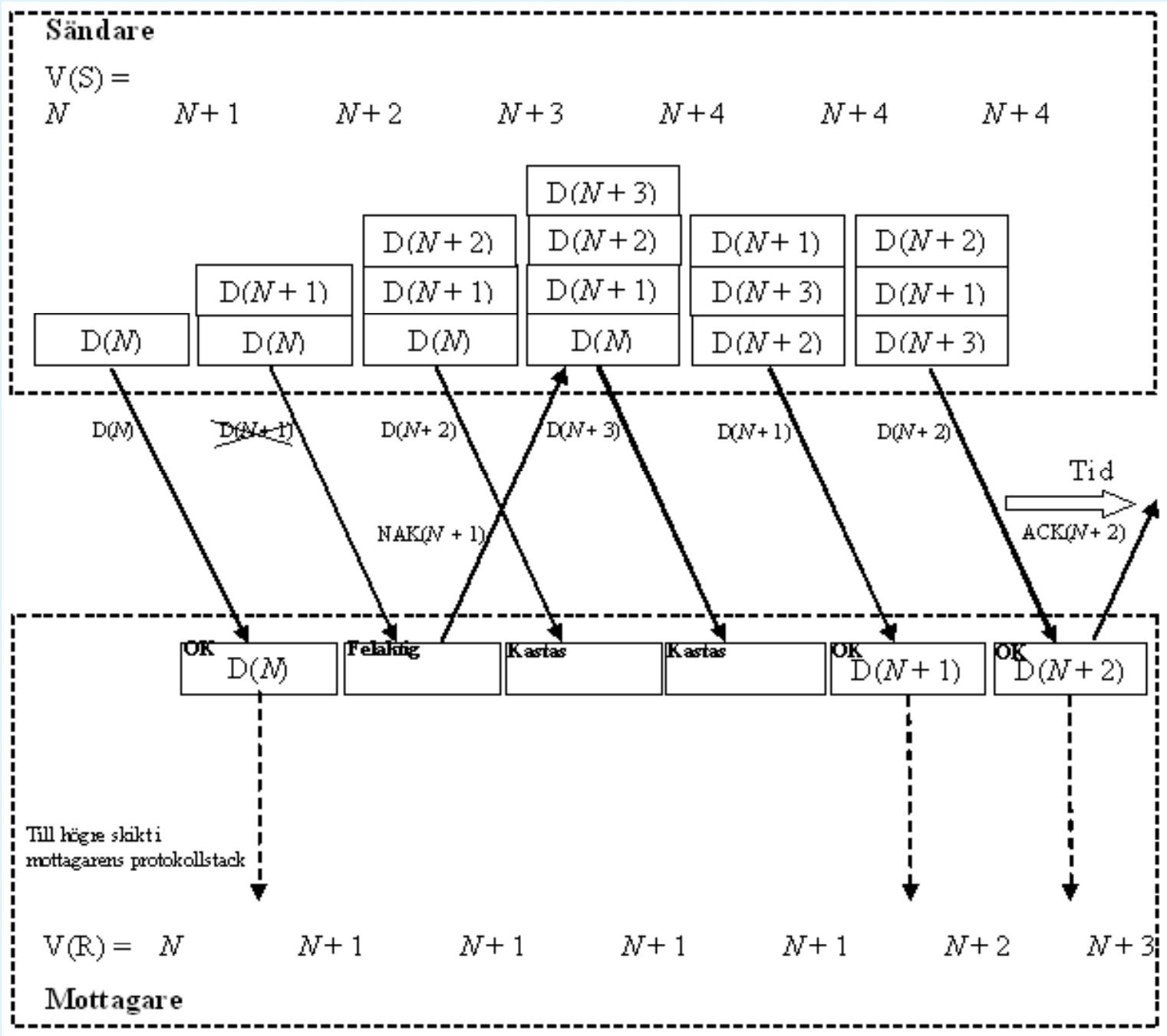

Sändare
V(S) =
N
N+1
N+2
N+3
N+4
N+5
N+5
D(N)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
D(N)
D(N+1)
D(N+2)
D(N+3)
D(N+4)
NAK(N)
Ny NAK(N)
ACK(N+4)
Tid
Felaktig
OK
Start av timer
för NAK(N)
Timeout för NAK(N),
dvs. D(N) saknas
Till högre skikt i
mottagarens protokollstack
V(R) = N N N N N N N+5
Mottagare

Go-back-N (1)

- Repeterar samtliga meddelanden från och med ett felaktigt.
- Fördel: mottagningsbufferten behöver plats för ett enda meddelande.

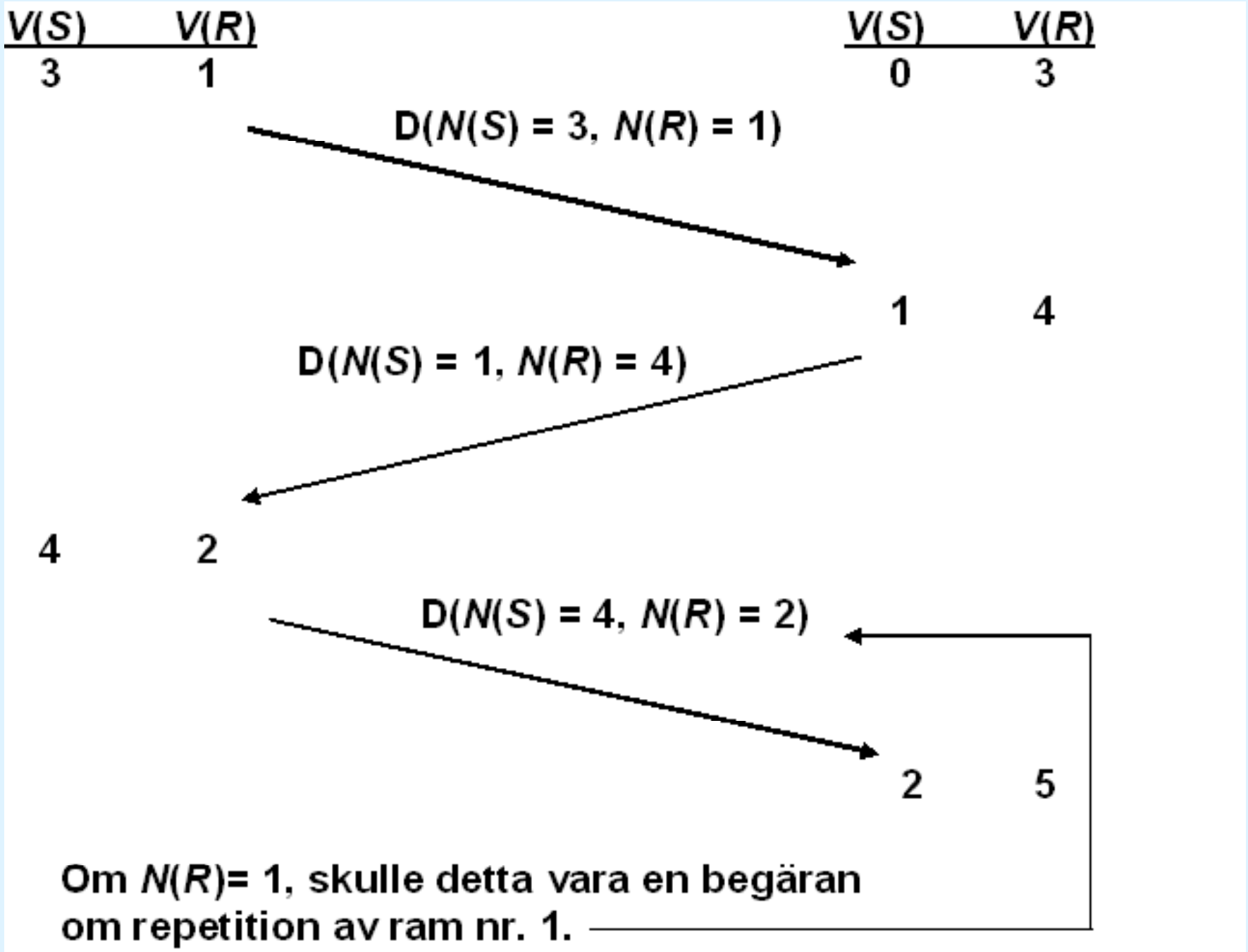
Go-back-N (2)

- Både ACK och NAK används.
- NAK(*N*) medför omsändning från och med meddelande *N*.
- ACK(*N*) och NAK(*N* + *1*) ger implicit kvittering för alla meddelanden till och med meddelande *N*.
- Varje meddelande som sänds har en timer hos sändaren så att utebliven ACK/NAK ger omsändning.


Sändare
V(S) =
N
N+1
N+2
N+3
N+4
N+4
N+4
D(N)
D(N+1)
D(N+2)
D(N+3)
D(N+1)
D(N+2)
D(N+2)
D(N+3)
D(N+1)
D(N+3)
D(N+2)
D(N+2)
D(N+1)
D(N+3)
D(N)
D(N+1)
D(N+2)
D(N+3)
D(N+2)
NAK(N+1)
Tid
ACK(N+2)
OK
D(N)
Felaktig
Kastas
Kastas
OK
D(N+1)
OK
D(N+2)
Till högre skikt i
mottagarens protokollstack
V(R) =
N
N+1
N+1
N+1
N+1
N+2
N+3
Mottagare

Piggyback acknowledgement

- Kvittenser inbäddade i nyttomeddelanden
- Onödig trafik reduceras


V(S) V(R)
3 1
V(S) V(R)
0 3
D(N(S) = 3, N(R) = 1)
1 4
D(N(S) = 1, N(R) = 4)
4 2
D(N(S) = 4, N(R) = 2)
2 5
Om N(R)= 1, skulle detta vara en begäran
om repetition av ram nr. 1.

Hammingavstånd

- *HD* = min(alla avstånd i en mängd av kodord)

- Om *HD* = *u* + 1,

  kan *u* st. bitfel upptäckas i ett felaktigt kodord.

- Om *HD* = 2*r* + 1,

  kan *r* st. bitfel korrigeras i ett felaktigt kodord.

Exempel på blockparitet (blocksumma)

	00000010	= STX
	10101000	1:a tecknet
	00100000	2:a tecknet
	10101101	3:e tecknet
	11100011	4:e tecknet
⊕	**1**0000011	= ETX
BCC =	**1**1000111	

Cyclic Redundancy Check (CRC)

- Används i många typer av LAN

  Exempelvis i Ethernet.

- Felsökning

- Felrättning

  CRC integrerad med felrättande kod

Generatorpolynom

- Bestämmer checksummans tillförlitlighet
- CRC-12: $G = x^{12} + x^{11} + x^3 + x^2 + 1$
- CRC-16: $G = x^{16} + x^{15} + x^2 + 1$
- CRC-CCITT: $G = x^{16} + x^{12} + x^5 + 1$
- CRC-32: $G = x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

CRC-beräkningar

$$\frac{D \cdot 2^n}{G} = Q \oplus \frac{R}{G} \qquad \frac{13}{4} = 3 + \frac{1}{4}$$

Sändaren skickar

$$T = D \cdot 2^n \oplus R$$

Mottagaren testar

$$\frac{T'}{G} = Q \oplus \frac{E}{G}$$

Om $E = 0$, felfri övering.

Feltyper med CRC

Med ett bra val av generatorpolynom kan mottagaren upptäcka följande:

- alla en-bits fel
- alla två-bitars fel
- alla udda antal bitar fel
- alla felskurar < $n + 1$ bitar
- de flesta felskurar ≥ $n + 1$ bitar

Reed-Solomon (RS) (1)

- Används för datakommunikation och lagringsmedia

  Det gör CRC också!

- Felsökning

- Felrättning

Reed-Solomon (RS) (2)

- RS(n, k)

  Antal data (s-bitars symboler): k

  Antal data inklusive checksummorna: n

  Antal checksummor (s-bitars symboler): $n - k$

Reed-Solomon (RS) (3)

- Maximalt antal data som kan rättas (t)

  $2t = n - k$

  dvs.

  $t = (n - k)/2$

Flödesreglering

- Programstyrd
- Hårdvarustyrd
- Fönsterteknik

Fönsterteknik

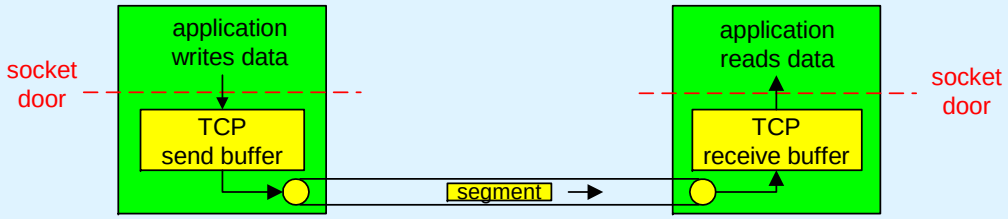
Omsändningsmetod	Sändningsfönster	Mottagningsfönster
Idle repeat request	1	1
Selective repeat	$k/2$	$k/2$
Go-back-N	$k-1$	1

Antal sekvensnummer: $k = 2^n$

TCP: Overview

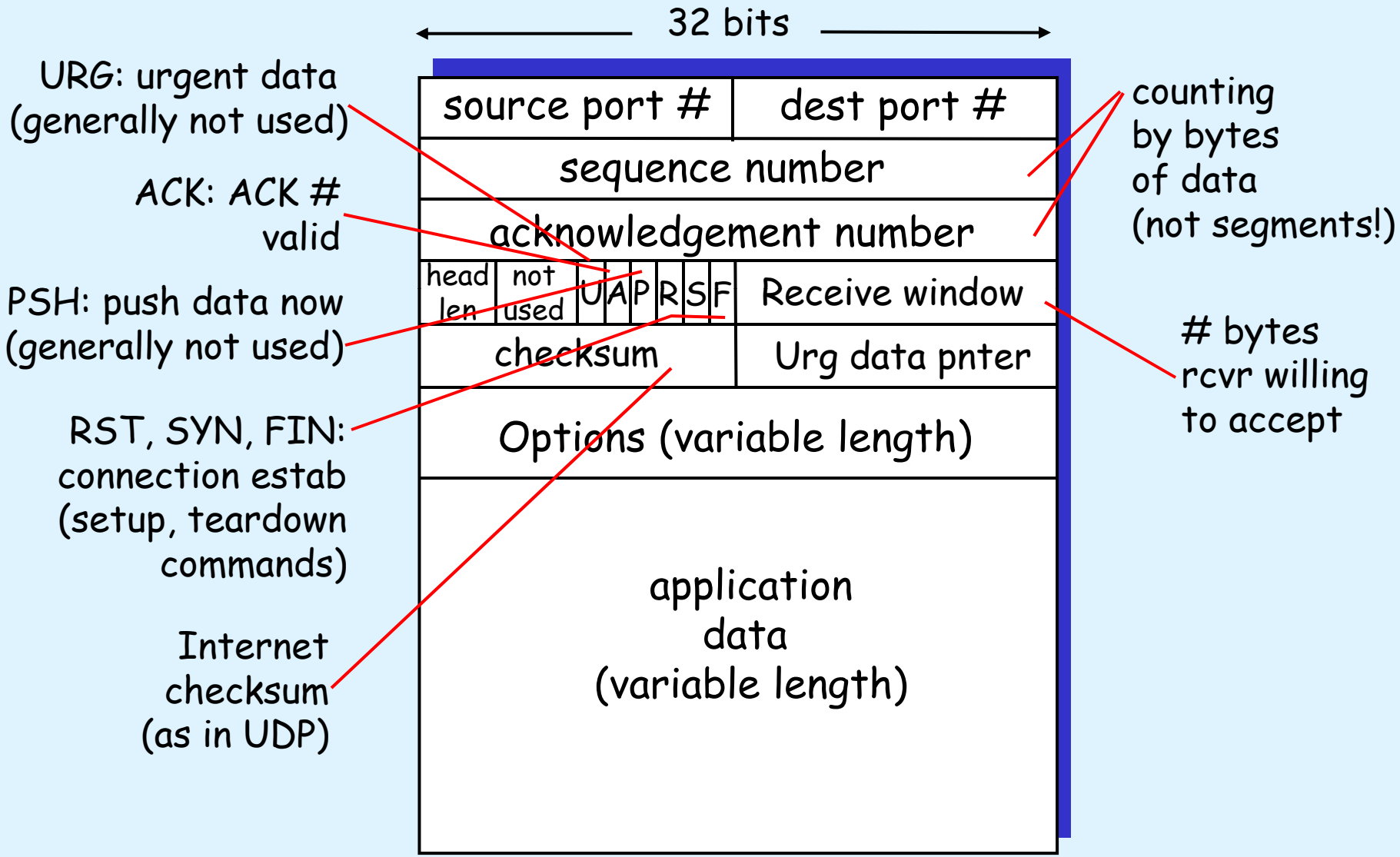
RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*



- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments

- initialize TCP variables:
  - seq. #s
  - buffers, flow control info (e.g. **`RcvWindow`**)

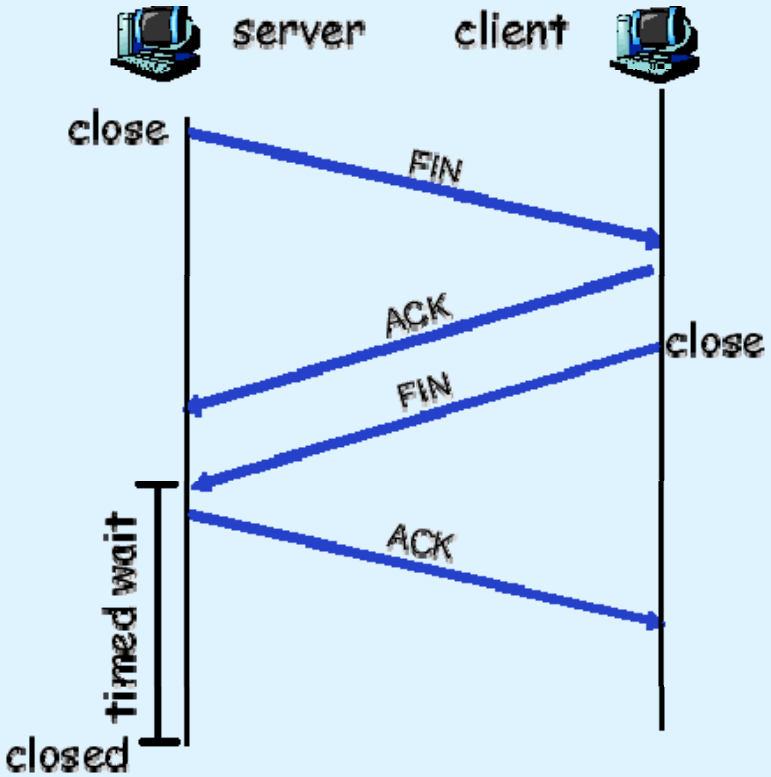
Three way handshake:

Step 1: client host sends TCP SYN segment to server
- specifies initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

TCP Connection Management (cont.)



Step 1: server end system sends TCP FIN control segment to client

Step 2: client receives FIN, replies with ACK. Closes connection, sends FIN.

Step 3: server receives FIN, replies with ACK.

- Enters "timed wait" - will respond with ACK to received FINs

Step 4: client, receives ACK. Connection closed.

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?

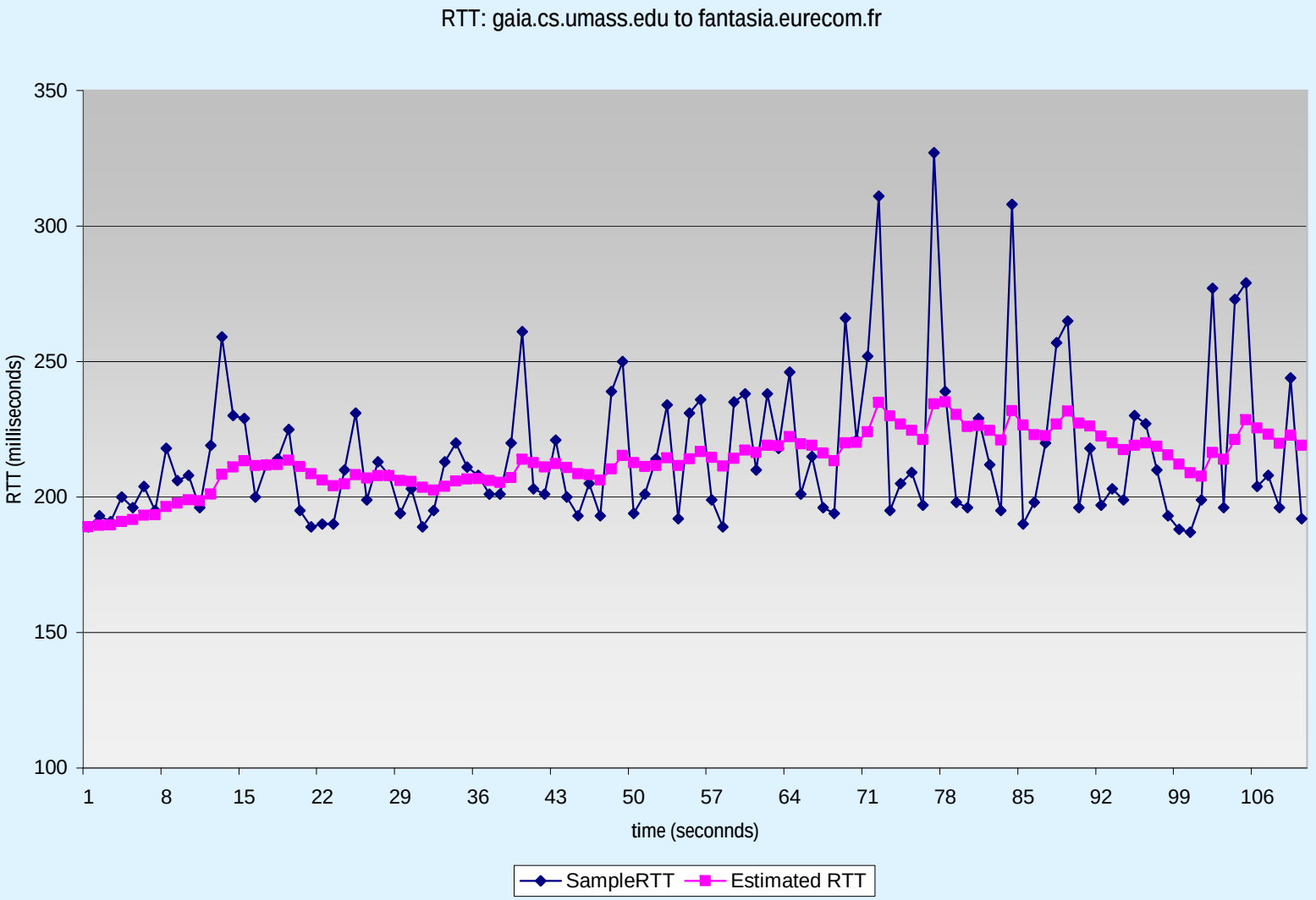
- **SampleRTT**: measured time from segment transmission until *ACK* receipt
- **SampleRTT** will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current **SampleRTT**

TCP Round Trip Time and Timeout

$$\mathbf{EstimatedRTT_{new} = (1-\alpha)\cdot EstimatedRTT_{old} + \alpha\cdot SampleRTT}$$

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$

Example RTT estimation:



TCP Round Trip Time and Timeout

Setting the timeout

- **EstimtedRTT** plus "safety margin"
  - large variation in **EstimatedRTT** -> larger safety margin
- first estimate of how much SampleRTT deviates from EstimatedRTT:

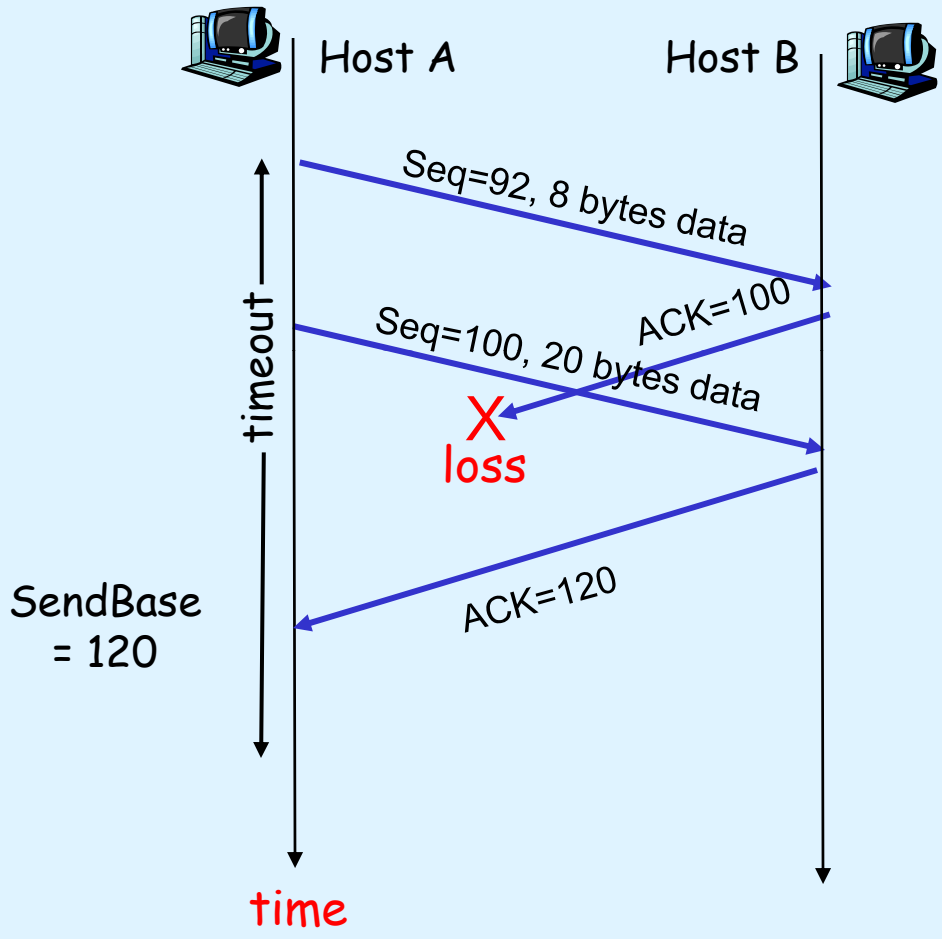
$$\mathbf{DevRTT_{new} = (1-\beta)\cdot DevRTT_{old} + \beta \cdot |SampleRTT-EstimatedRTT|}$$

(typically, β = 0.25)

Then set timeout interval:

$$\mathbf{TimeoutInterval = EstimatedRTT + 4\cdot DevRTT}$$

TCP retransmission scenarios (more)

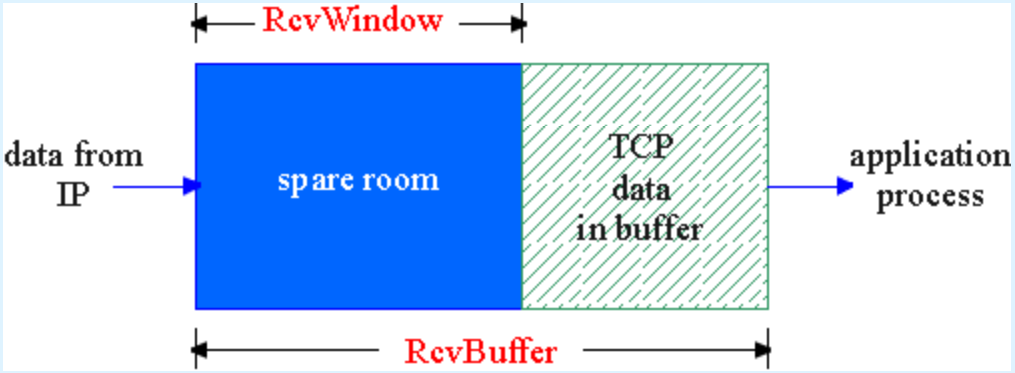


Cumulative ACK scenario

Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

TCP Flow control: how it works



- Rcvr advertises spare room by including value of **RcvWindow** in segments
- Sender limits unACKed data to **RcvWindow**
  - guarantees receive buffer doesn't overflow

- spare room in buffer

= RcvWindow

= RcvBuffer-[LastByteRcvd - LastByteRead]

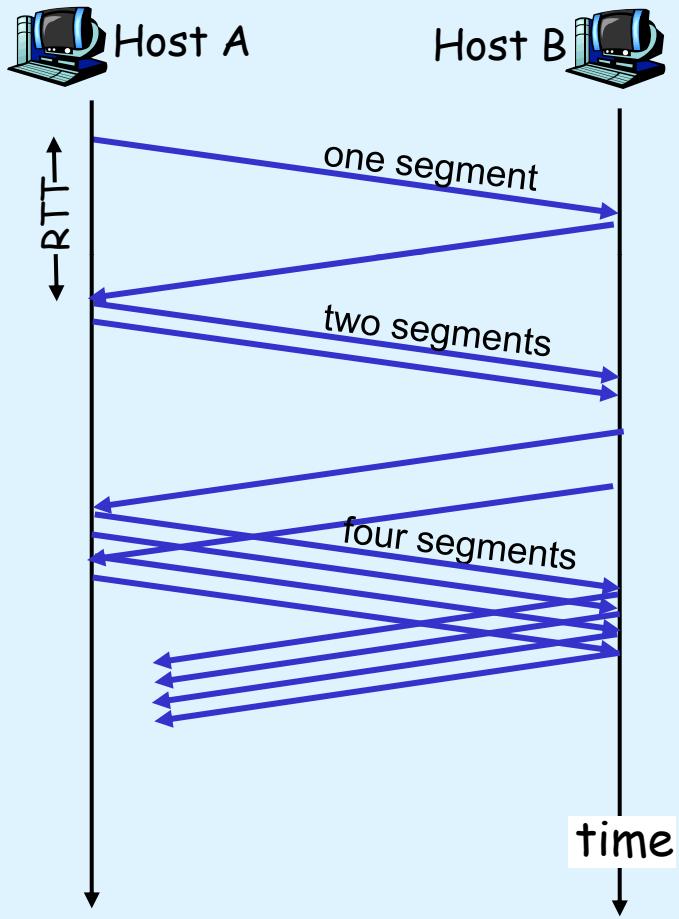
Principles of Congestion Control

Congestion:

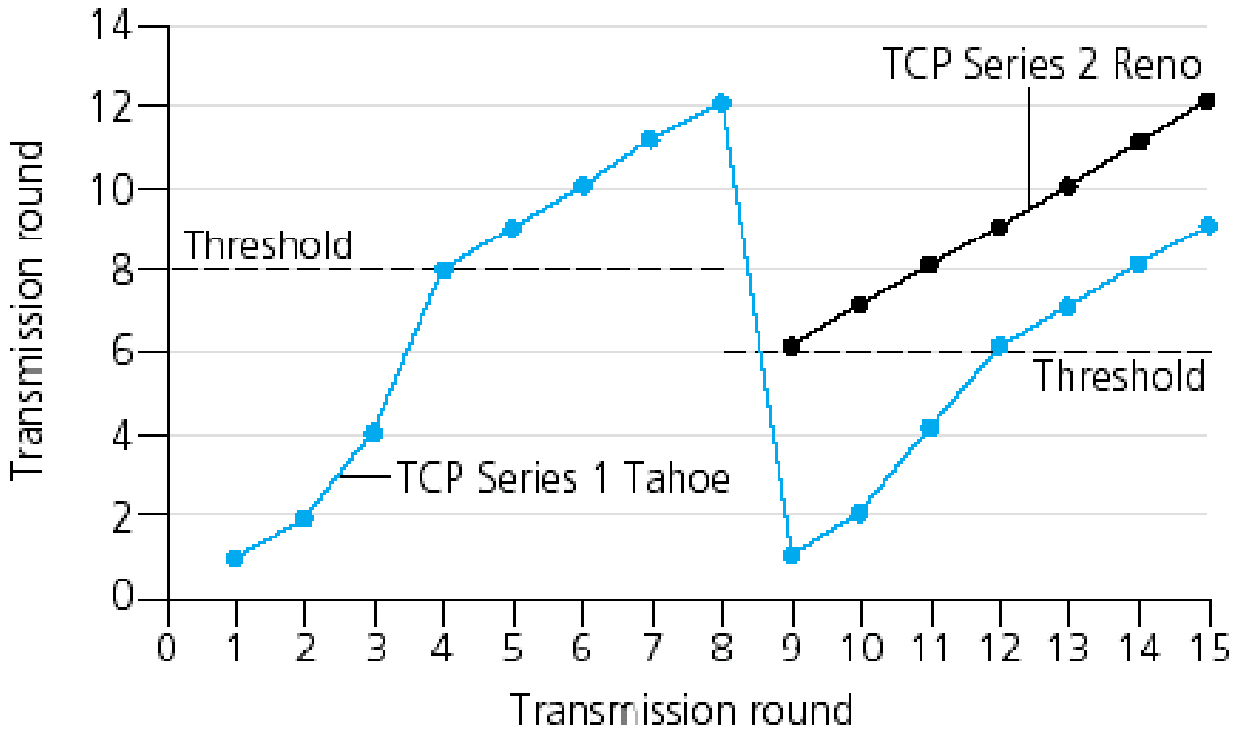
- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
  - double **CongWin** every RTT
  - done by incrementing **CongWin** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



Refinement


14
12
10
8
6
4
2
0
Transmission round
Threshold
TCP Series 2 Reno
Threshold
TCP Series 1 Tahoe
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Transmission round

Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in slow-start phase, window grows exponentially.
- When **CongWin** is above **Threshold**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate *ACK* occurs, **Threshold** set to **CongWin/2** and **CongWin** set to **Threshold**.
- When timeout occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS.