Assignment 3

Lab: Useful Tools

Prof. Ai-Chun Pang

TA / Ping-Yu Yang, Wen-Hsin Wu, Kuang-Hui Huang

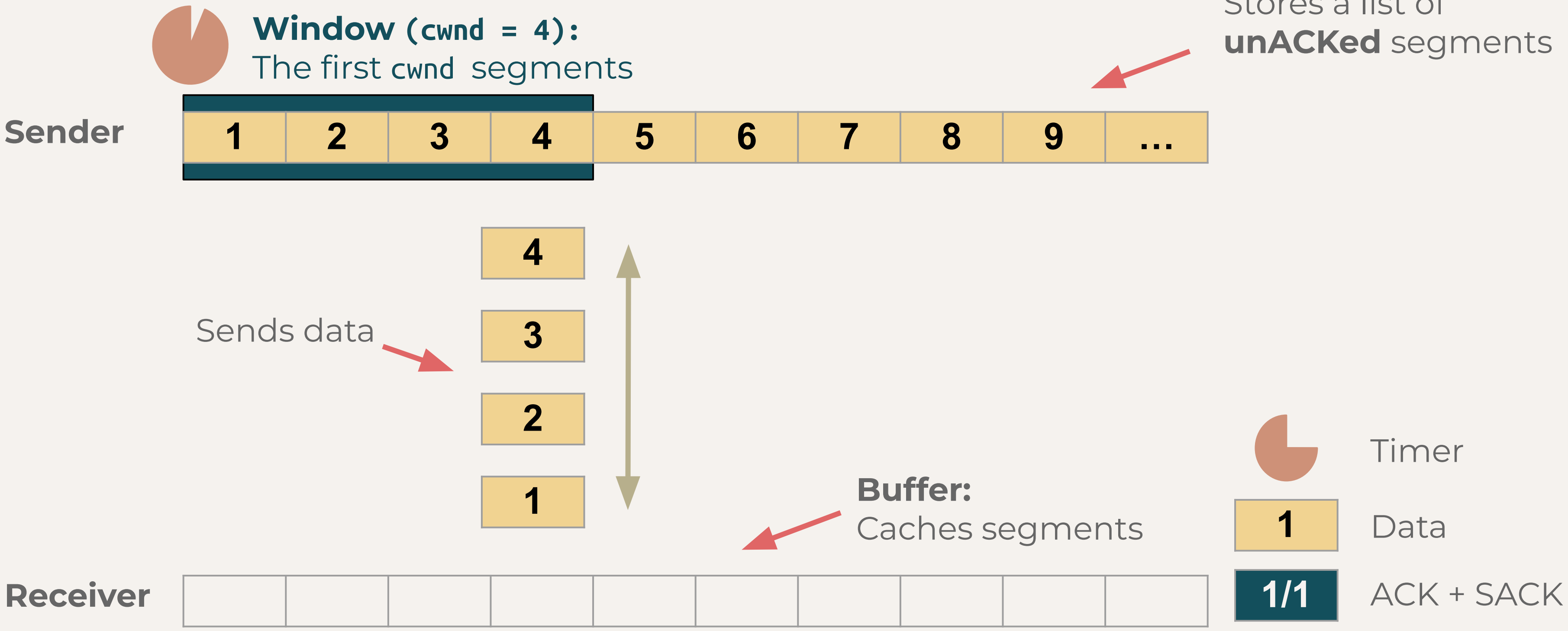
Goals

- UDP socket (file transmission)
- Reliable data transfering (SACK)
- Congestion control

SACK Protocol

GBN + SR

SACK case 1 (working normally)



Window `(cwnd = 4)`:
The first `cwnd` segments

Transmit queue:
Stores a list of
unACKed segments

Sender

Sends data

Buffer:
Caches segments

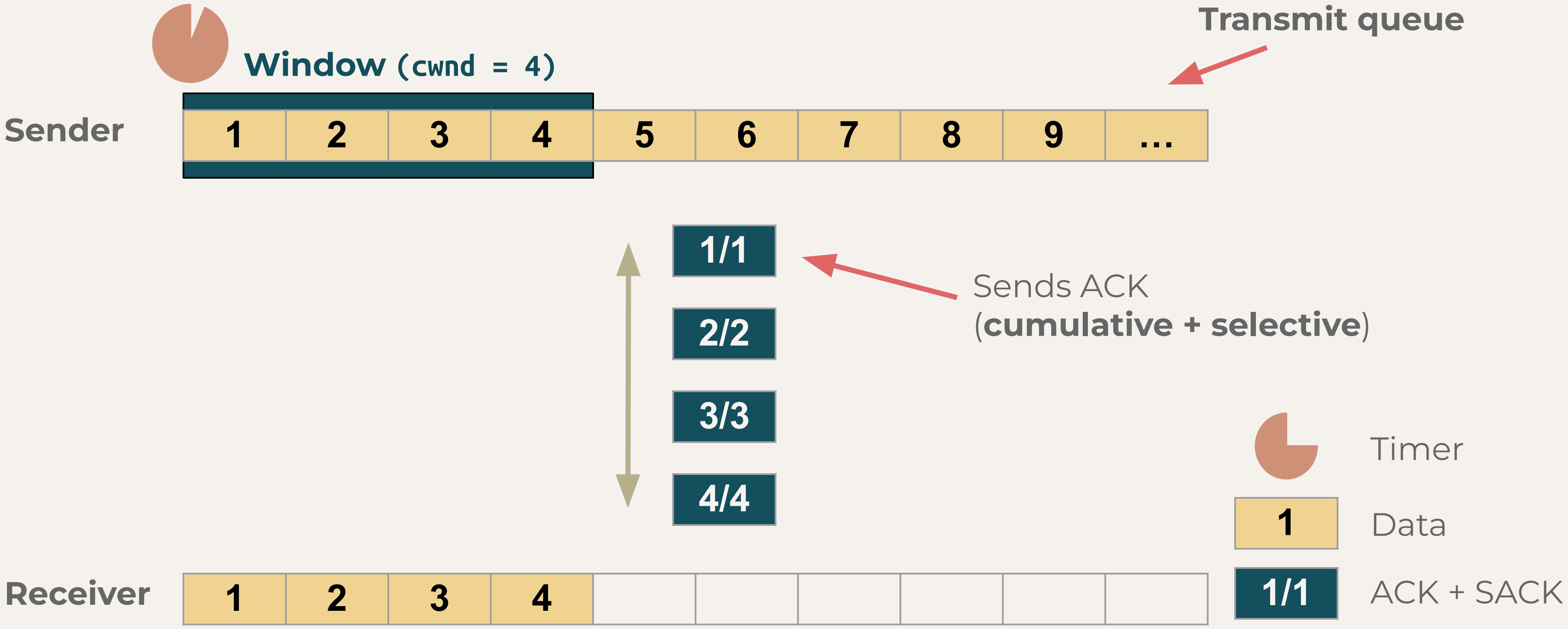
Receiver

Timer

Data

ACK + SACK

SACK case 1 (working normally)



Window (cwnd = 4)

Transmit queue

Sender

1 2 3 4 5 6 7 8 9 ...

1/1

2/2

3/3

4/4

Sends ACK
(**cumulative + selective**)

Receiver

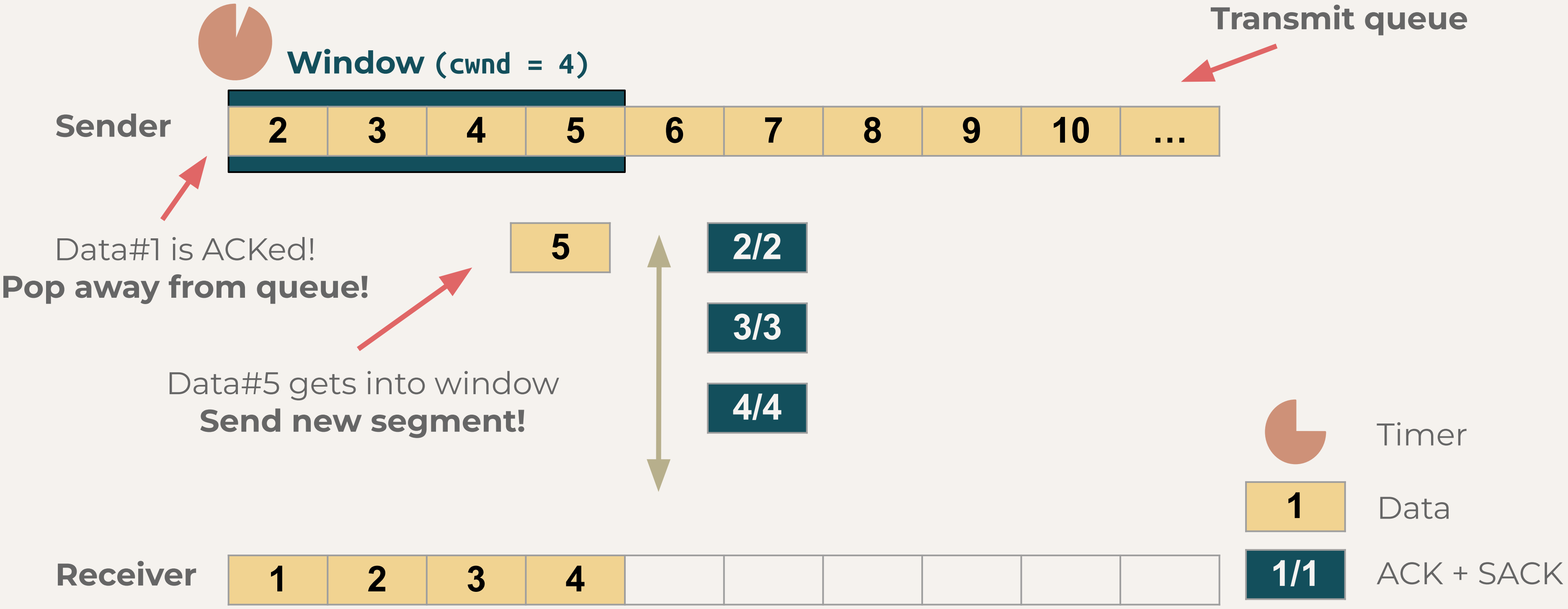
1 2 3 4

Timer

1 Data

1/1 ACK + SACK

SACK case 1 (working normally)



Window `(cwnd = 4)`

Transmit queue

Sender

2	3	4	5	6	7	8	9	10	...

Data#1 is ACKed!
Pop away from queue!

5

Data#5 gets into window
Send new segment!

2/2

3/3

4/4

Receiver

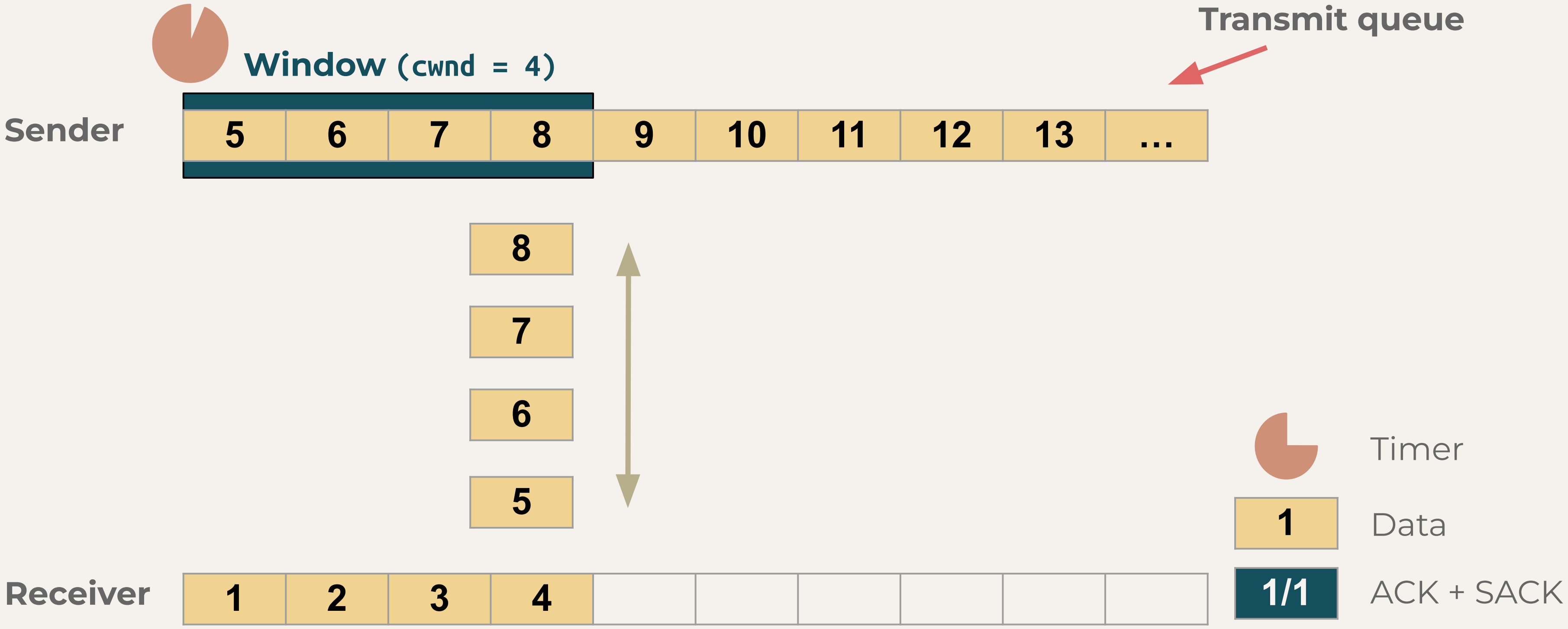
1	2	3	4						

Timer

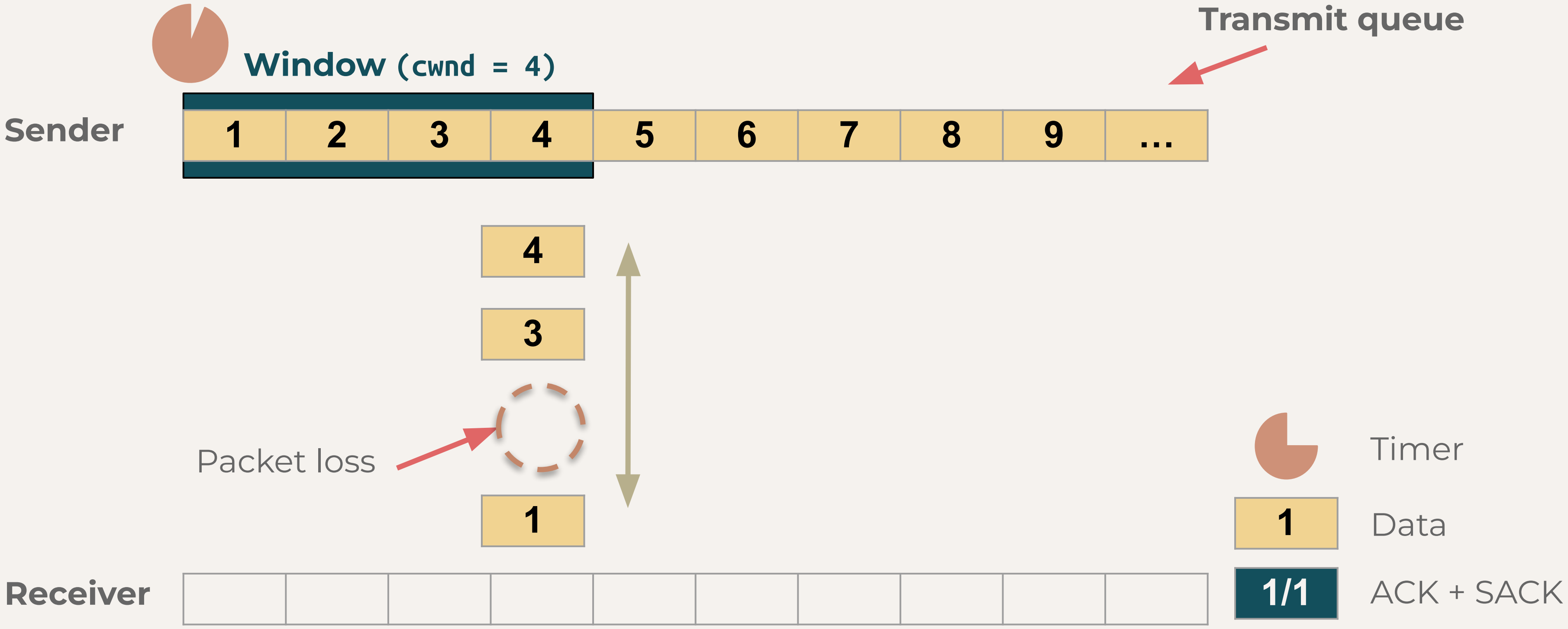
1 Data

1/1 ACK + SACK

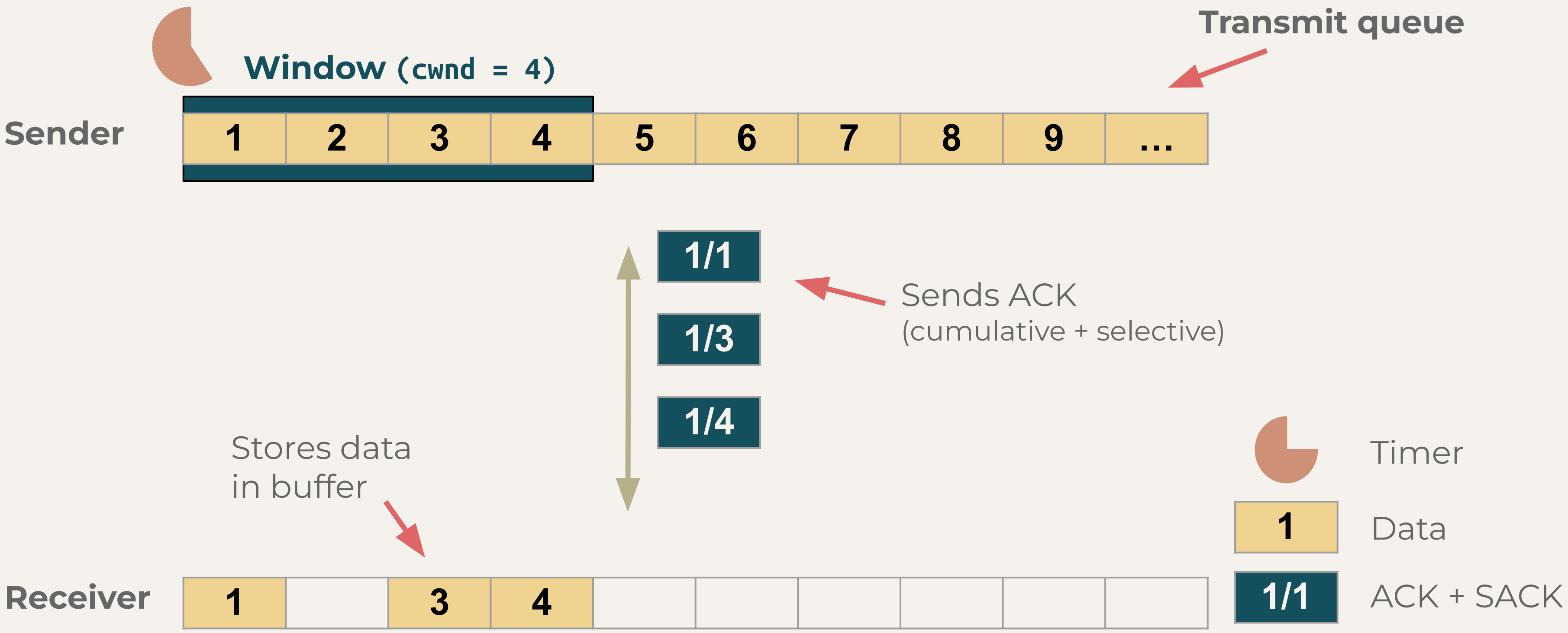
SACK case 1 (working normally)



SACK case 1 (Packet loss)



SACK case 1 (Packet loss)



Transmit queue

Window (cwnd = 4)

Sender

1 2 3 4 5 6 7 8 9 ...

1/1

1/3

1/4

Sends ACK
(cumulative + selective)

Stores data
in buffer

Receiver

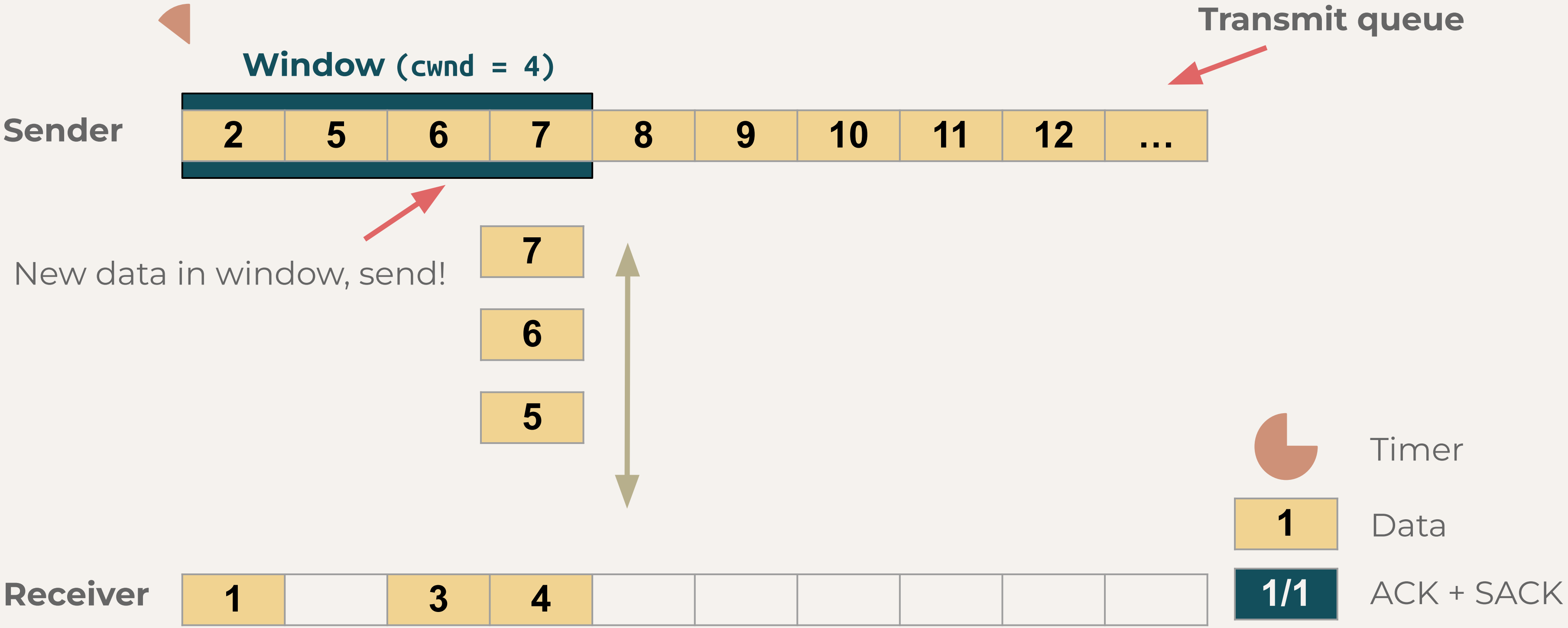
1 3 4

Timer

1 Data

1/1 ACK + SACK

SACK case 1 (Packet loss)



Window (cwnd = 4)

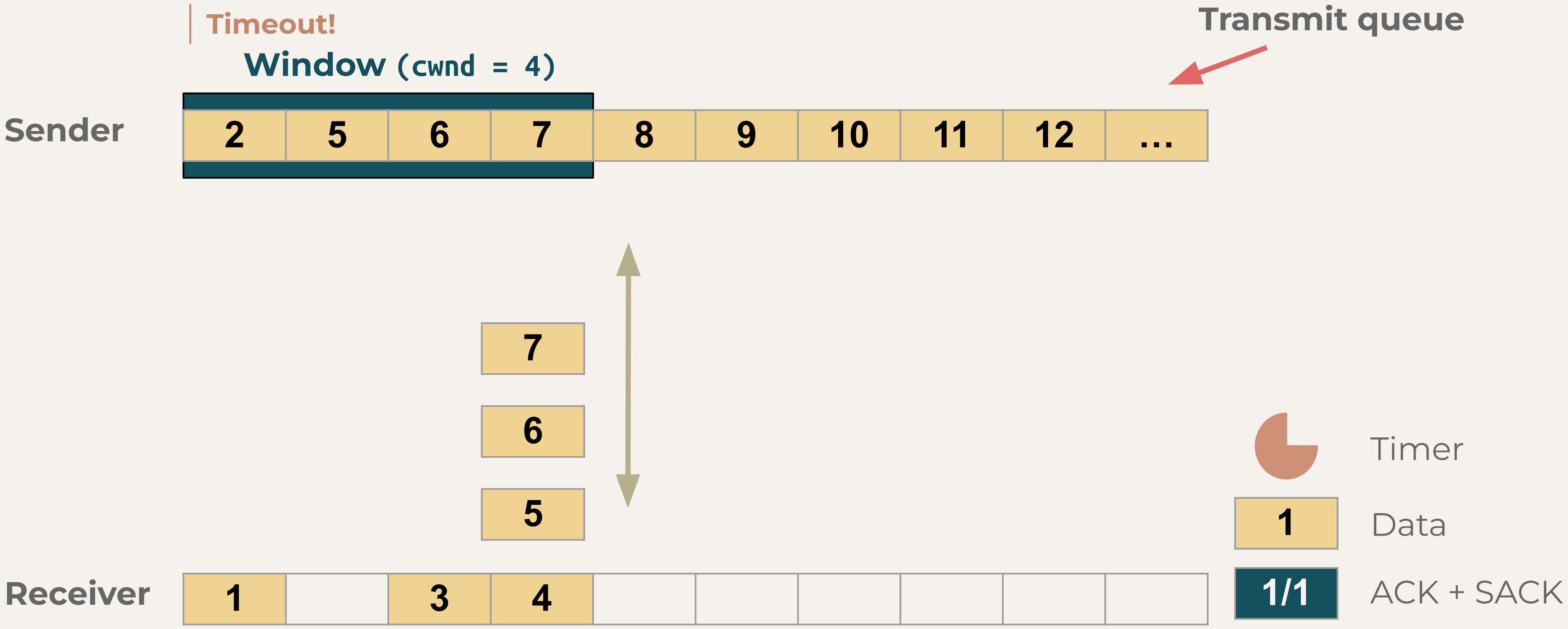
Transmit queue

Sender	2	5	6	7	8	9	10	11	12	...
Receiver	1		3	4						

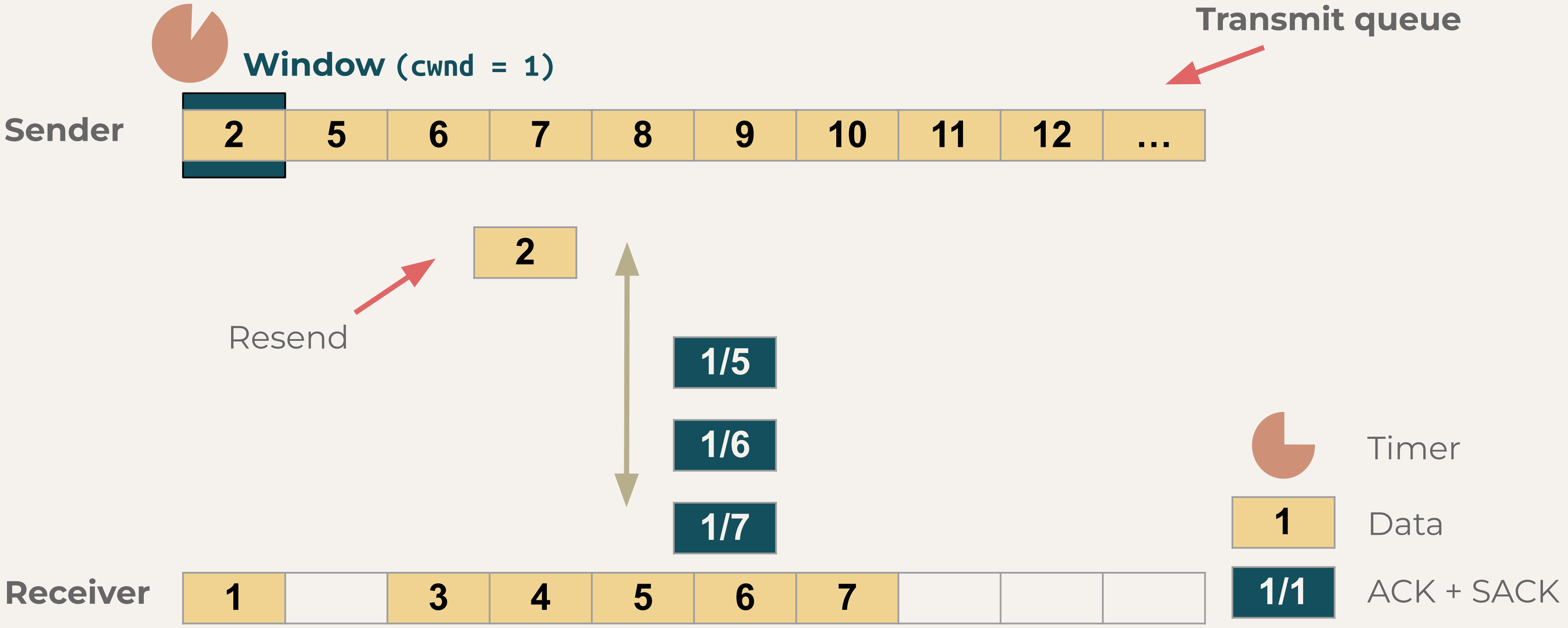
New data in window, send!

	Timer
1	Data
1/1	ACK + SACK

SACK case 1 (Packet loss)



SACK case 1 (Packet loss)



Transmit queue

Window (cwnd = 1)

Sender	2	5	6	7	8	9	10	11	12	...
Receiver	1		3	4	5	6	7			

Resend

2

1/5

1/6

1/7

Timer

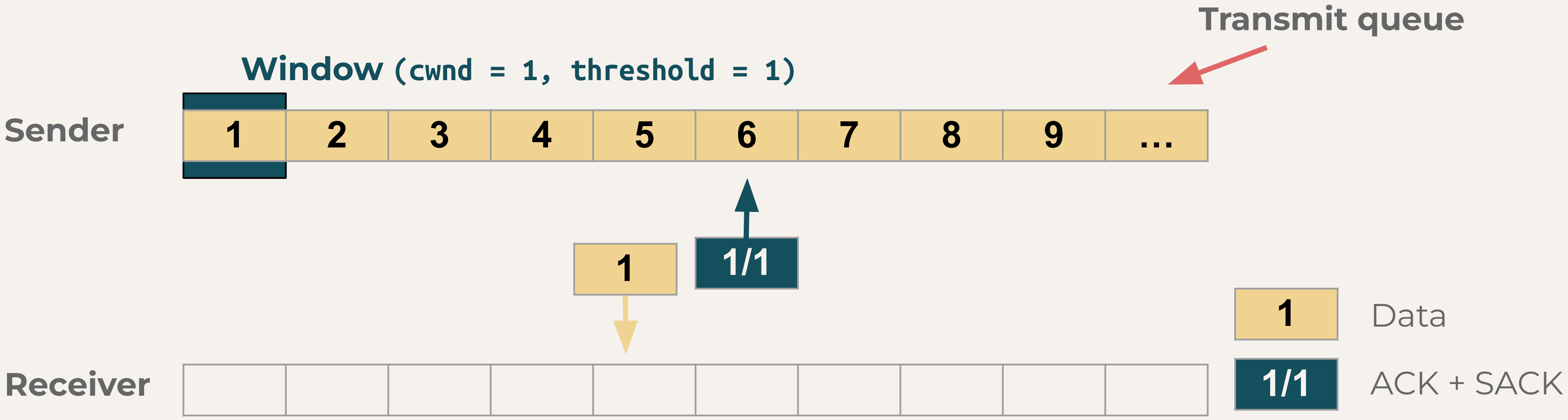
1 Data

1/1 ACK + SACK

SACK with Congestion Control

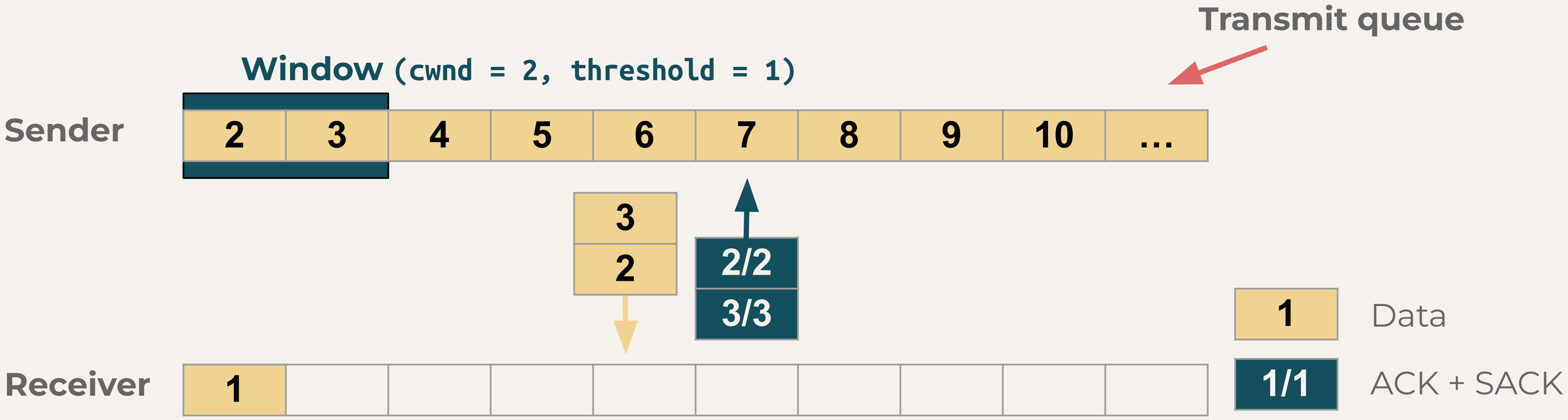
SACK + Congestion Control

- Sends data 1 (`cwnd = 1, threshold = 1`)
- Receives ACK 1/1 (`cwnd = 2, threshold = 1`)



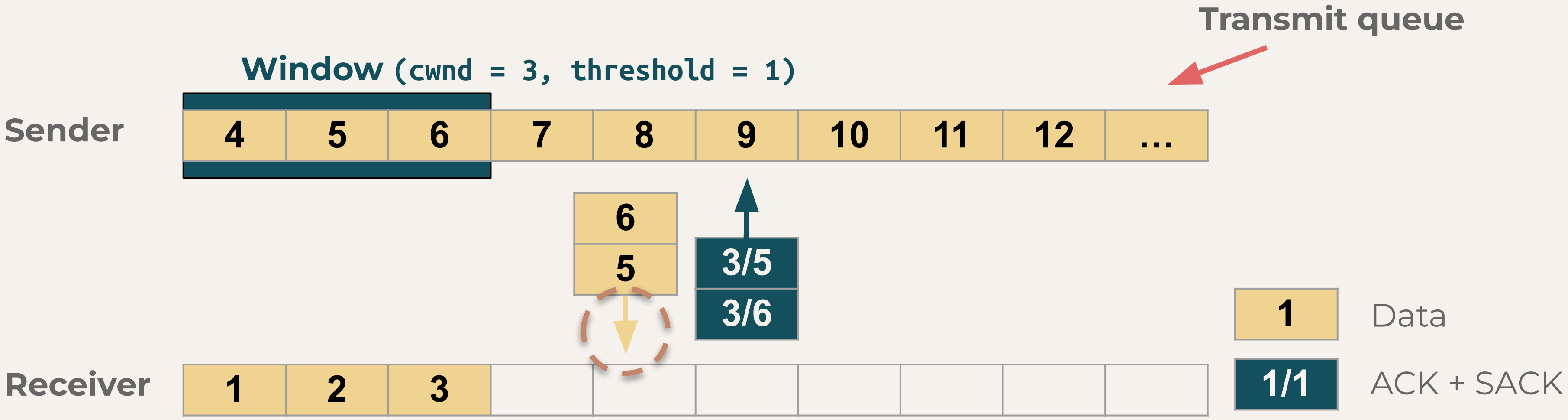
SACK + Congestion Control

- Sends data 2, 3 (`cwnd = 2, threshold = 1`)
- Receives ACK 2/2, 3/3 (`cwnd = 3, threshold = 1`)



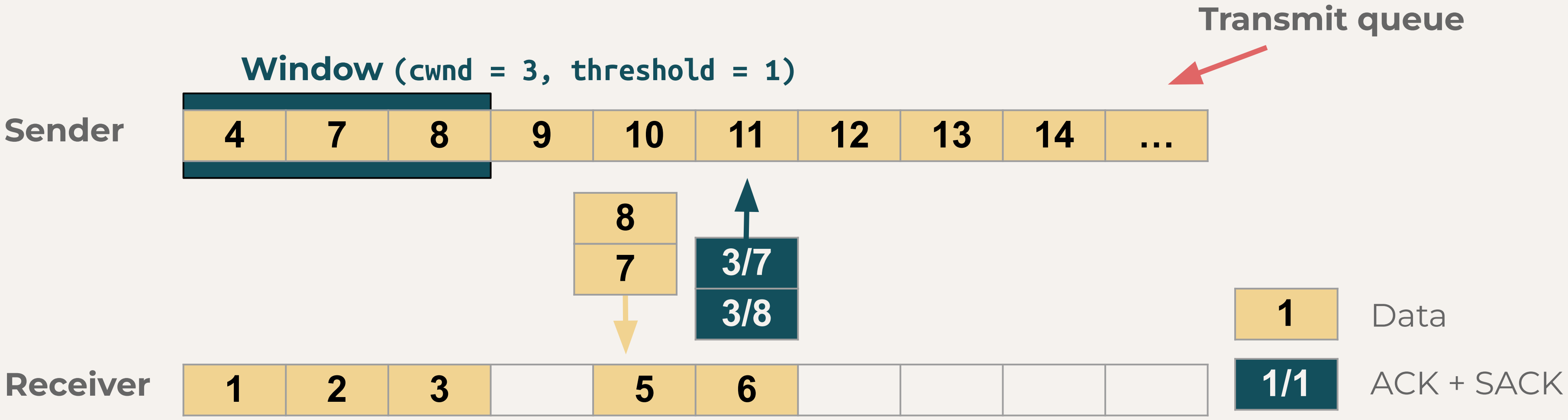
SACK + Congestion Control

- Sends data 4, 5, 6 (`cwnd = 3, threshold = 1`)
- **Lost data 4**, receive ACK 3/5, 3/6. Duplicate cumulative ACK, `cwnd = 3.`



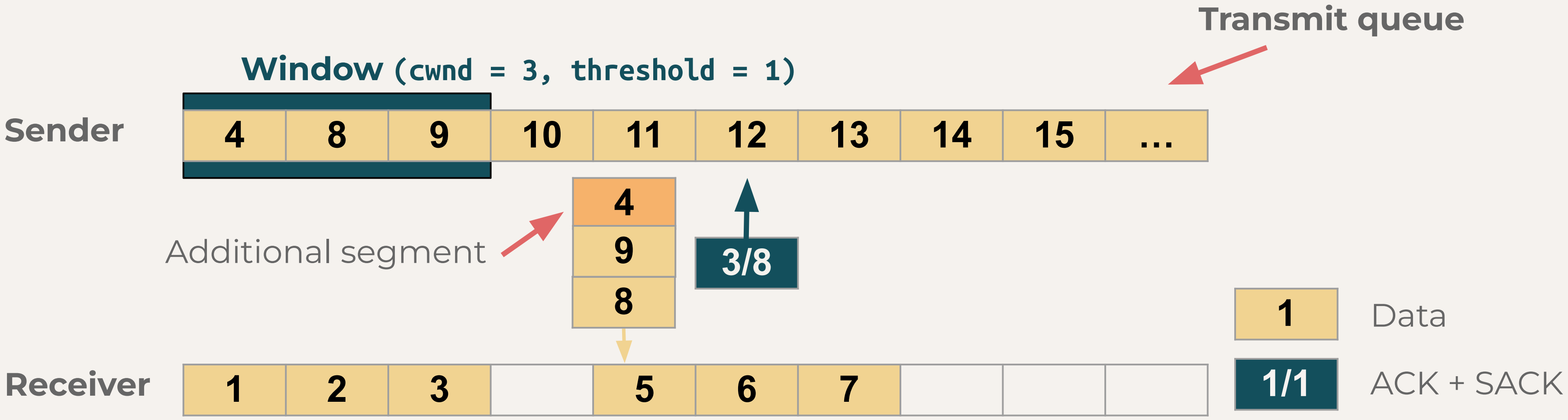
SACK + Congestion Control

- As long as there's new segment in window, the sender will send new segments.



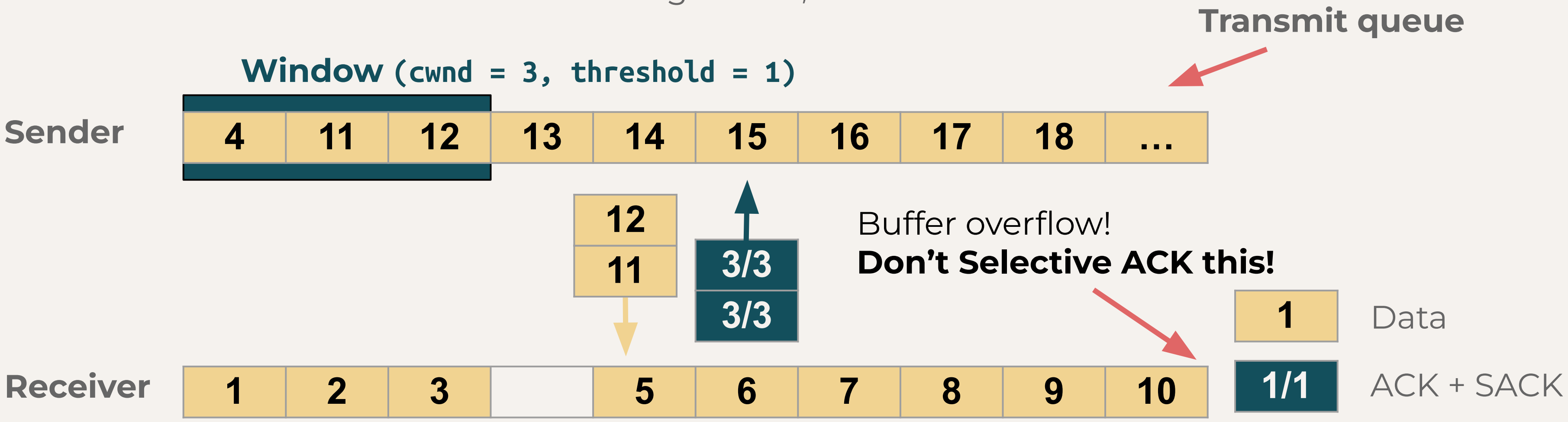
SACK + Congestion Control

- When receiving ACK 3/7, the sender sends Data 9. But **3 duplicate cumulative ACK happened!** (ACK 3/5, 3/6, 3/7)
- To remedy packet loss, the sender **sends an additional segment (Data 4).**



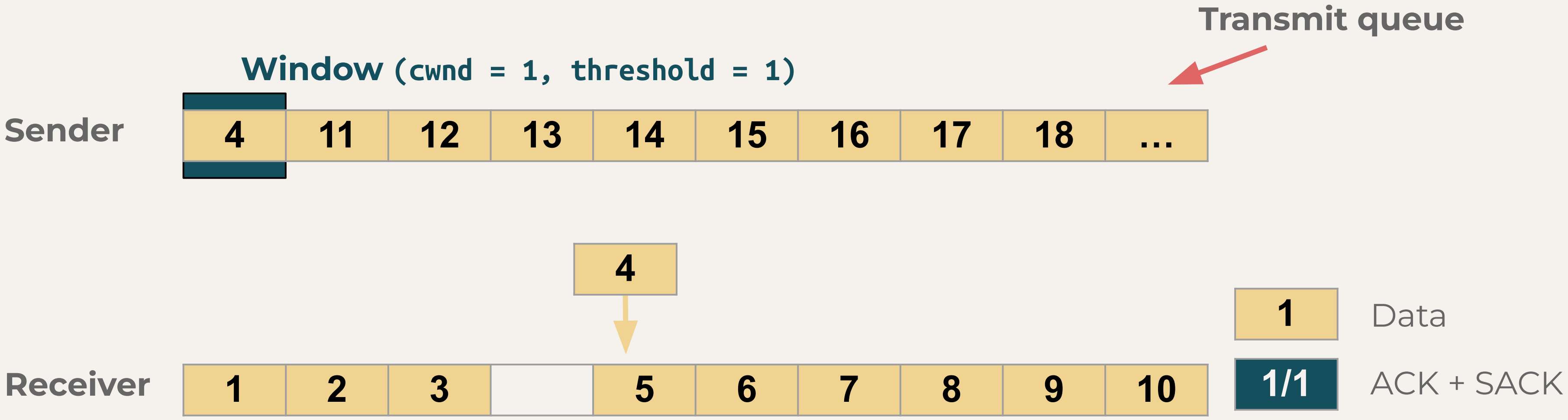
SACK + Congestion Control

- Assuming the additional Data 4 packet is also lost, the sender will still keep sending as long as there's new segments in window.
- **Buffer overflow** when sending data 11, 12. Send back ACK



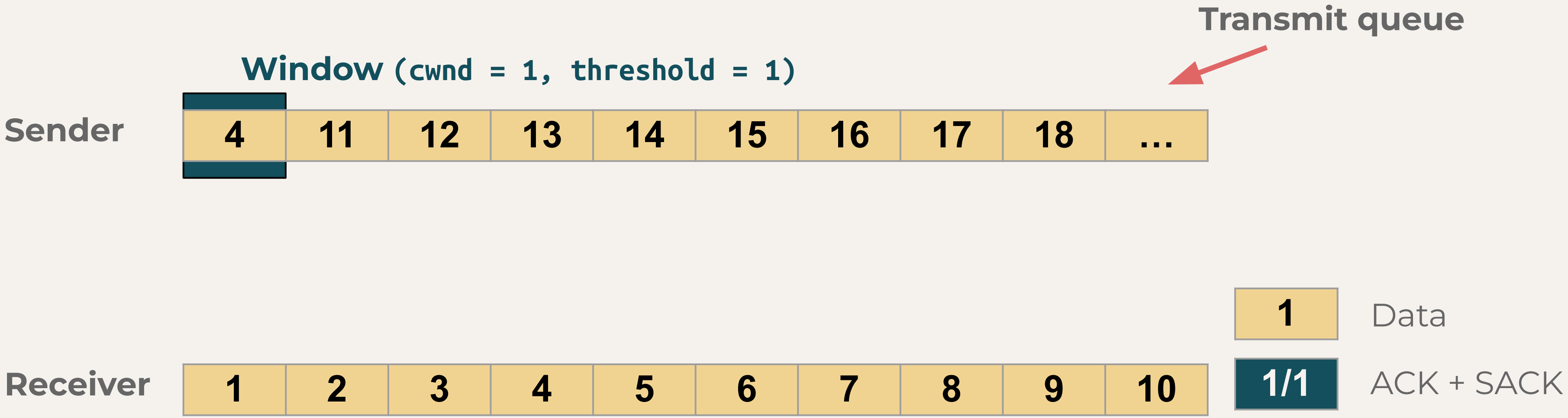
SACK + Congestion Control

- **Timeout happens!** `(cwnd = 1, threshold = 1)`
- Resend packet again



SACK + Congestion Control

- After receiving Data 4, the buffer is filled.



Transmit queue

Window `(cwnd = 1, threshold = 1)`

Sender: 4 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | ...

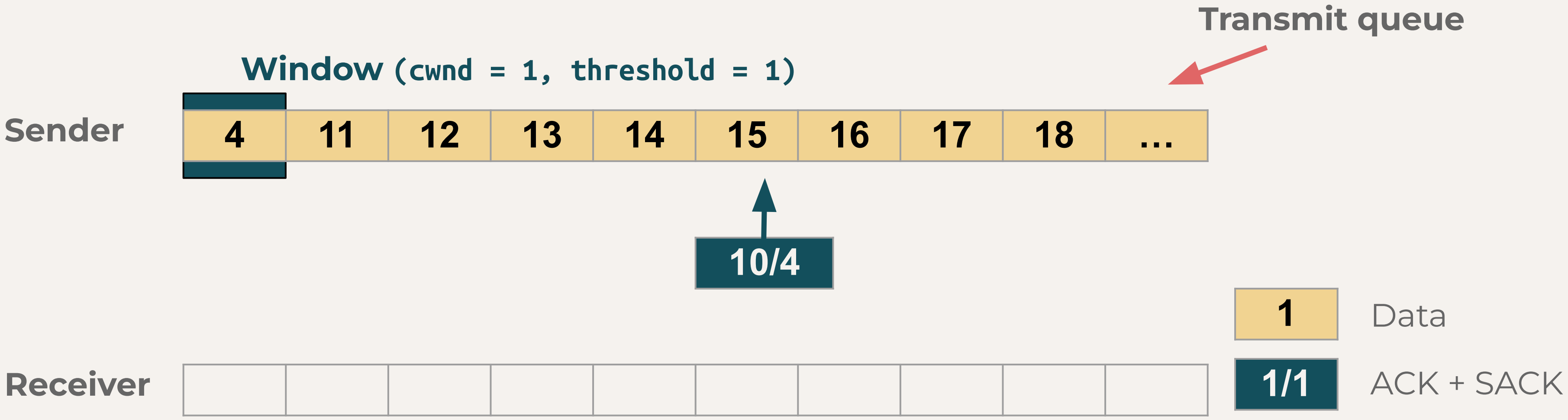
Receiver: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

1 Data

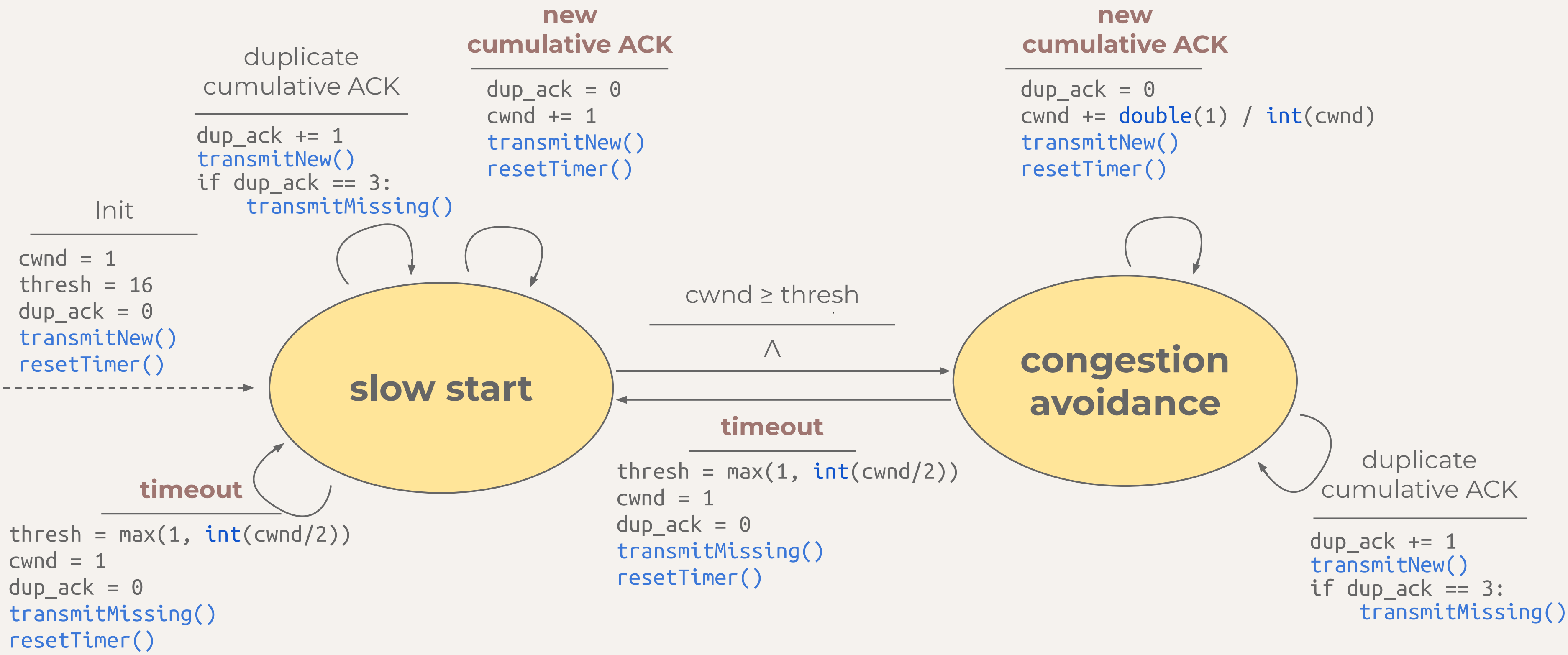
1/1 ACK + SACK

SACK + Congestion Control

- After receiving Data 4, the buffer is filled.
- Send back ACK + flush buffer + deliver data to application



SACK + Congestion Control



Init

```
cwnd = 1
thresh = 16
dup_ack = 0
transmitNew()
resetTimer()
```

duplicate cumulative ACK

```
dup_ack += 1
transmitNew()
if dup_ack == 3:
    transmitMissing()
```

new cumulative ACK

```
dup_ack = 0
cwnd += 1
transmitNew()
resetTimer()
```

timeout

```
thresh = max(1, int(cwnd/2))
cwnd = 1
dup_ack = 0
transmitMissing()
resetTimer()
```

cwnd ≥ thresh

∧

timeout

```
thresh = max(1, int(cwnd/2))
cwnd = 1
dup_ack = 0
transmitMissing()
resetTimer()
```

new cumulative ACK

```
dup_ack = 0
cwnd += double(1) / int(cwnd)
transmitNew()
resetTimer()
```

duplicate cumulative ACK

```
dup_ack += 1
transmitNew()
if dup_ack == 3:
    transmitMissing()
```

Assignment 3 Announcement

Docker

- We provide a docker config (docker-compose.yml) for you to run our example code. If you use Windows, you will need to install **Windows Subsystem Linux (WSL 2)** first.
- **Please make sure you can compile and run your code well in the provided docker container.**

Docker Installation

- Windows
  - Install Windows Subsystem Linux (WSL): Guide
  - Install Docker Desktop
- Ubuntu
  - Install Docker through your terminal:

    ```
    # apt update
    # apt install docker.io
    ```
- macOS
  - Install Docker Desktop

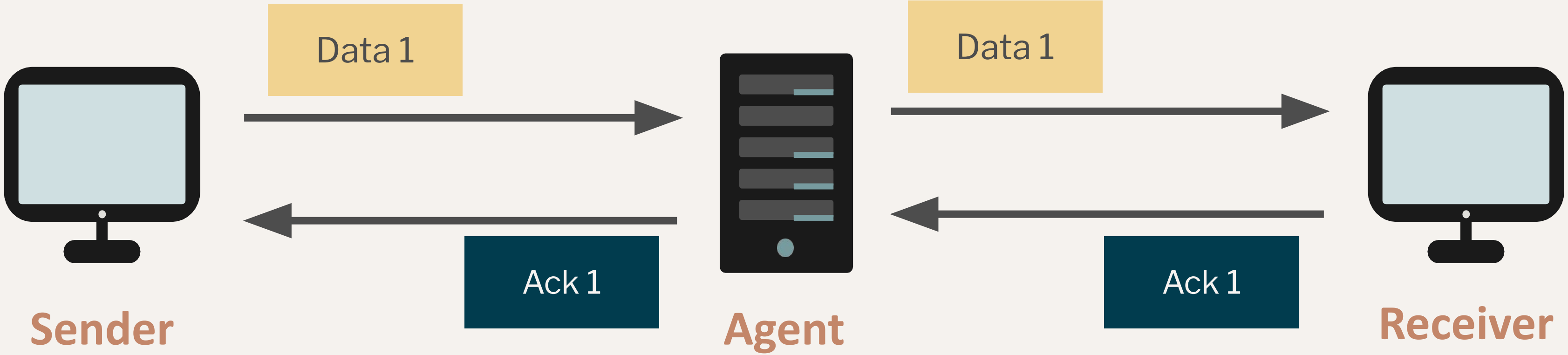
Start Your Container

- Clone repository to your host and run the container:

```
$ git clone <your_repository>
$ docker-compose up -d
$ docker exec -it <container_name> bash
```

Specification (1/16)

- Implement three components: sender, receiver and agent.



Specification (2/16)

- **Programming language: C/C++**
- **Sender / Receiver**
  - Send / receive **file content** by UDP
  - Receiver compute **SHA-256** hash of the file stream.
  - Provide reliable transmission
  - Congestion control
- **Agent**
  - Forward Data & ACK packets
  - **Randomly drop or corrupt data packet, not ACK nor FIN**
  - Compute error rate

Specification (3/16)

- **Reliable Transmission**
  - Data & ACK
  - Time out & Retransmission (**SACK protocol**)
  - **Sequence number**
- **Buffer handling** [**receiver side**]
  - Buffer Overflow: **Drop the packet** if the packet is **out of buffer**
  - Flush (write) to the file: Only when **buffer is full** or **all segments for the file are received**.

Specification (4/16)

- **Congestion Control** [**sender side**]
    - Slow Start
        1. Send single packet in the beginning
        2. When window size is under the threshold, it increases **exponentially** until packet loses
        3. When window size is equal to or over the threshold, it increases **linearly** until packet loses
    - Packet loss / Time out
        4. Set **threshold** to `max(1, int(cwnd / 2))`
        5. Set **window size** to 1
        6. Retransmit – for the first “unACKed packet” (`transmitMissing()`)

Specification (5/16)

- **Congestion Control** [**sender side**]
  - Duplicate cumulative ACK
    1. Detect when **3** duplicate cumulative ACK has happened.
    2. Retransmit the first “unACKed packet” (`transmitMissing()`)

Specification (6/16)

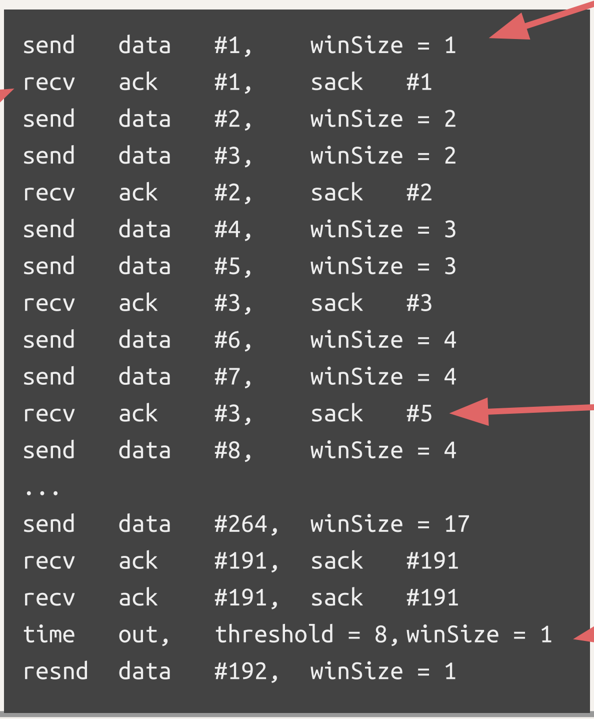
- **Show Message:** Log to stdout
  - Sender:
    - `send, recv, data, ack, fin, finack, sequence number, ack number, sack number, time out, resnd, winSize, threshold`
  - Receiver:
    - `send, recv, data, ack, fin, finack, sequence number, ack number, sack number, drop (corrupted/buffer overflow), flush, sha256, finsha`
  - Agent:
    - `get, fwd, data, ack, fin, finack, sequence number, ack number, sack number, drop, corrupt, error rate`

Specification (7/16)

- Show Message for **sender**

Uses tab: `send\tdata\t#%d,\twinSize = %d`

ACK segment have cumulative & selective ACK number



```
send   data   #1,    winSize = 1
recv   ack    #1,    sack    #1
send   data   #2,    winSize = 2
send   data   #3,    winSize = 2
recv   ack    #2,    sack    #2
send   data   #4,    winSize = 3
send   data   #5,    winSize = 3
recv   ack    #3,    sack    #3
send   data   #6,    winSize = 4
send   data   #7,    winSize = 4
recv   ack    #3,    sack    #5
send   data   #8,    winSize = 4
...
send   data   #264,  winSize = 17
recv   ack    #191,  sack    #191
recv   ack    #191,  sack    #191
time   out,   threshold = 8, winSize = 1
resnd  data   #192,  winSize = 1
```

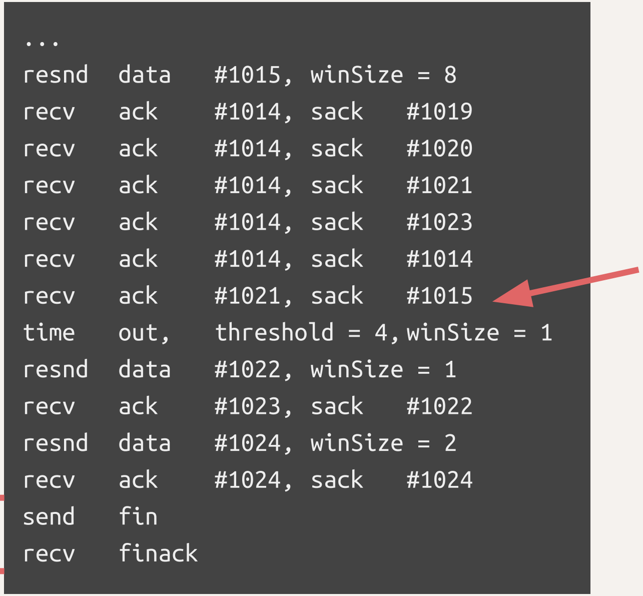
Out of order SACK-ed #5
Data#8 gets into window, send!

Might be "`send`"
or "`resnd`"

Timeout!
`TransmitMissing()`

Specification (8/16)

- Show Message for **sender**



```
...
resnd  data   #1015, winSize = 8
recv   ack    #1014, sack   #1019
recv   ack    #1014, sack   #1020
recv   ack    #1014, sack   #1021
recv   ack    #1014, sack   #1023
recv   ack    #1014, sack   #1014
recv   ack    #1021, sack   #1015
time   out,   threshold = 4, winSize = 1
resnd  data   #1022, winSize = 1
recv   ack    #1023, sack   #1022
resnd  data   #1024, winSize = 2
recv   ack    #1024, sack   #1024
send   fin
recv   finack
```

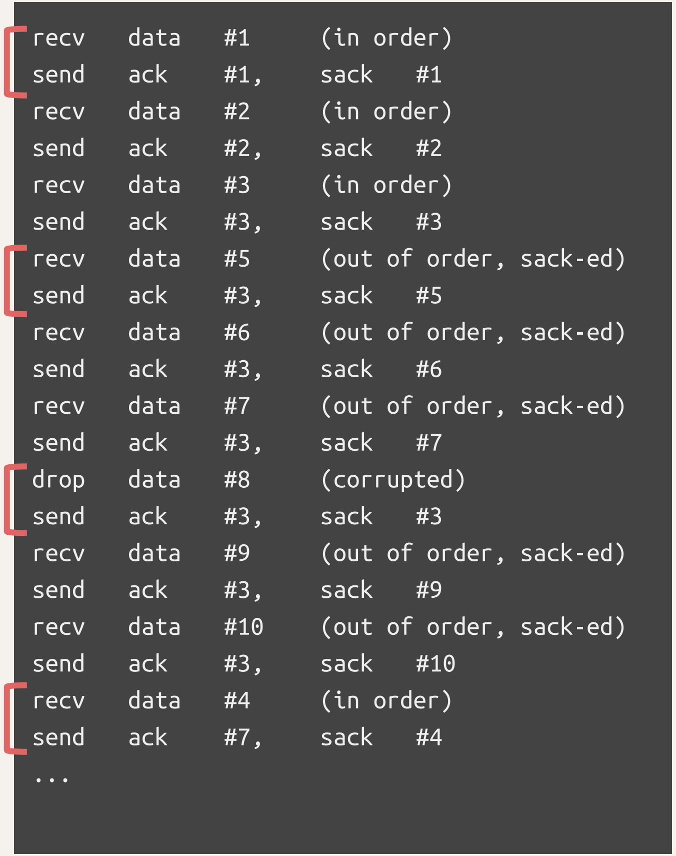
Every data segments
has been ACKed!

New ACK!
But no new segments gets
into the window...

Finish the stream
(agent will never drop those 2)
(there may be more ACK in between)

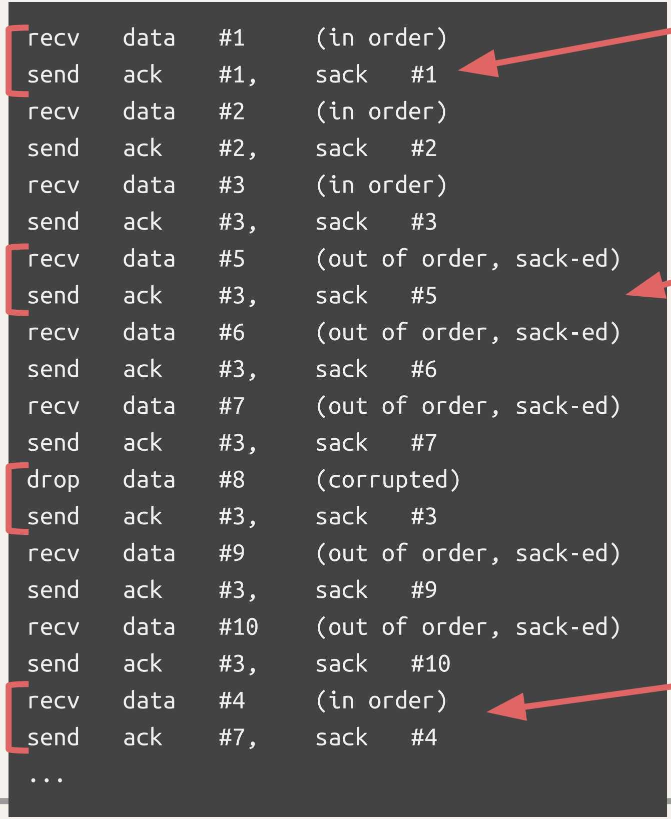
Specification (9/16)

- Show Message for **receiver**
- Check a packet in the following order:
  a. **corrupt** or not
  b. **in / under / above buffer**



```
recv   data   #1     (in order)
send   ack    #1,    sack   #1
recv   data   #2     (in order)
send   ack    #2,    sack   #2
recv   data   #3     (in order)
send   ack    #3,    sack   #3
recv   data   #5     (out of order, sack-ed)
send   ack    #3,    sack   #5
recv   data   #6     (out of order, sack-ed)
send   ack    #3,    sack   #6
recv   data   #7     (out of order, sack-ed)
send   ack    #3,    sack   #7
drop   data   #8     (corrupted)
send   ack    #3,    sack   #3
recv   data   #9     (out of order, sack-ed)
send   ack    #3,    sack   #9
recv   data   #10    (out of order, sack-ed)
send   ack    #3,    sack   #10
recv   data   #4     (in order)
send   ack    #7,    sack   #4
...
```

Specification (10/16)



```
recv data #1   (in order)
send ack  #1,  sack #1
recv data #2   (in order)
send ack  #2,  sack #2
recv data #3   (in order)
send ack  #3,  sack #3
recv data #5   (out of order, sack-ed)
send ack  #3,  sack #5
recv data #6   (out of order, sack-ed)
send ack  #3,  sack #6
recv data #7   (out of order, sack-ed)
send ack  #3,  sack #7
drop data #8   (corrupted)
send ack  #3,  sack #3
recv data #9   (out of order, sack-ed)
send ack  #3,  sack #9
recv data #10  (out of order, sack-ed)
send ack  #3,  sack #10
recv data #4   (in order)
send ack  #7,  sack #4
...
```

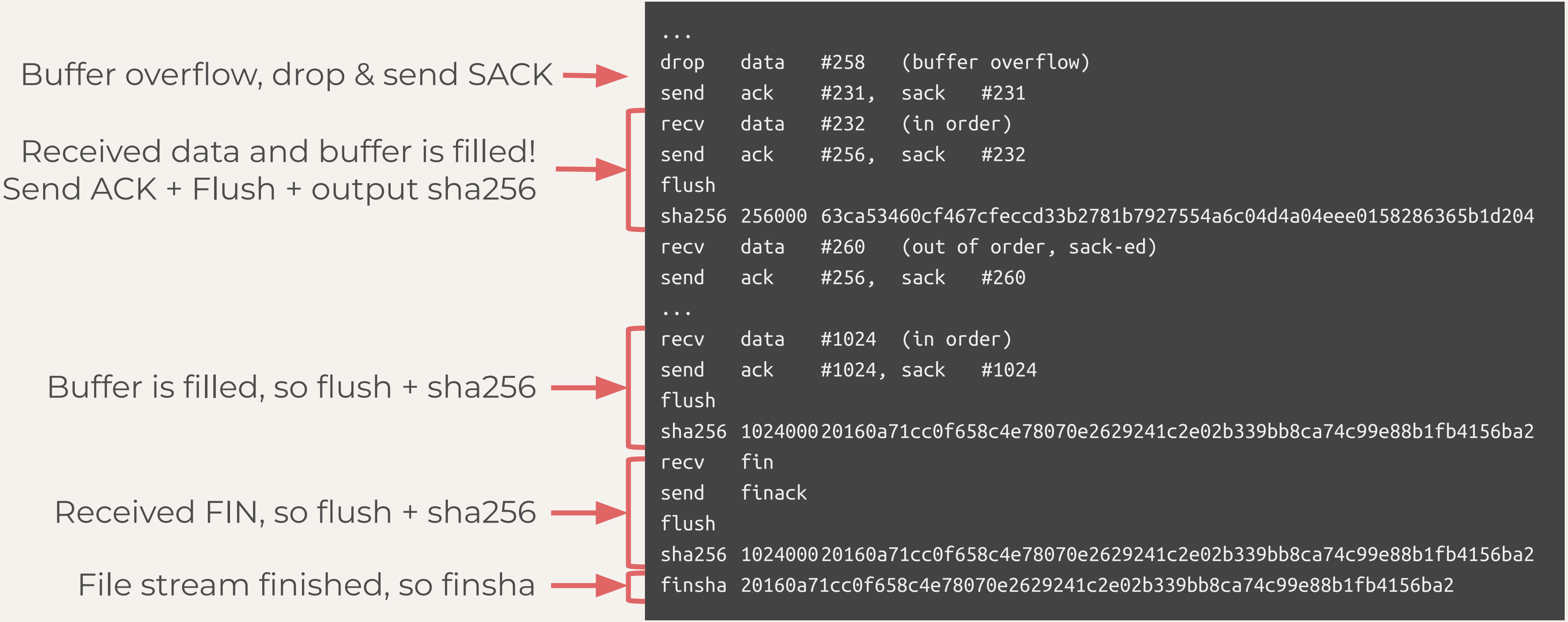
In order! Update base!

Out of order but in buffer range.
Store in buffer and SACK.

Corrupt but still sends SACK
(but `sackNumber = ackNumber`)

In order! Update base!
(we haven't SACKed Data#8)

Specification (11/16)



Buffer overflow, drop & send SACK

Received data and buffer is filled!
Send ACK + Flush + output sha256

Buffer is filled, so flush + sha256

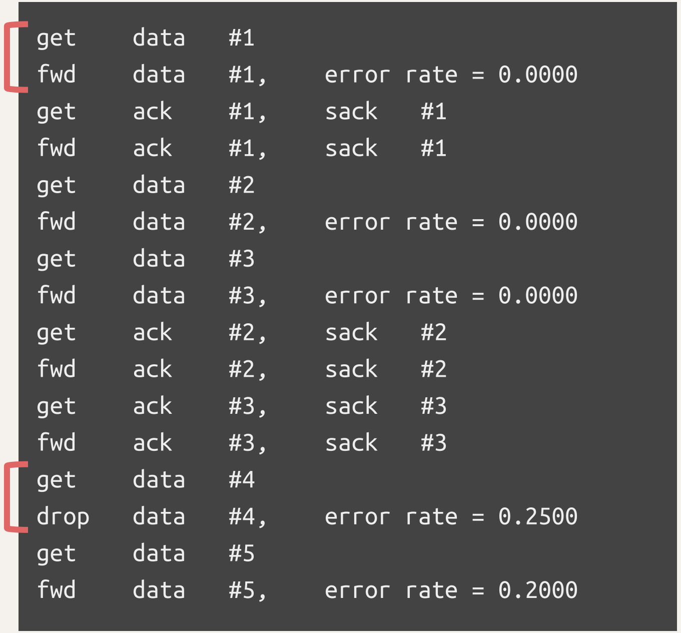
Received FIN, so flush + sha256

File stream finished, so finsha

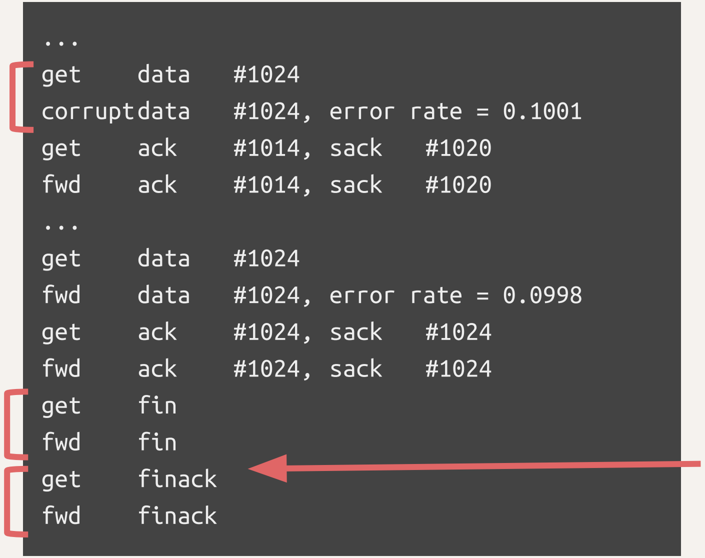
```
...
drop   data   #258   (buffer overflow)
send   ack    #231,  sack   #231
recv   data   #232   (in order)
send   ack    #256,  sack   #232
flush
sha256 256000 63ca53460cf467cfeccd33b2781b7927554a6c04d4a04eee0158286365b1d204
recv   data   #260   (out of order, sack-ed)
send   ack    #256,  sack   #260
...
recv   data   #1024  (in order)
send   ack    #1024, sack   #1024
flush
sha256 1024000 20160a71cc0f658c4e78070e2629241c2e02b339bb8ca74c99e88b1fb4156ba2
recv   fin
send   finack
flush
sha256 1024000 20160a71cc0f658c4e78070e2629241c2e02b339bb8ca74c99e88b1fb4156ba2
finsha 20160a71cc0f658c4e78070e2629241c2e02b339bb8ca74c99e88b1fb4156ba2
```

Specification (12/16)

- Show Message for **agent**



```
get    data   #1
fwd    data   #1,    error rate = 0.0000
get    ack    #1,    sack   #1
fwd    ack    #1,    sack   #1
get    data   #2
fwd    data   #2,    error rate = 0.0000
get    data   #3
fwd    data   #3,    error rate = 0.0000
get    ack    #2,    sack   #2
fwd    ack    #2,    sack   #2
get    ack    #3,    sack   #3
fwd    ack    #3,    sack   #3
get    data   #4
drop   data   #4,    error rate = 0.2500
get    data   #5
fwd    data   #5,    error rate = 0.2000
```



```
...
get    data   #1024
corruptdata   #1024, error rate = 0.1001
get    ack    #1014, sack   #1020
fwd    ack    #1014, sack   #1020
...
get    data   #1024
fwd    data   #1024, error rate = 0.0998
get    ack    #1024, sack   #1024
fwd    ack    #1024, sack   #1024
get    fin
fwd    fin
get    finack
fwd    finack
```

There may be more get & fwd in between FIN & FINACK

Specification (13/16)

- **Packet structure**
    - The format used for transmission should be the same as the right side (defined in `def.h`):
    - `fin`: 0 or 1
    - `syn`: 0 or 1 (just make it 0)
    - `ack`: 0 or 1
    - `checksum`: we will use `crc32()` in `zlib.h` to calculate checksum

```
struct header {
    int length;
    int seqNumber;
    int ackNumber;
    int sackNumber;
    int fin;
    int syn;
    int ack;
    unsigned long checksum;
};

struct segment {
    struct header head;
    char data[MAX_SEG_SIZE];
};
```

Specification (14/16)

- **Settings**
  - Sender
    - Default threshold: 16
    - Default window size: 1
  - Receiver
    - Default packet data size (payload): `MAX_SEG_SIZE (1000)` bytes
    - Default buffer size: `MAX_SEG_BUF_SIZE (256)` (# of packets)
  - Agent
    - Default packet data size (payload): `MAX_SEG_SIZE (1000)` bytes
  - Default time out: `TIMEOUT_MILLISECONDS (1000)` milliseconds.

Specification (15/16)

- You are required to write a Makefile for compilation.

```
$ make sender            // To compile sender code
$ make agent             // To compile agent code
$ make receiver          // To compile receiver code
$ make                   // To run the above 3 compilation
```

- After compilation, there will be 3 binary files named “`sender`,” “`agent`,” and “`receiver`.”

Specification (16/16)

- Execute the following commands in different terminals and **in sequence**.

```
$ ./agent <agent_port> <send_ip> <send_port> <recv_ip> <recv_port> <error_rate>
$ ./receiver <recv_ip> <recv_port> <agent_ip> <agent_port> <dst_filepath>
$ ./sender <send_ip> <send_port> <agent_ip> <agent_port> <src_filepath>
```

- The error rate will be a floating point number between 0 and 1.
- Only legal `IP` and `port` will be used in this assignment.
- `src_filepath` may be `/dev/stdin`, and `dst_filepath` may be `/dev/null`.
- The input file is a binary file, i.e., it can have null bytes.

Grading Policy (1/2)

- This assignment accounts for 15% of the total score.
- **File Transmission (20%)**
  - Receives and stores the file correctly
- **Buffer handling (10%)**
  - Flush + drop buffer overflow
- **Reliable transmission (20%)**
  - SACK protocol
- **SHA-256 Hash (10%)**
  - finsha & sha256

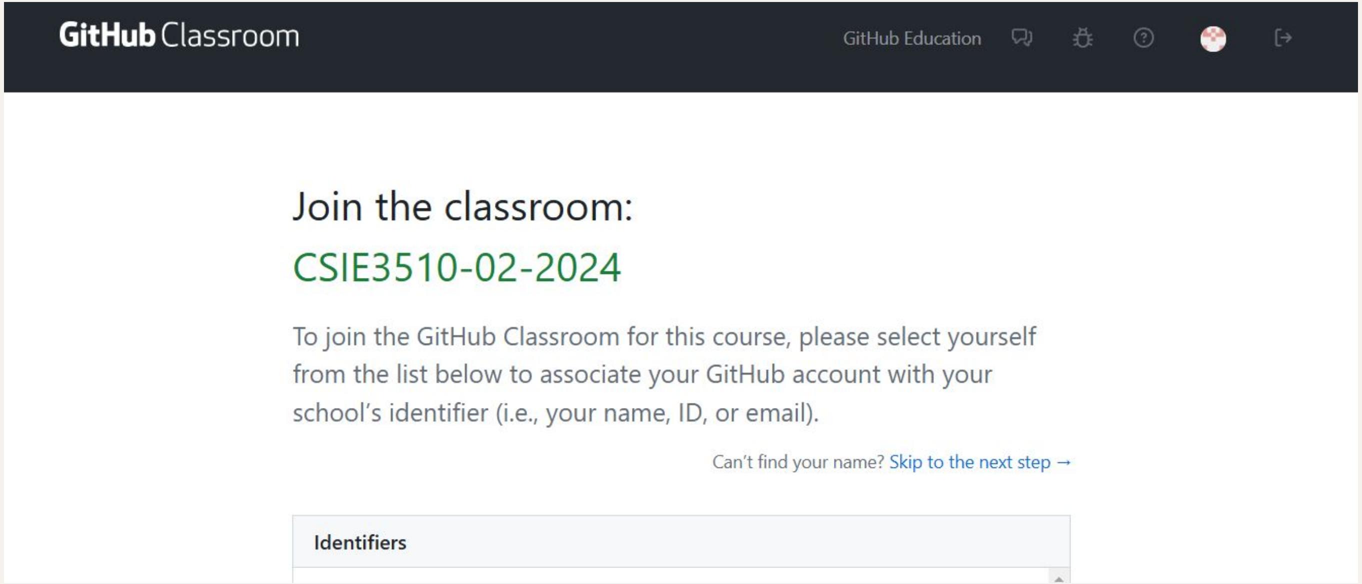
Grading Policy (2/2)

- **Congestion control (15%)**
  - Window size & threshold
- **Show Message (10%)**
  - Show message correctly
- **Report (15%)**
  - Explain your program structure (5% * 3)

    (including **3 flow charts** for **sender**, **agent**, and **receiver**)

Github Classroom

- Get the access of assignment materials via **Github Classroom**.



Join the classroom:

CSIE3510-02-2024

To join the GitHub Classroom for this course, please select yourself from the list below to associate your GitHub account with your school’s identifier (i.e., your name, ID, or email).

Can’t find your name? Skip to the next step →

Identifiers

Submission

- Report
  - Your report should be a **pdf** file. Submit it to **Gradescope**.
  - PDF file name: <studentID>_hw3.pdf
    - e.g., B11902999_hw3.pdf
- Codes
  - Please push all the **source code** (i.e., without your report, and the execution file) to **Github classroom** assigment.
- The penalty for the wrong format is **10 points**.
- **No plagiarism is allowed. A plagiarist will be graded zero.**

Submission

- Deadline
    - Due Date : 13:00, December 11st, 2024 (temporary)
    - Penalty for late submission is **20 points per day**.

Sample Codes

- We will provide sample codes for your reference
  - *sender.cpp, receiver.cpp, agent.cpp*
  - *crc32.cpp, sha256.cpp*
  - *def.h*
  - *Makefile*

Supplementary Materials

crc32()

- Compute CRC-32 Checksum. (ref. `crc32.cpp`)

```
#include <zlib.h>
unsigned long crc32(unsigned long crc, const Bytef * buf, unsigned int len);
```

- **`crc`**
  - The previous value for the checksum.
  - In this homework, we can set it to `0L`.
- **`buf`**: Specifies the buffer to contain the data to be added to this checksum.
- **`len`**: Specifies the size of `buf`.

SHA-256

- Compute SHA-256 hash. (ref. `sha256.cpp`)

```
#include <openssl/evp.h>
EVP_MD_CTX *EVP_MD_CTX_new(void);
int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl);
int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);
int EVP_MD_CTX_copy_ex(EVP_MD_CTX *out, const EVP_MD_CTX *in);
int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s);
void EVP_MD_CTX_free(EVP_MD_CTX *ctx);
```

Contact us if you have any problem. •ω•)ฅ

TA Email: ntu.cnta@gmail.com