CSCI4430 Tutorial-4
Assignment 2 Review(part 1)

(slides adapted from csci4430@2018spring by Xiaolu Li)

zncheng@cse.cuhk.edu.hk

February 21, 2019

Content

- Our goal
- Course VM access
- Application Overview
- Lossy Network Overview
- Selective Repeat Overview

Our goal

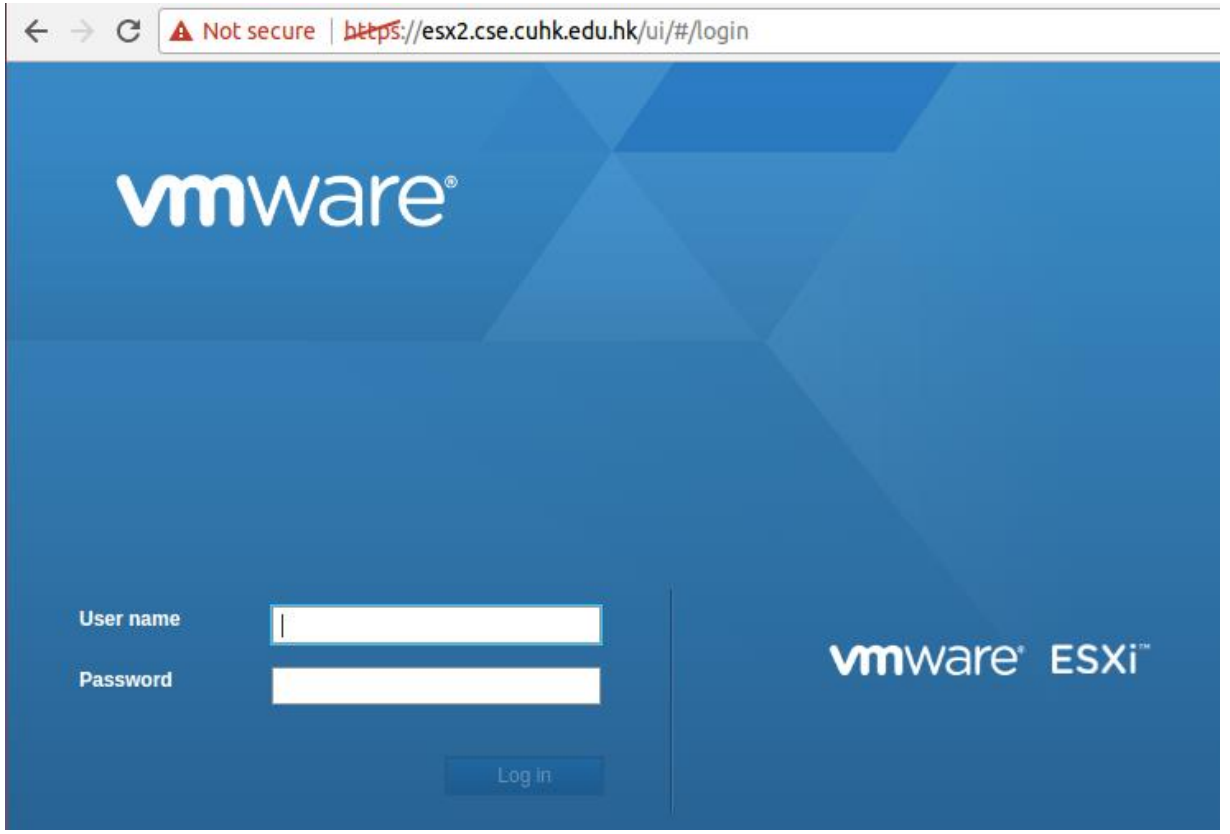
- In assignment 2, we are going to implement Selective Repeat protocol to support the reliable data transfer between a client and a server over UDP network connection.
- Before we start to implement the Select Repeat protocol, we may first want to know:
  - What is the application data that is going to be transferred?
    - run the given application code in tcp version in our course VM
  - How lossy network works?
    - create lossy network
    - example code using UDP connection

Course VM access

- Thanks to our tutor Mi! She sent us email about our course VMs at Jan 28, 2018. Please check your mail box!
- Our course VMs are only accessible within the CSE network.
  - CSE Lab
  - CSE VPN

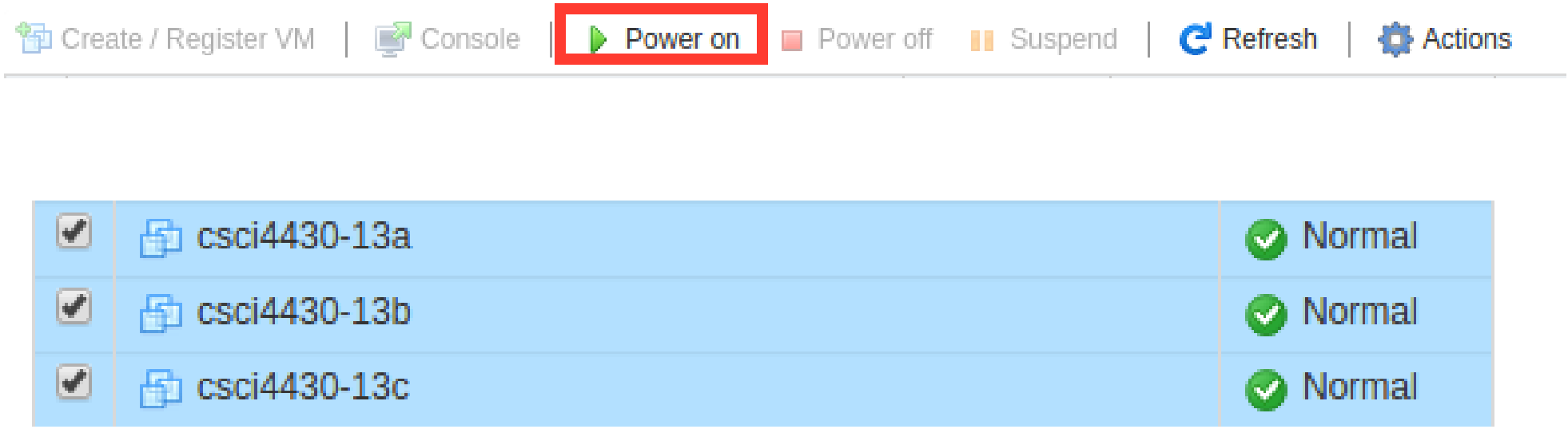
Course VM access

- web interface:
  - https://esx2.cse.cuhk.edu.hk/ui
  - Different groups have different links to login in, find the links for your group in the email
- username: cse unix username
- passowrd: cse unix password



Course VM access

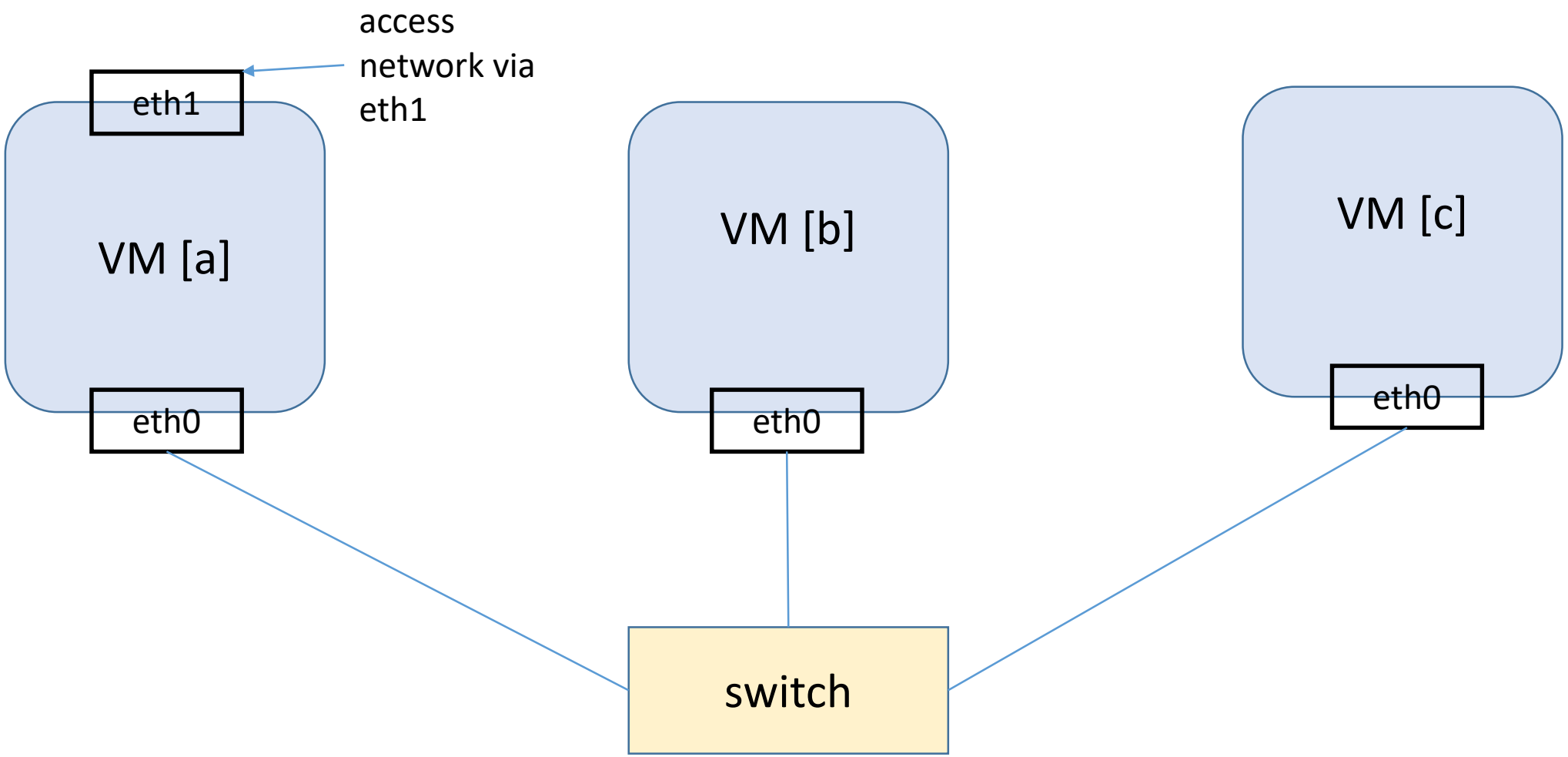
- power on our VMs



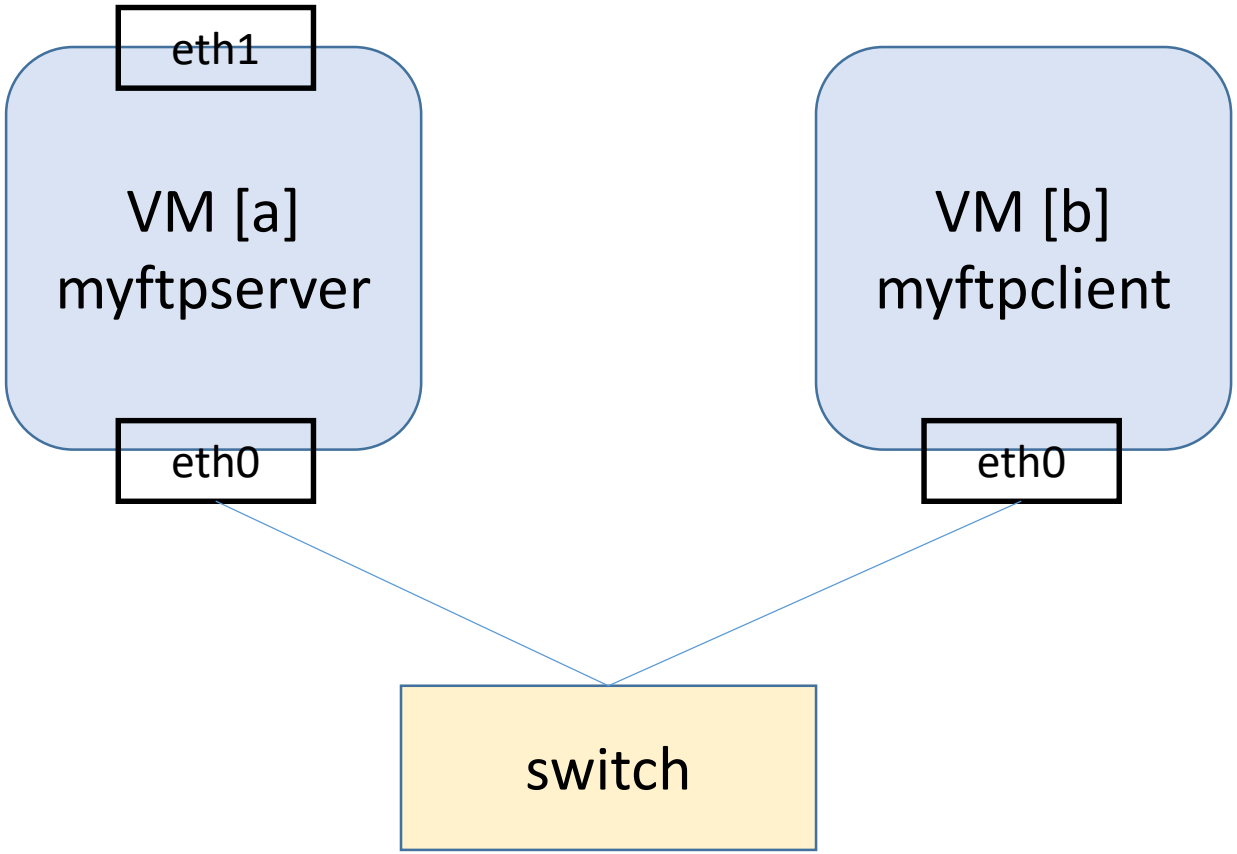
Course VM access

- After we power on our VMs
  - log in with the default username and password: csci4430
  - change the default password as you like
- Log in
  - First check your redirected port via the email you received previously
  - > ssh –p $port csci4430@projgw.cse.cuhk.edu.hk
- Send file to your VM:
  - > scp –P $port filename csci4430@projgw.cse.cuhk.edu.hk:~
  - > scp –P $port –r dirname csci4430@projgw.cse.cuhk.edu.hk:~
- Copy file from VM to linux1-16
  - > scp filename $cseunixusername@linux1:~

Course VM architecture:



Application Overview:



Application Overview: tcp version

- Download the tcp version application from course website
- upload the file to our vm[a]
  - > scp –P $port myftp_tcp.tar.gz csci4430@projgw.cse.cuhk.edu.hk:~
- copy it to vm[b]
  - > scp myftp_tcp.tar.gz $vmb:~
  - (vmb is the ip or hostname of vm[b])

Application Overview: tcp version

- On vm[a]:
- extract tar files
  - > tar –zxvf myftp_tcp.tar.gz
  - > cd myftp_tcp
- create data directory
  - > mkdir data
- Compile
  - > make
- Start myftpserver on port 12345
  - > ./myftpserver 12345

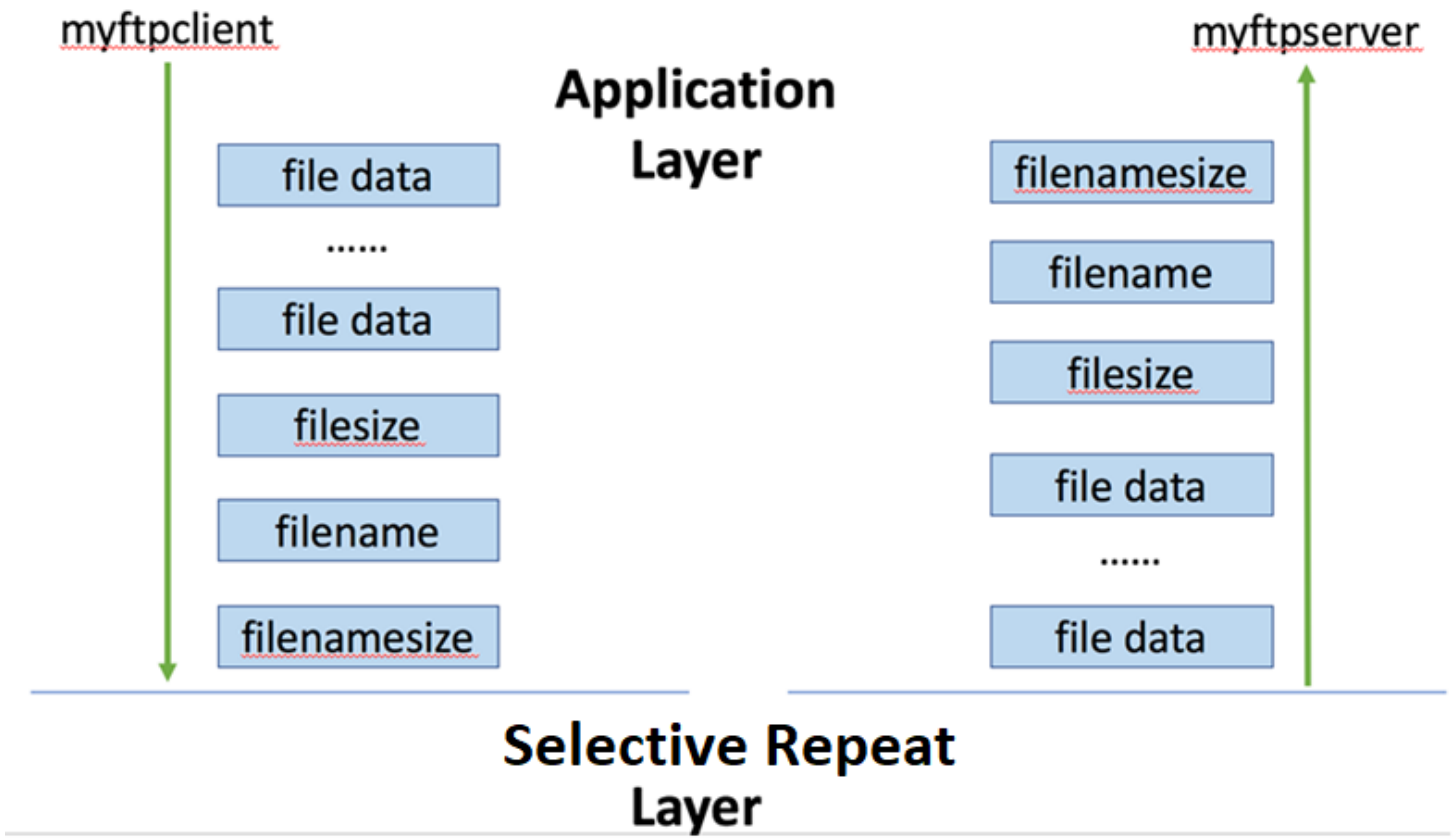
Application Overview: tcp version

- On vm[b]:
- extract tar files
  - > tar –zxvf myftp_tcp.tar.gz
  - > cd myftp_tcp
- compile
  - > make
- create a file called “testfile”
  - please create one by yourself
- run myftpclient to upload this file
  - > ./myftpclient $serverip 12345 testfile

Application Overview: tcp version

- After the client finishes uploading the file, we can check the correctness of the uploaded file under the “data” directory in vm[a].

Application Details:



Lossy Network: troller

- We provide an executable file “troller”
  - It runs in the server side.
  - It can drop or reorder UDP packets with some probabilities.
- How troller works?
  - First we create a NFQUEUE.
  - Next we add some routing rules which we are interested in.
  - The packets which match the rule will go to our NFQUEUE.
  - For each packet in this NFQUEUE, troller decides whether it will be delivered.

Lossy Network: troller

- Install required library:
  - > sudo -E apt-get update
  - > sudo -E apt-get install libnetfilter-queue-dev
- Setup NFQUEUE and iptables rules:
  - > sudo iptables -t filter -F
  - > sudo iptables -A INPUT -p udp -s $clientip -d $serverip -j NFQUEUE --queue-num 0
  - > sudo iptables -A OUTPUT -p udp -s $serverip -d $clientip -j NFQUEUE --queue-num 0
- It is suggested that you create a script to set NFQUEUE and iptables rules. (e.g. setup.sh)

Lossy Network: troller

- To simulate a lossy network:
  - > sudo troller <drop_ratio> <reorder_ratio>
- To clean the lossy network:
  - method1: > sudo iptables -t filter -F
  - method2: > sudo ./troller 0 0
    - This means that the probability to drop a packet is 0 and the probability to reorder a packet is 0

Lossy Network: lossy_sample.tar.gz

- We provide another sample code for us to get an overview of how lossy network works.
- Please download lossy_sample.tar.gz from tutorial website.
- Compile the sample code on vm[a] and vm[b].
- In this sample code, client sends integer 0, 1, 2, …… to server. And server receives an integer in a while loop.

lossy_sample: client.c

- create UDP socket
- initialize address of server

```
// create an IPv4/UDP socket
int fd = socket(AF_INET, SOCK_DGRAM, 0);

// initialize the address of server
struct sockaddr_in destination;
memset(&destination, 0, sizeof(struct sockaddr_in));
destination.sin_family = AF_INET;
inet_pton(AF_INET, ip, &(destination.sin_addr));
destination.sin_port = htons(12345);
```

lossy_sample: client.c

- send an integer

```
int num=0;
while (1) {
  int tmpnum=htonl(num);
  int len = sizeof(tmpnum);
  sendto(fd, (char*)&tmpnum, len, 0, (struct sockaddr *)&destination, sizeof(destination));
  num++;
  sleep(1);
}
```

lossy_sample: server.c

- initialize

```
// create a IPv4/UDP socket
int fd = socket(AF_INET, SOCK_DGRAM, 0);

// initialize the address
struct sockaddr_in address;
memset(&address, 0, sizeof(address));
address.sin_family = AF_INET;
address.sin_port = htons(12345);
address.sin_addr.s_addr = htonl(INADDR_ANY);

// Bind the socket to the address
bind(fd, (struct sockaddr *)&address, sizeof(struct sockaddr));
```

lossy_sample: server.c

- receive integers

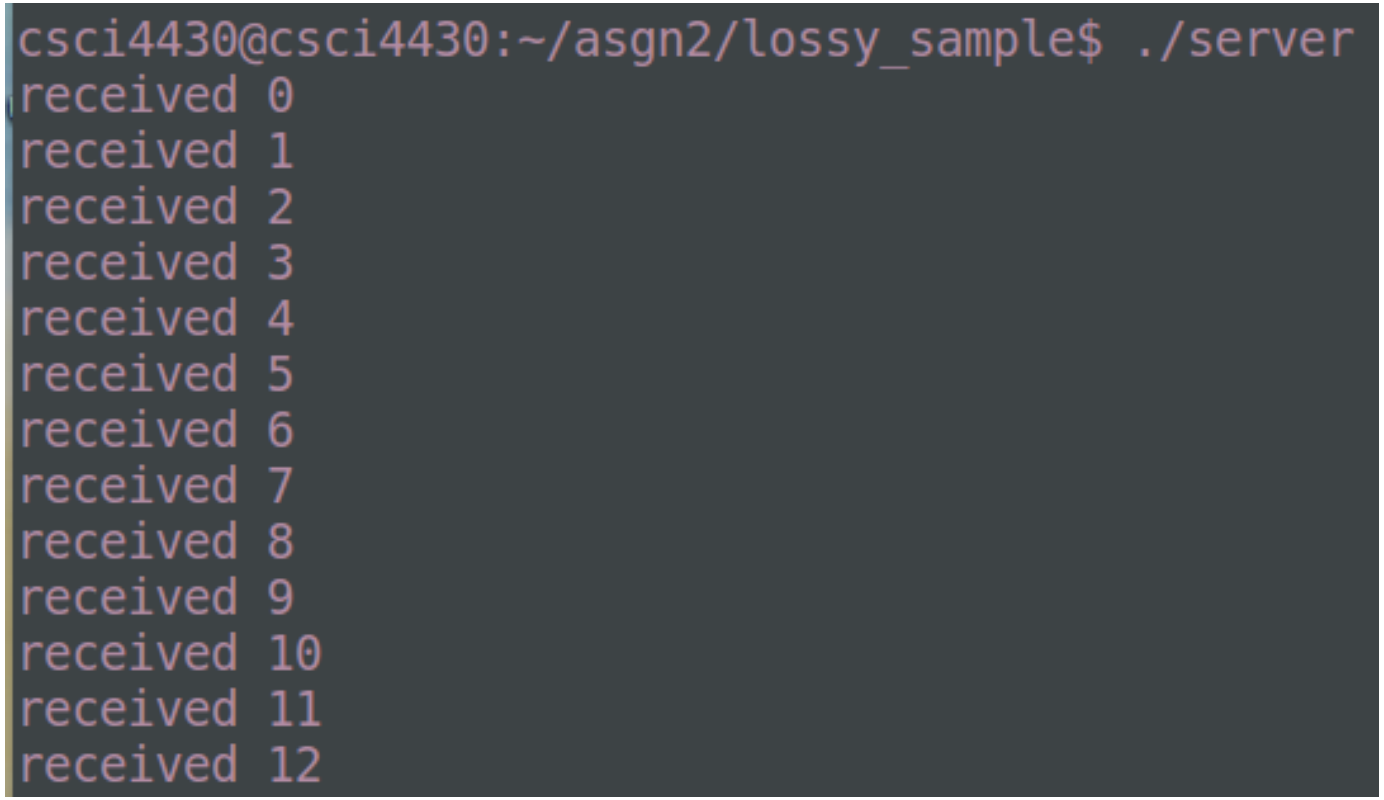
```
while (1) {
  // Receive the message (we don't care about the address of the sender)
  int tmpnum;
  int len = recvfrom(fd, (char*)&tmpnum, sizeof(tmpnum), 0, NULL, NULL) ;
  int num = ntohl(tmpnum);
  printf("received %d\n", num);
}
```

lossy network: drop_ratio=0, reorder_ratio=0

- Before we start our sample code
  - method1: clean the iptables rules on vm[a]
    - > sudo iptables -t filter -F
  - method2: create another terminal on vm[a] and run troller
    - > sudo ./troller 0 0
- You can use one of the methods. Either is OK.

lossy network: drop_ratio=0, reorder_ratio=0

- First start server on vm[a]
  - > ./server
- Then start client on vm[b]
  - > ./client $serverip
- We can see that the server receives all the packet in order



lossy network: drop_ratio=0, reorder_ratio=1

- First start server on vm[a]
  - > ./server
- Then start client on vm[b]
  - > ./client $serverip
- We can see that the server receives all the packet but out of order

```
csci4430@csci4430:~/asgn2/lossy_sample$ ./server
received 1
received 0
received 3
received 4
received 5
received 6
received 7
received 8
received 2
received 10
received 11
received 9
received 13
received 12
```

lossy network: drop_ratio=0.5, reorder_ratio=0

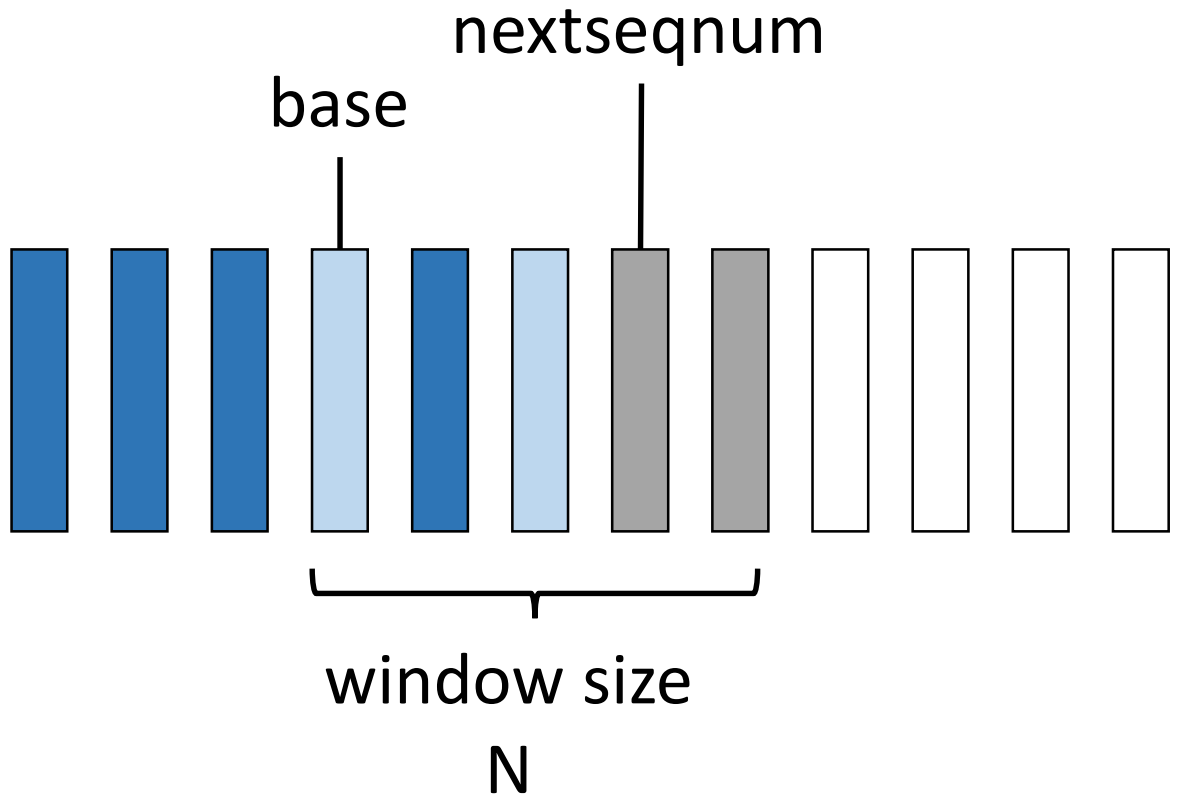
- First create a terminal on vm[a] and run
  - > sudo ./setup.sh
  - > sudo ./troller 0.5 0
- Second start server
- Third start client
- We can see that some packets are lost.

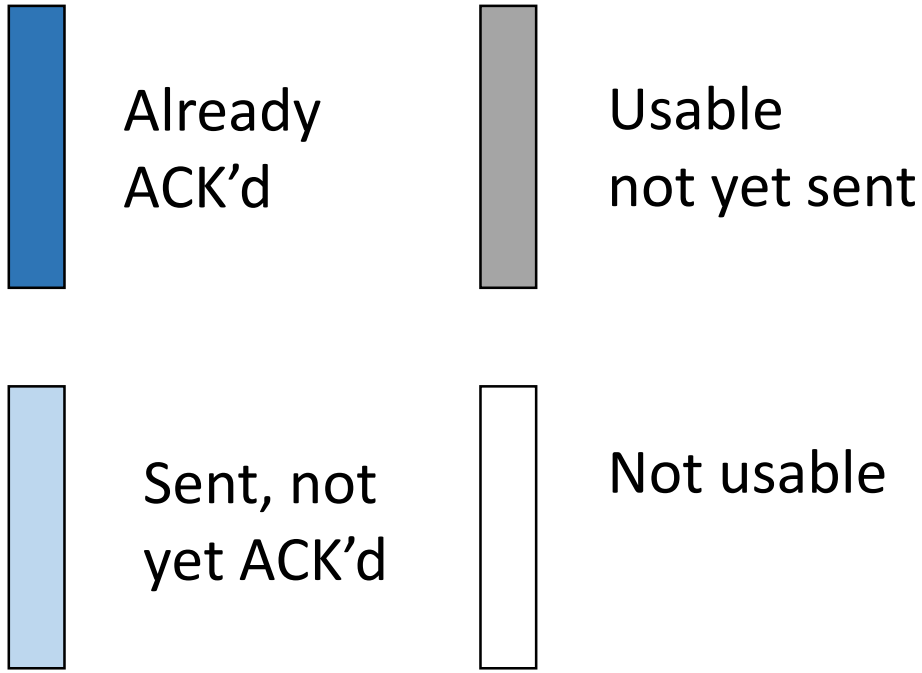
```
csci4430@csci4430:~/asgn2/lossy_sample$ ./server
received 1
received 2
received 4
received 7
received 9
received 10
received 11
received 14
received 16
received 22
received 23
received 28
received 29
```

lossy network

- In application overview, we know that client will send data to server in the following manner:
  - filename size
  - filename
  - file size
  - file data
- Using UDP network connection and our ”troller”, any packets can be lost.
- This is why we need to implement Selective Repeat protocol to support the reliable data transfer!

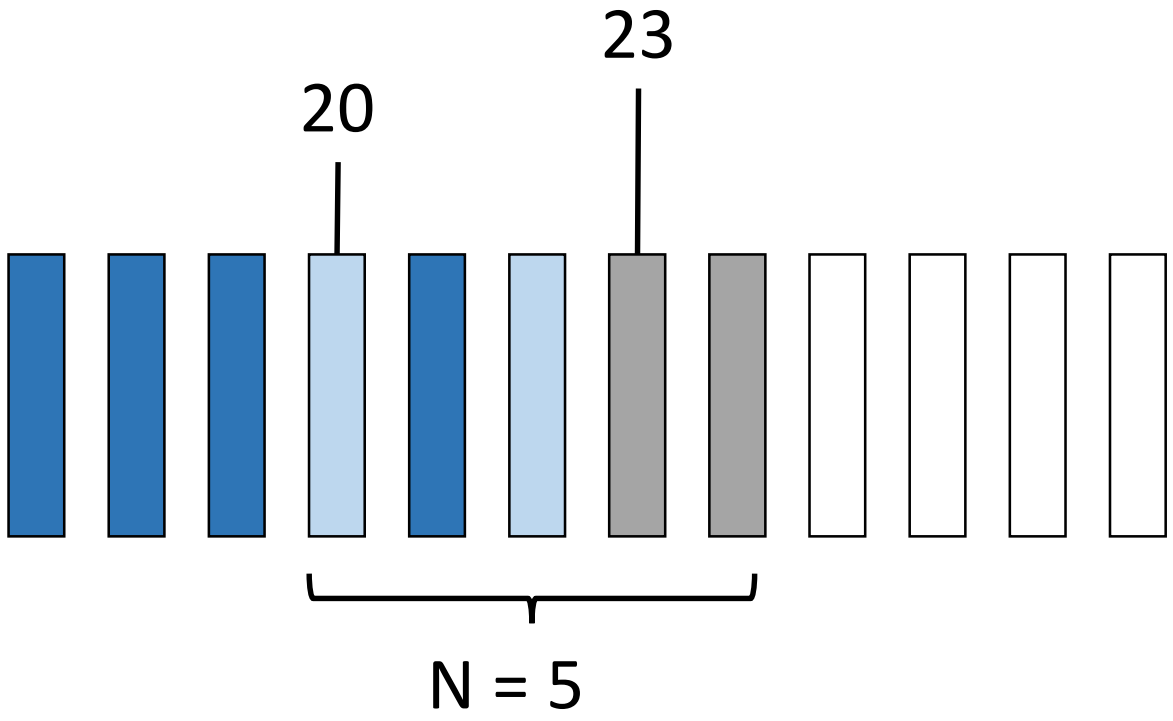
Selective Repeat Overview: Sender





- base: the seqnum of the oldest un-ACK'd packet
- nextseqnum: smallest unused seqnum

Selective Repeat Overview: Sender

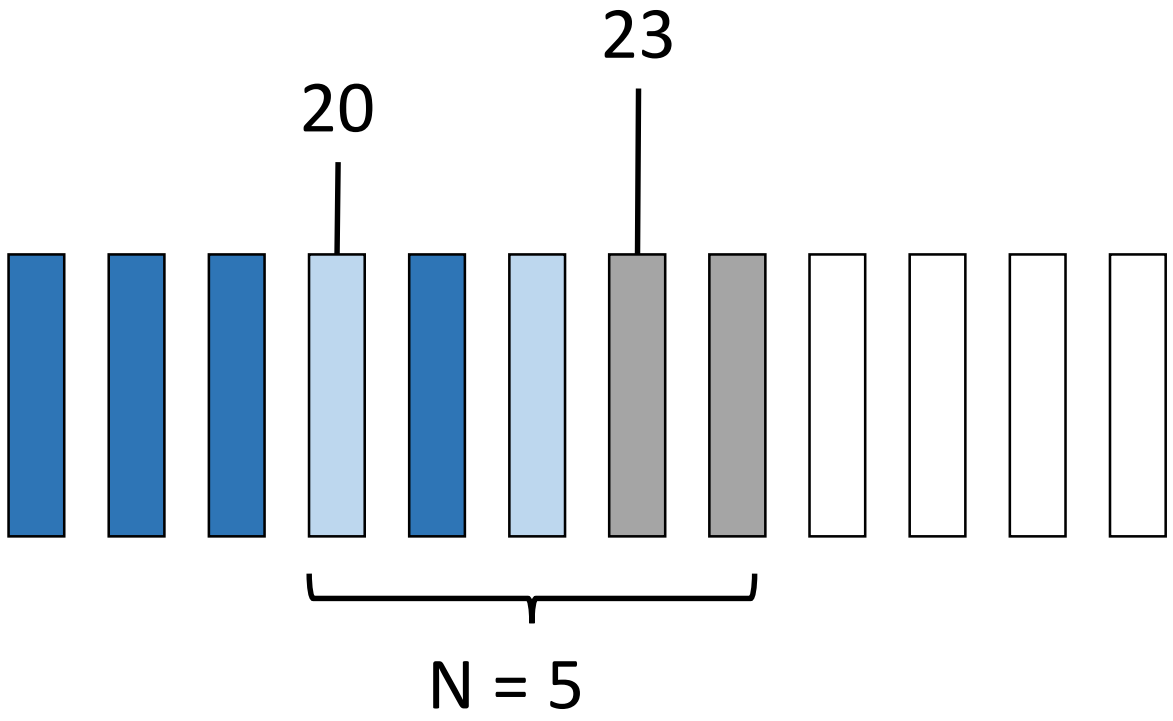


- Let’s suppose at this moment:
  - base = 20
  - nextseqnum = 23
  - N = 5
- Information:
  - Sender has received ACK of [1, 19] and 21
  - Sender has sent out 20, 22 but not ACK’d
  - Timers for 20 and 22
  - Window is not full

Selective Repeat Overview: Sender

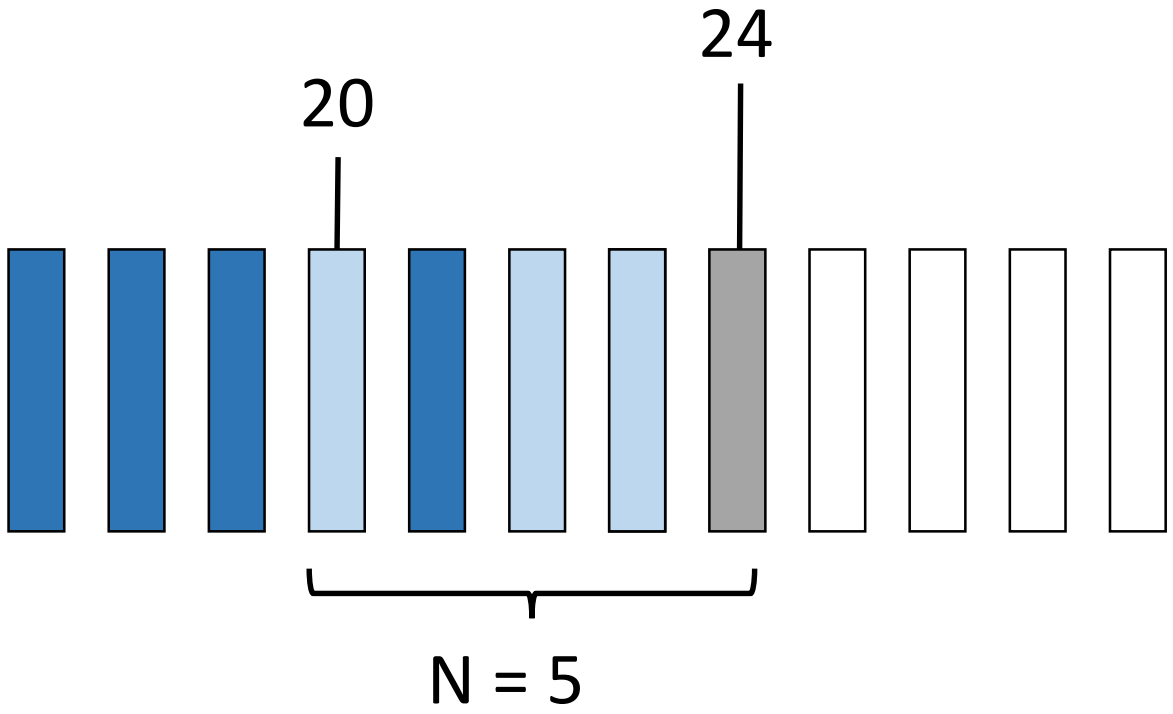
- Basically, sender needs to deal with 3 types of events:
  - Invocation from above
  - Receipt of an ACK
  - A timeout event

Sender: Invocation from above



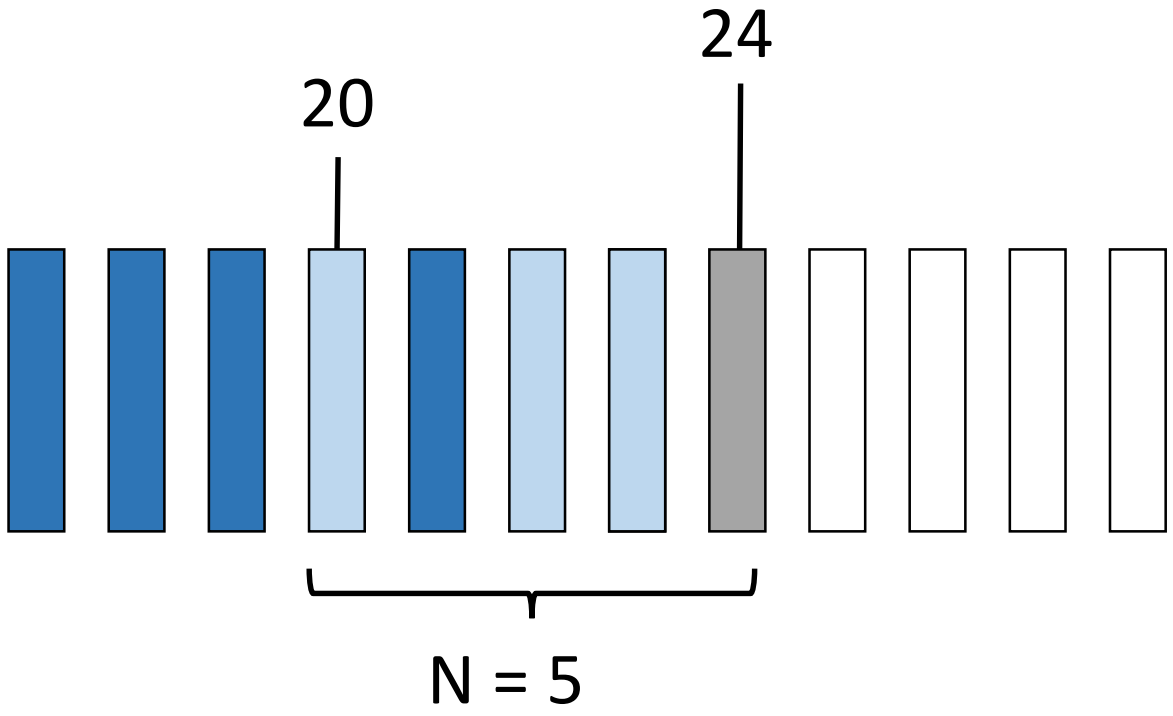
- Suppose
  - base = 20
  - nextseqnum = 23
  - N = 5
- If new data comes and request to be sent, what will happen?
- There is still vacancy in current window. So the packet will be created and sent immediately!
- Start a Timer for it.

Sender: Invocation from above



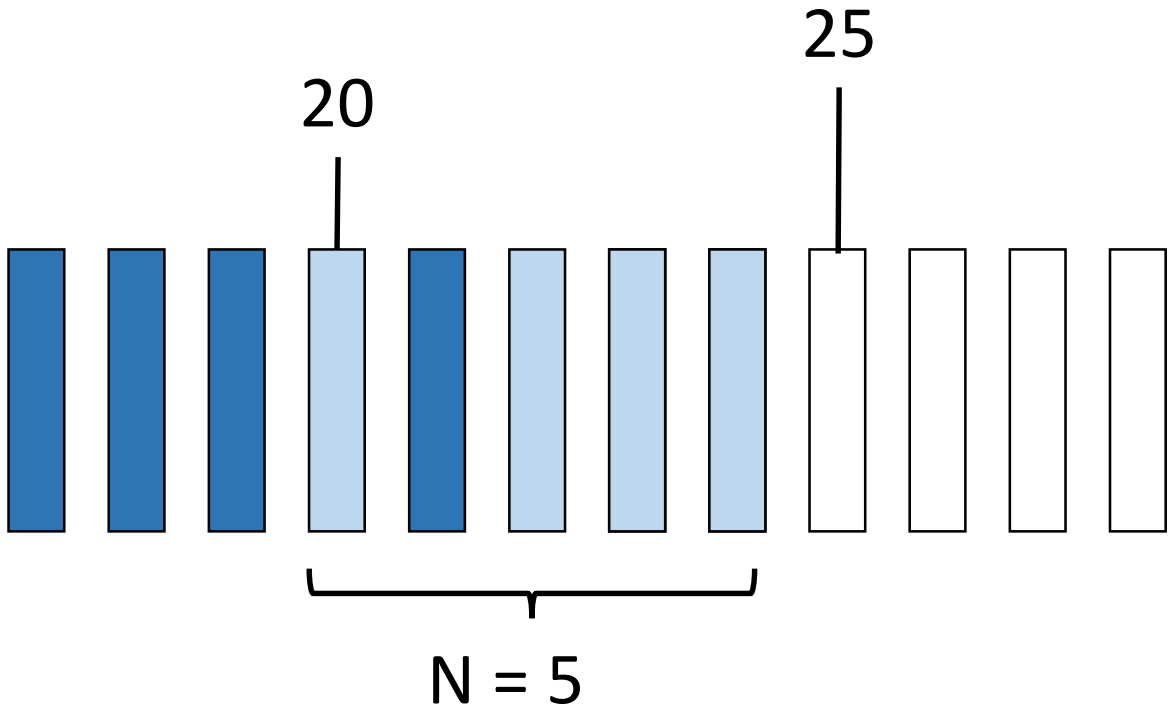
- Now
  - base = 20
  - nextseqnum = 24
  - N = 5

Sender: Invocation from above



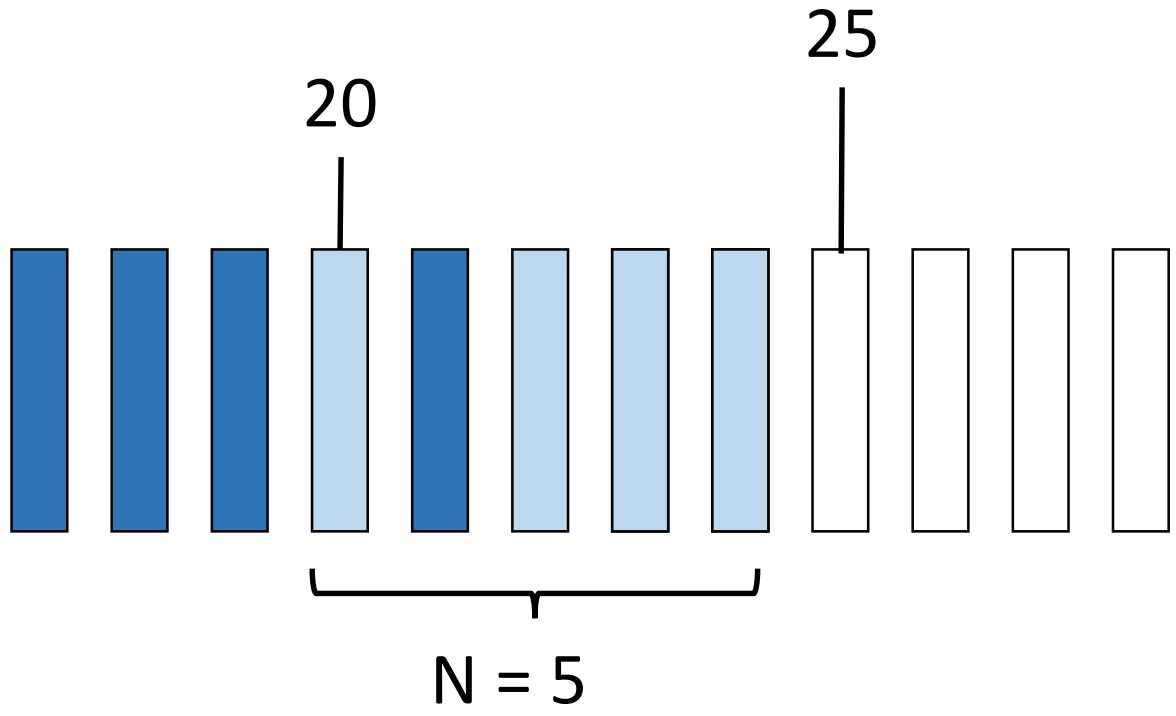
- Now
  - base = 20
  - nextseqnum = 24
  - N = 5
- At this moment, what if new data comes?
- There is still vacancy and it can be sent immediately!
- Start another timer for it.

Sender: Invocation from above



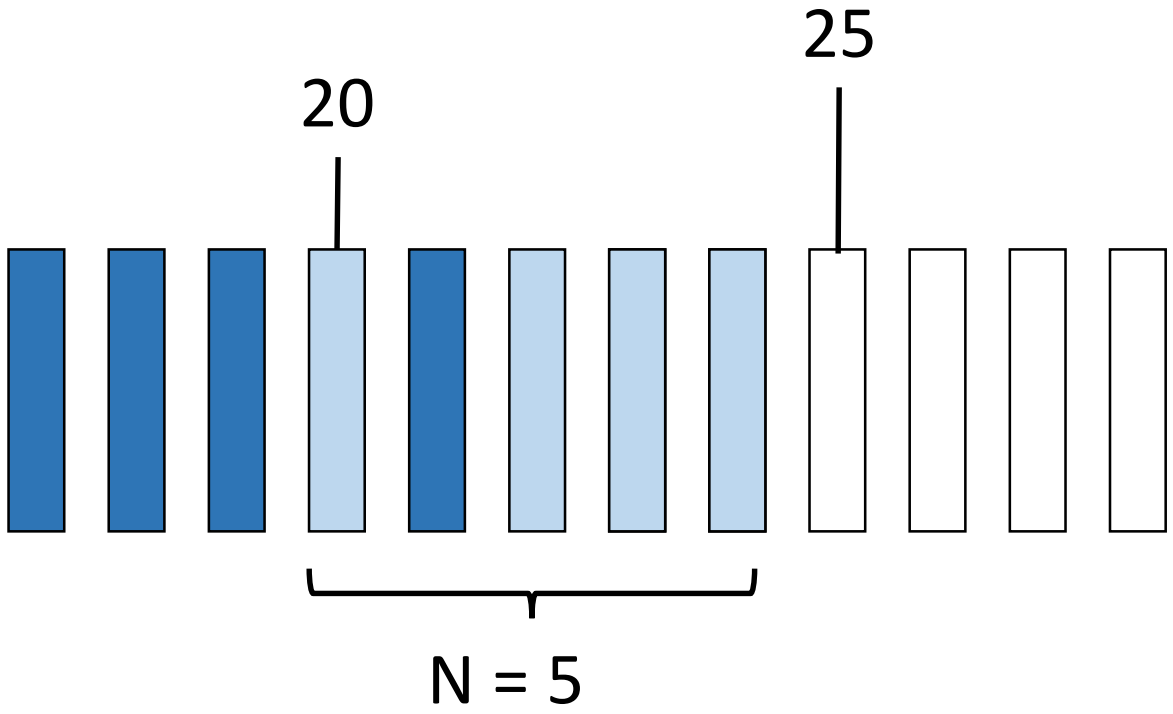
- Now
  - base = 20
  - nextseqnum = 25
  - N = 5

Sender: Invocation from above



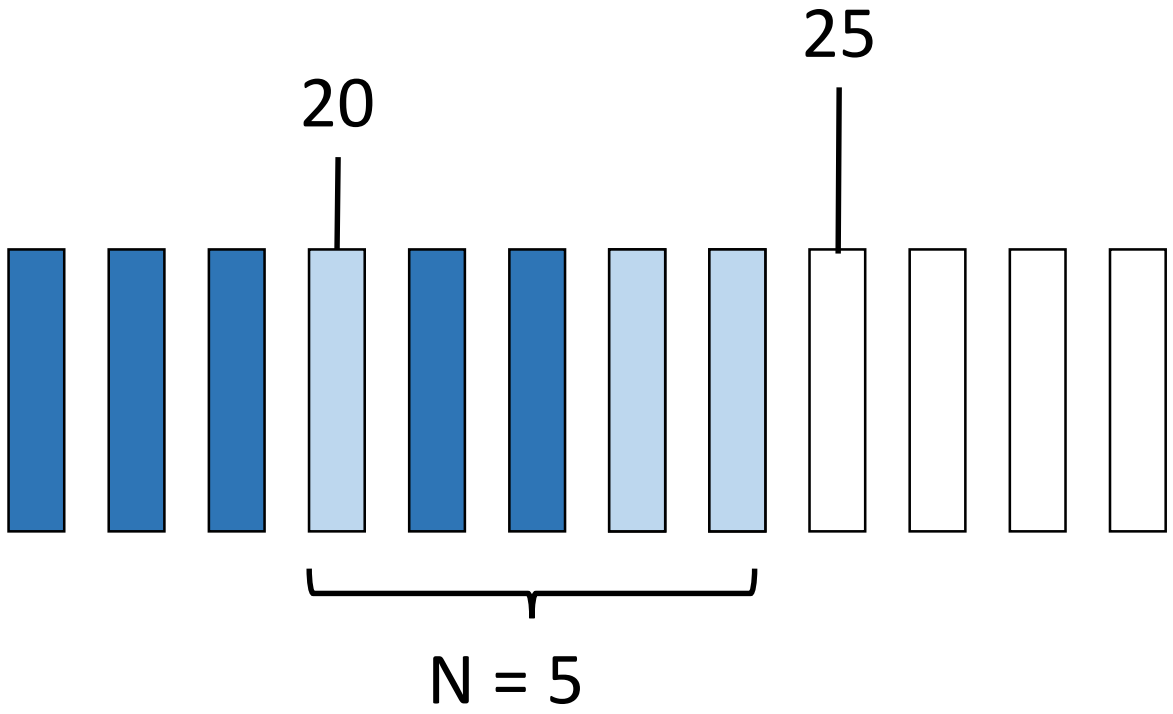
- Now
  - base = 20
  - nextseqnum = 25
  - N = 5
- If here comes new data, sender cannot send it immediately.
- Because the current window is full.

Sender: Receipt of an ACK



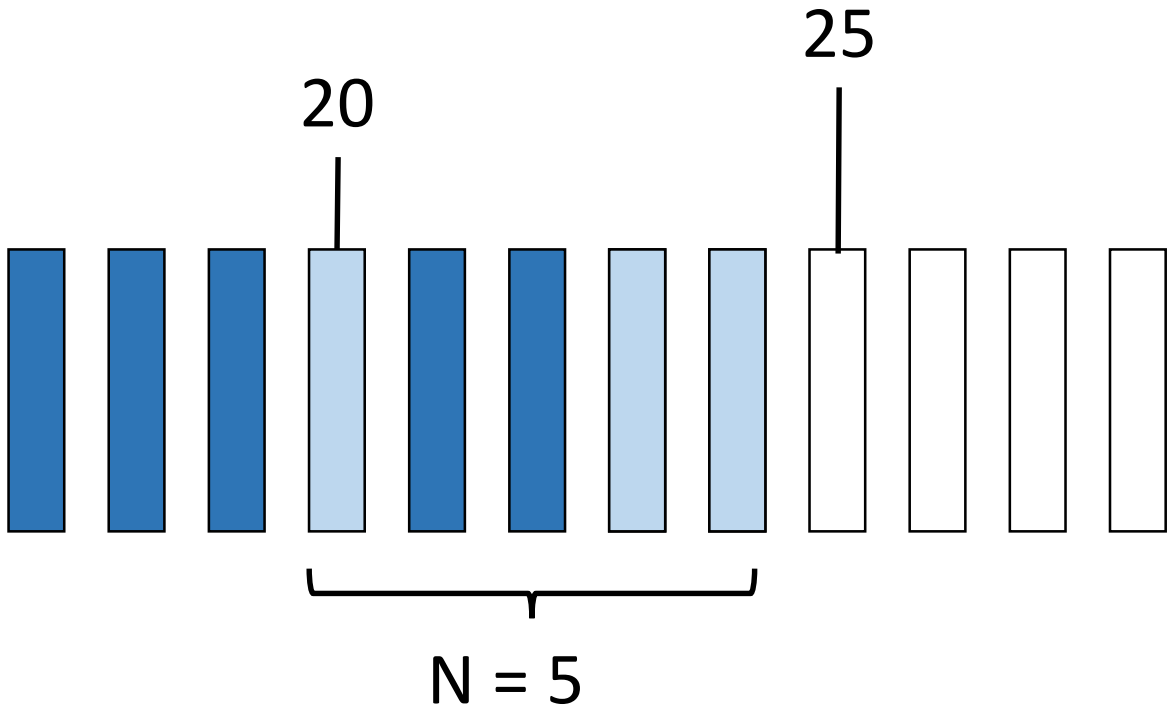
- Now
  - base = 20
  - nextseqnum = 25
  - N = 5
- What will happen if the sender receives the ACK(22)?

Sender: Receipt of an ACK



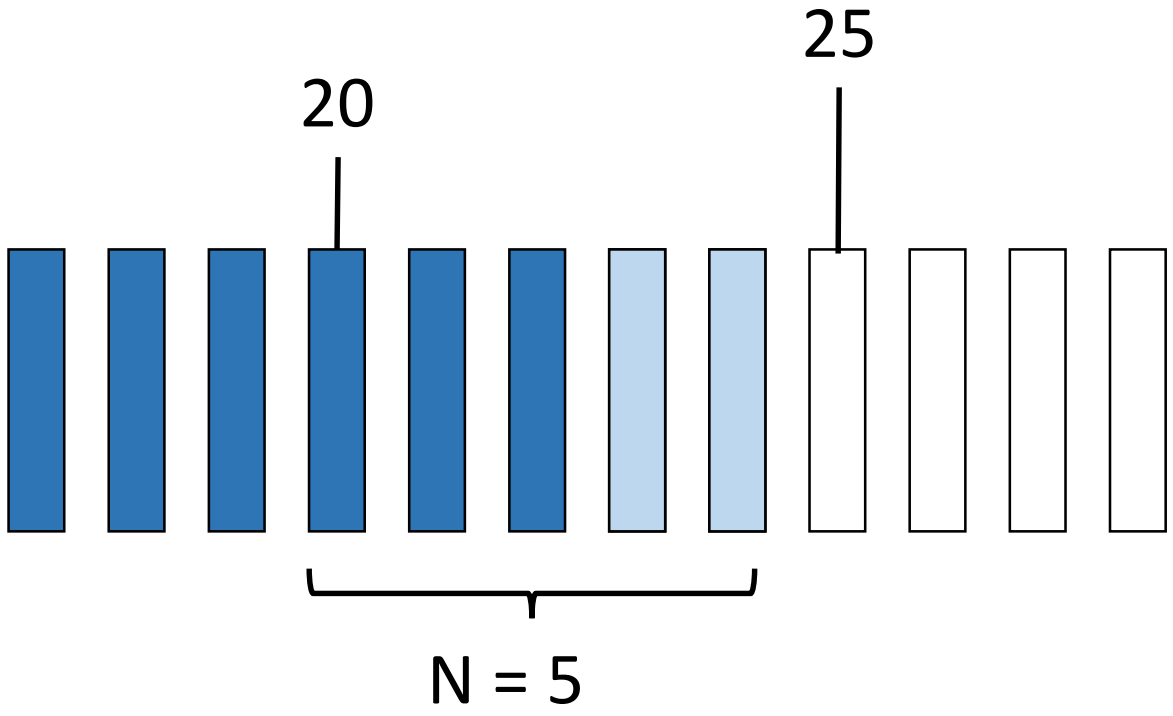
- Now
  - base = 20
  - nextseqnum = 25
  - N = 5
- Simply mark 22 as ACKed.

Sender: Receipt of an ACK



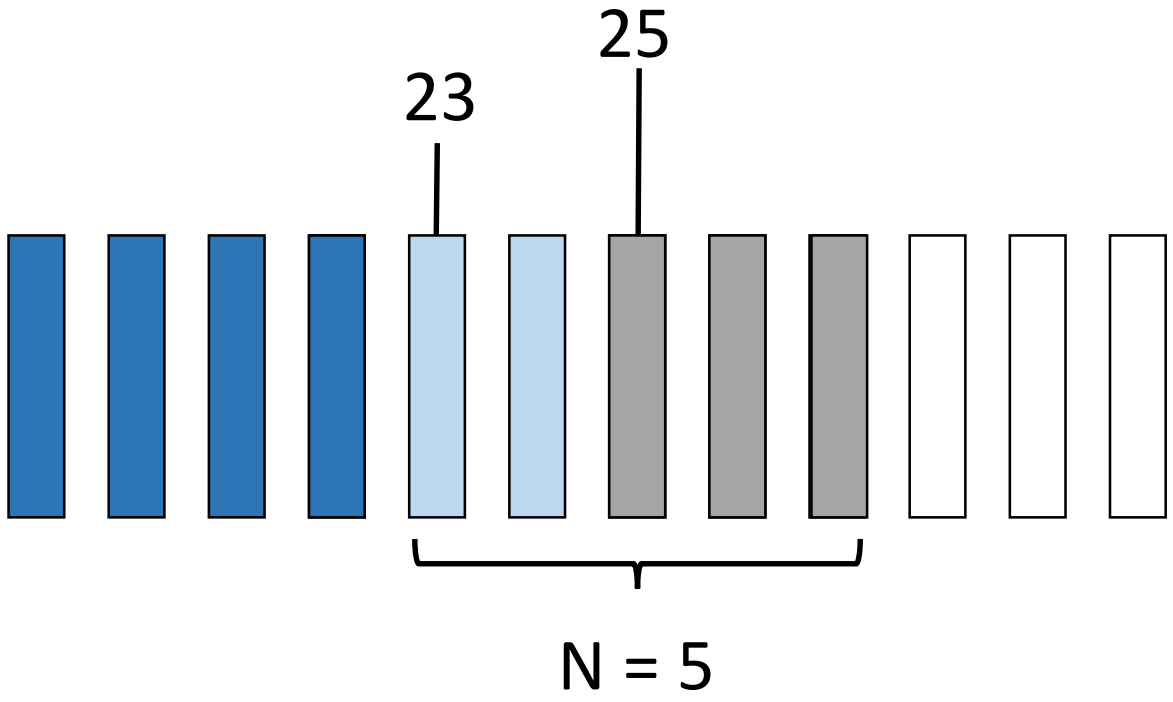
- Now
  - base = 20
  - nextseqnum = 25
  - N = 5
- What will happen if the sender receives the ACK(20)?

Sender: Receipt of an ACK



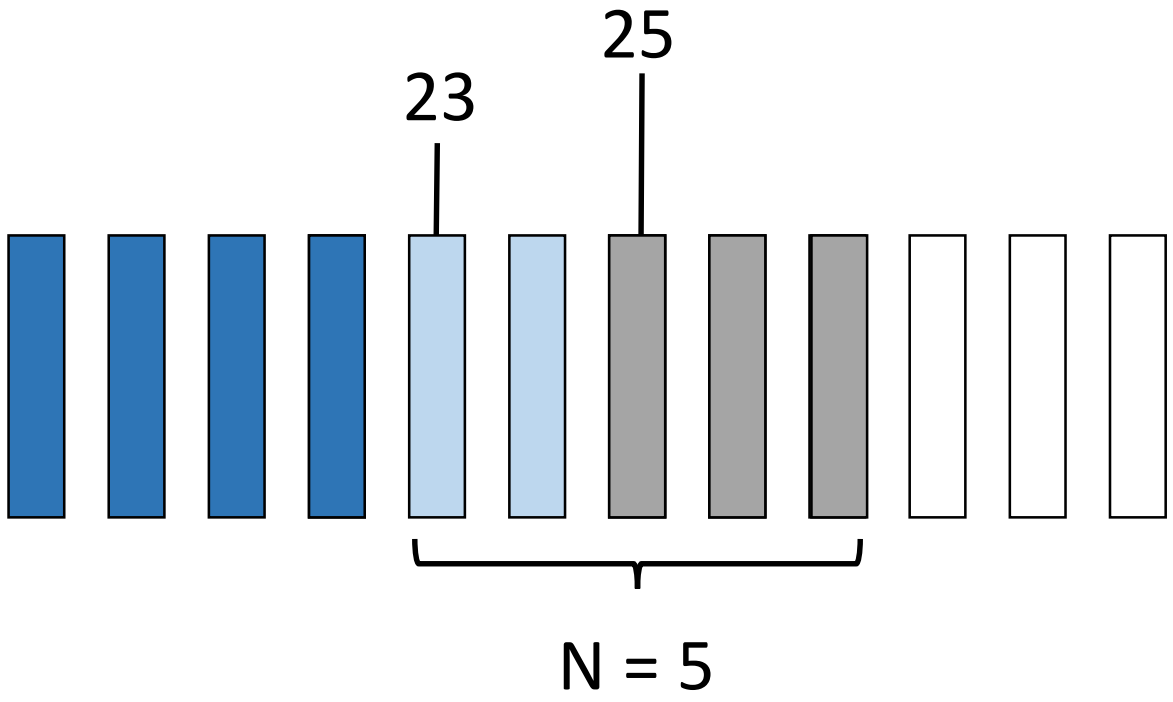
- Now
  - base = 20
  - nextseqnum = 25
  - N = 5
- First mark 20 as ACKed.
- 20 is the smallest unAacked seq #, now we can advance the base to next unacked seq 23

Sender: Receipt of an ACK



- Now
  - base = 23
  - nextseqnum = 25
  - N = 5
- Window has vacancy now!

Sender: timeout event

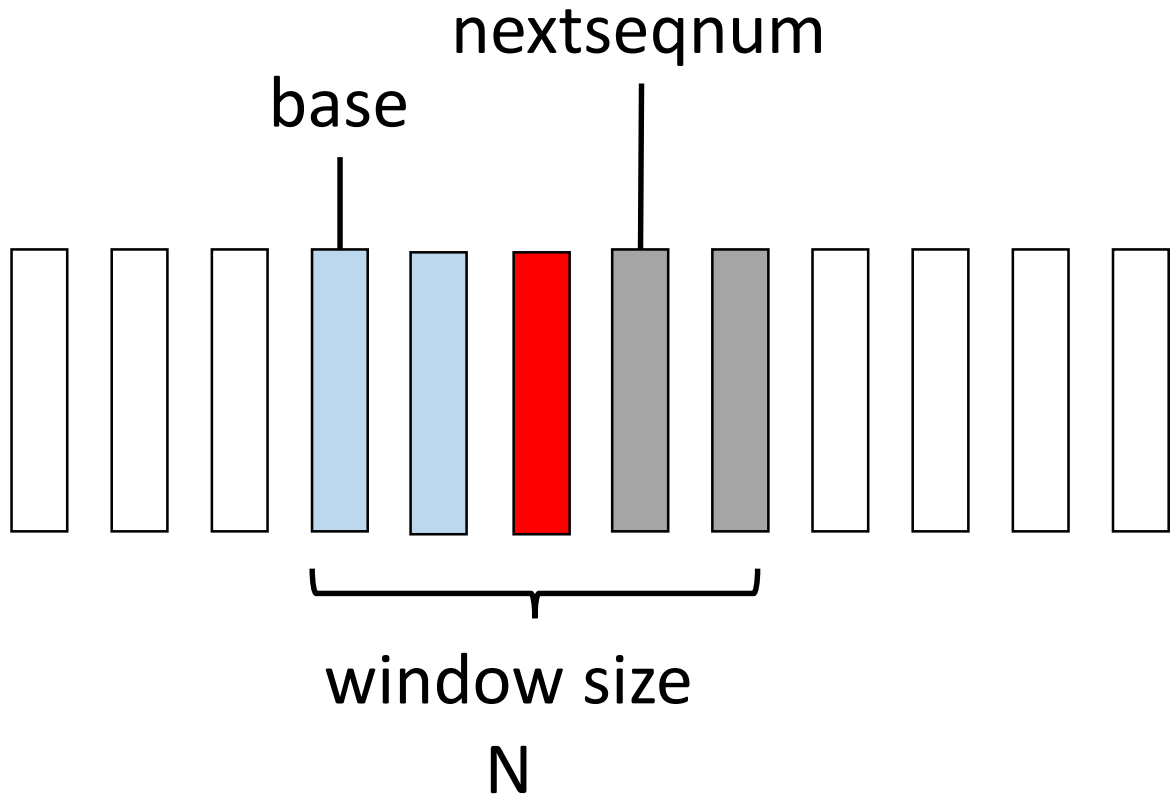


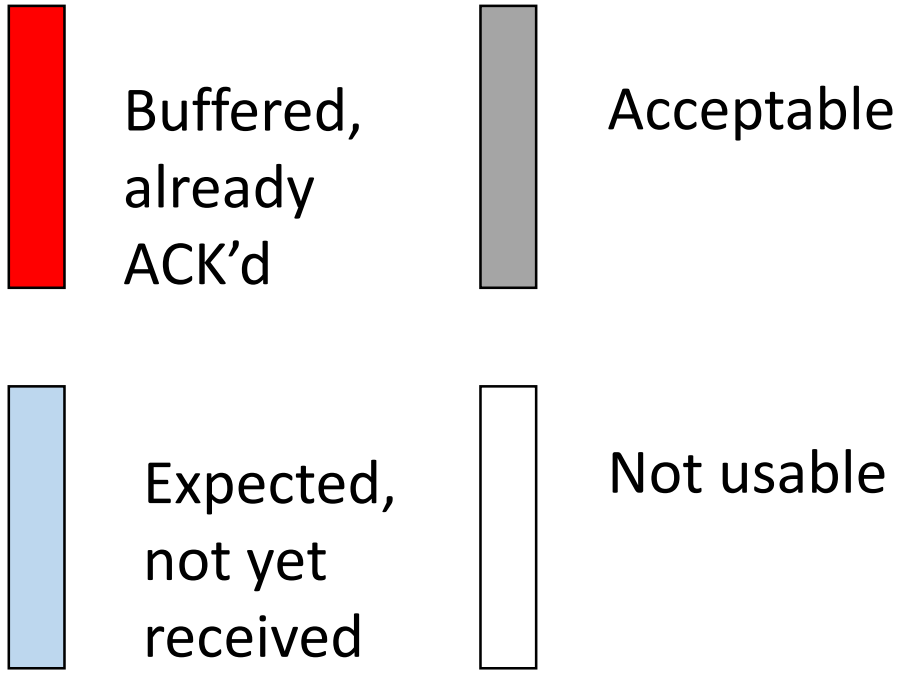
- Now
  - base = 23
  - nextseqnum = 25
  - N = 5
- What if sender doesn’t receive ACK(24) after the timer for packet 24 timeout?
- Sender only re-send packet 24 and reset the timer.

Select Repeat Overview: Receiver

- Receiver Ack individual packet.
  - Out-of-order packet
    - Ack it and buffer it
  - In-order packet
    - Ack it and deliver it (also deliver in-order buffered packets)
    - Advance window
- Suppose receive an ACKed packet again
  - ACK it again
  - Drop the packet

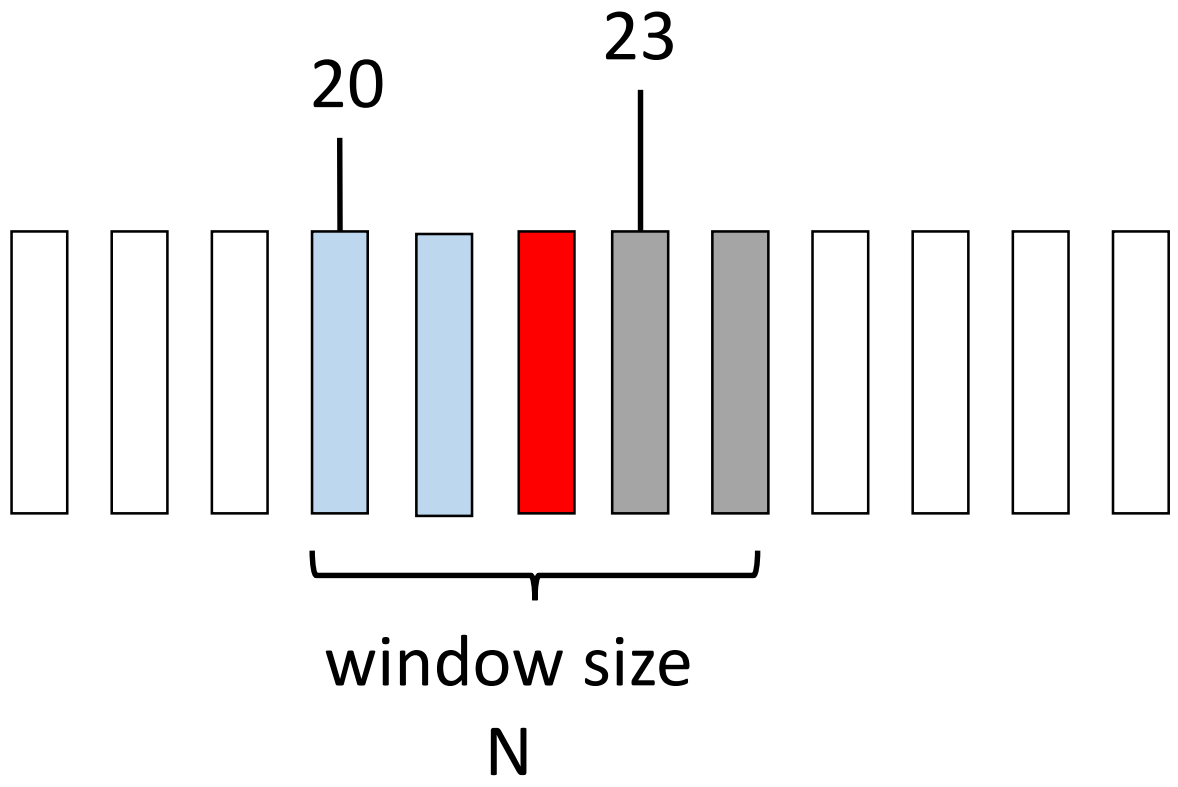
Selective Repeat Overview: Receiver





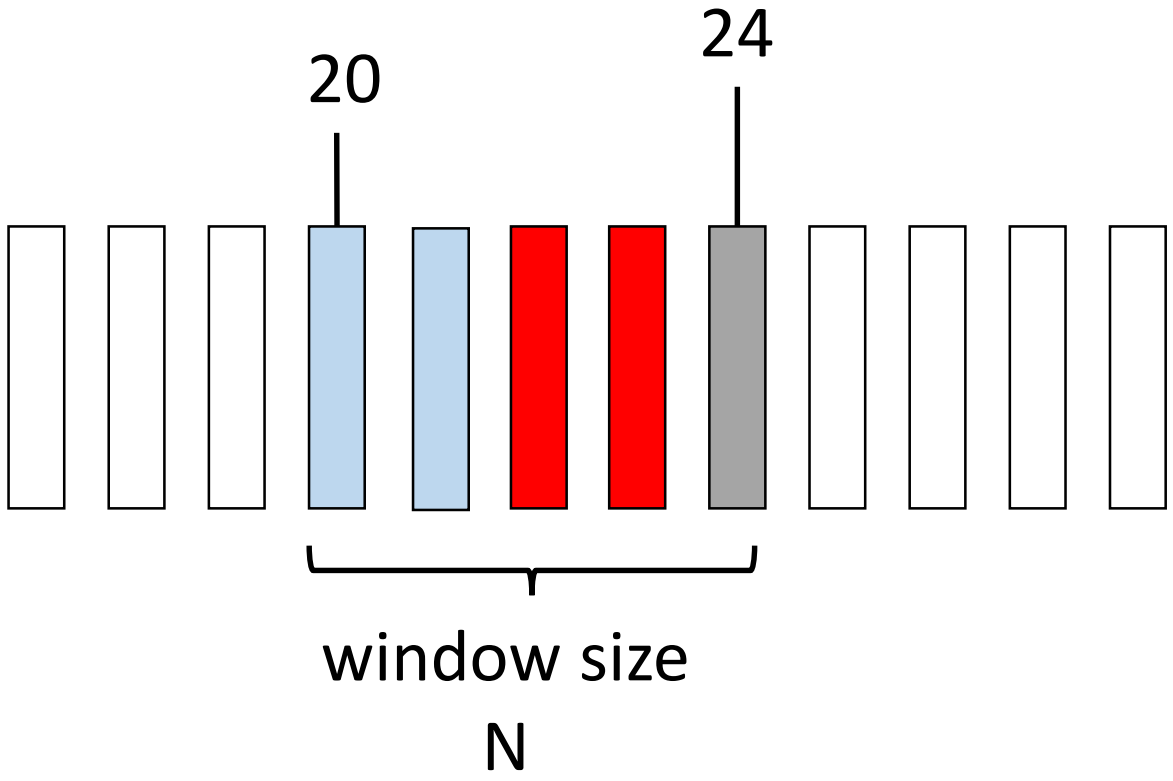
- base: the seqnum of the oldest un-ACK'd packet
- nextseqnum: smallest unused seqnum

Selective Repeat Overview: Receiver



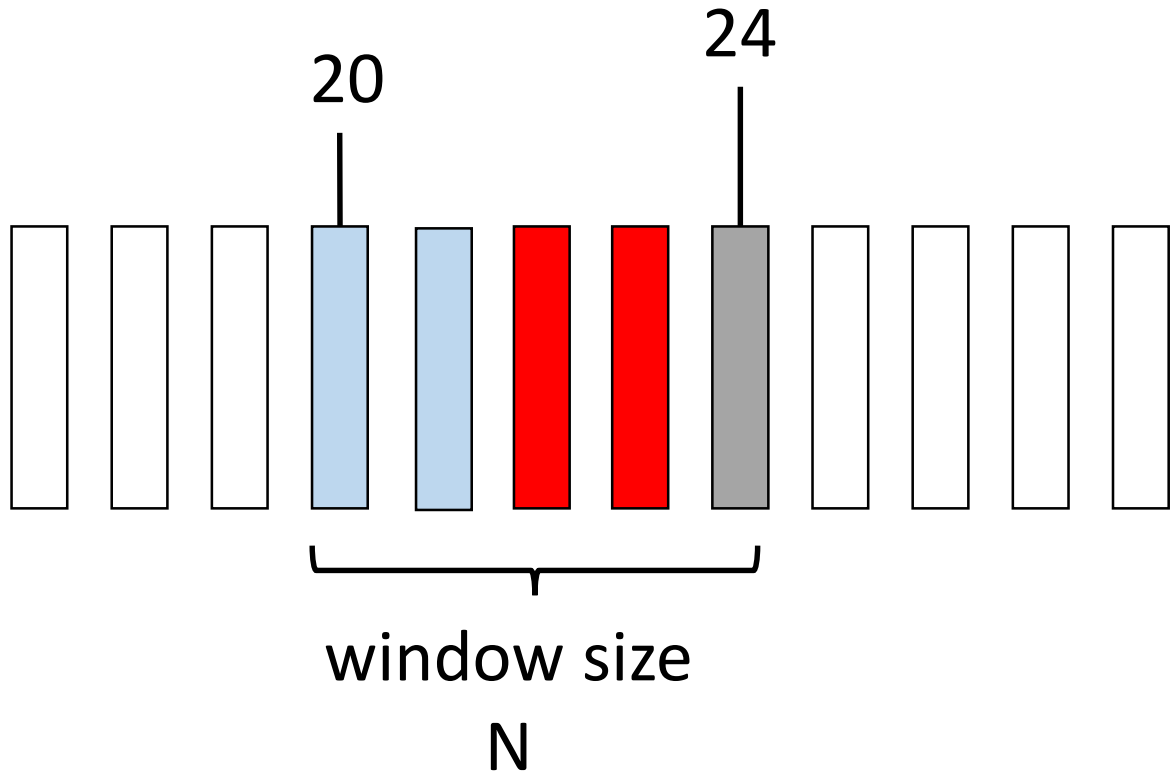
- now
  - base = 20
  - nextseqnum = 23
  - N = 5
- What will happen if the receiver receives the packet 23?

Selective Repeat Overview: Receiver



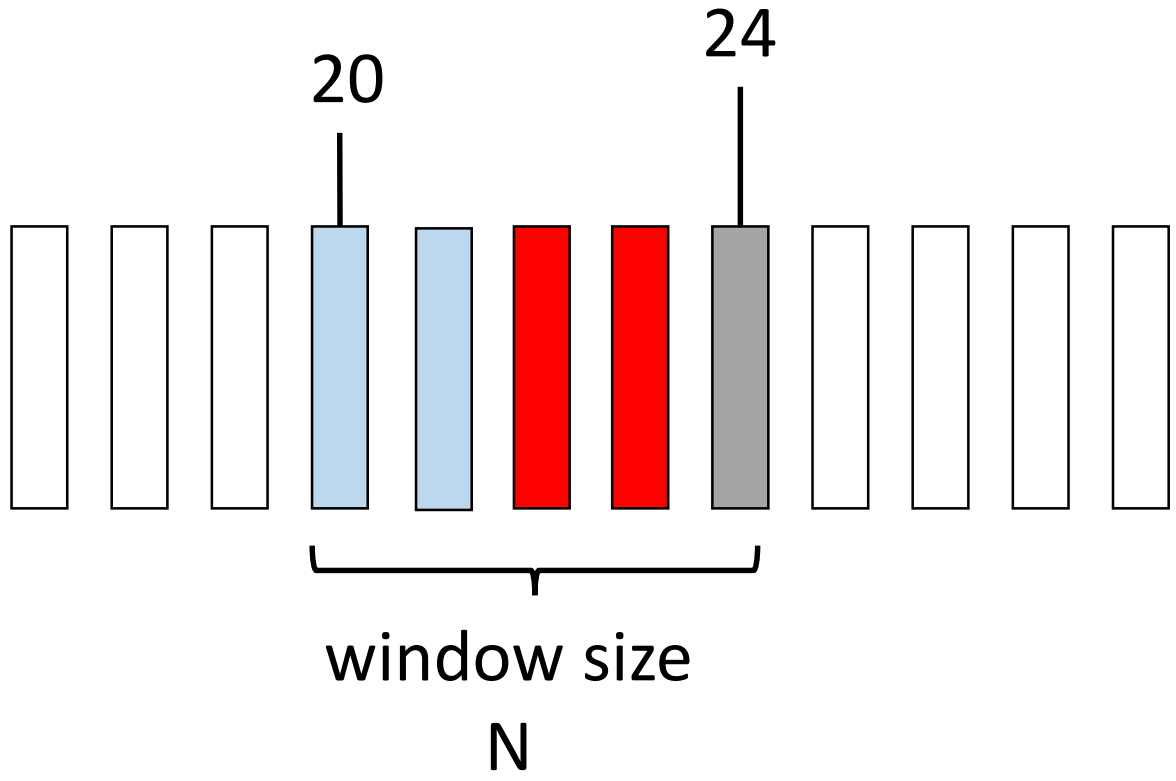
- now
  - base = 20
  - nextseqnum = 24
  - N = 5
- Buffer it. ACK it.

Selective Repeat Overview: Receiver



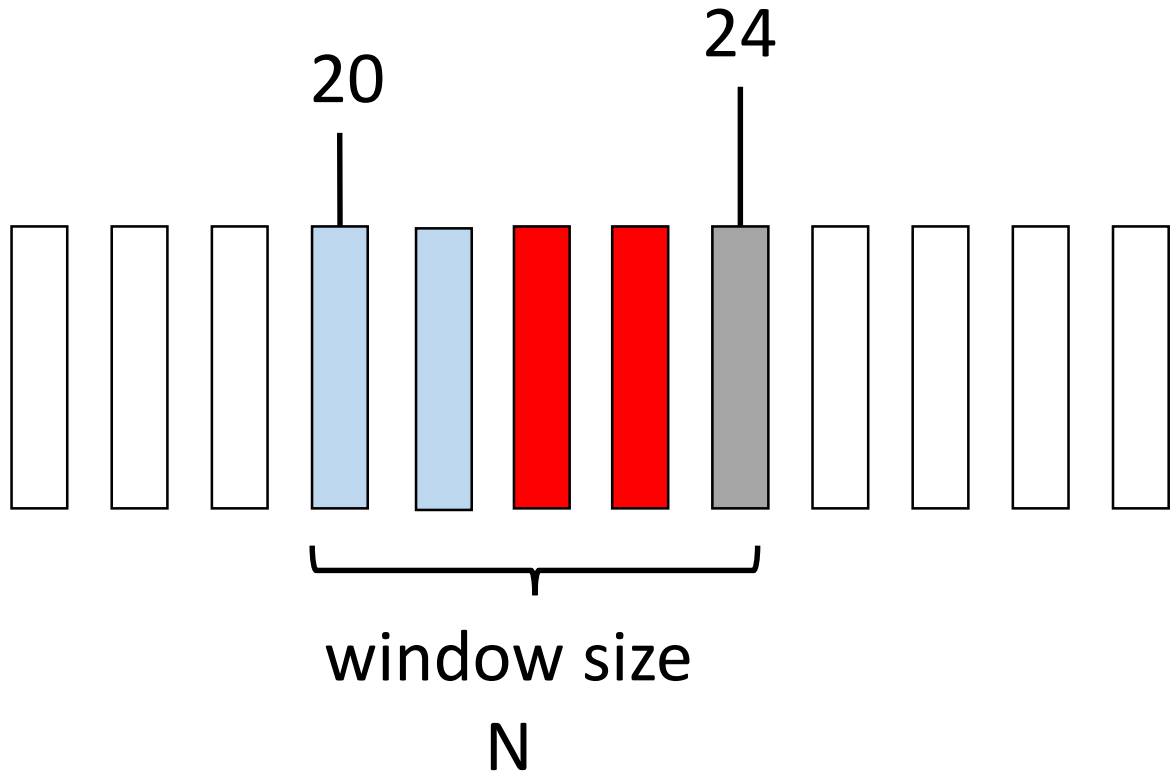
- now
  - base = 20
  - nextseqnum = 24
  - N = 5
- What if receive packet 23 again?

Selective Repeat Overview: Receiver



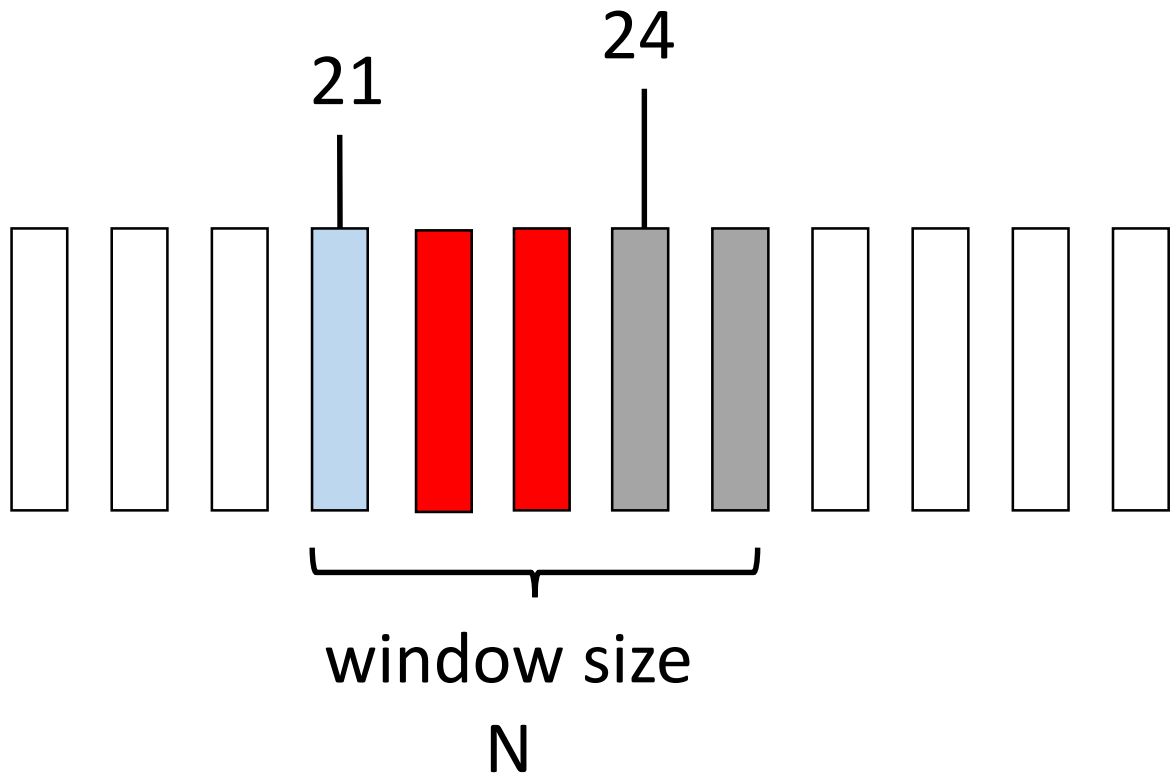
- now
  - base = 20
  - nextseqnum = 24
  - N = 5
- ACK it again, but drop it!

Selective Repeat Overview: Receiver



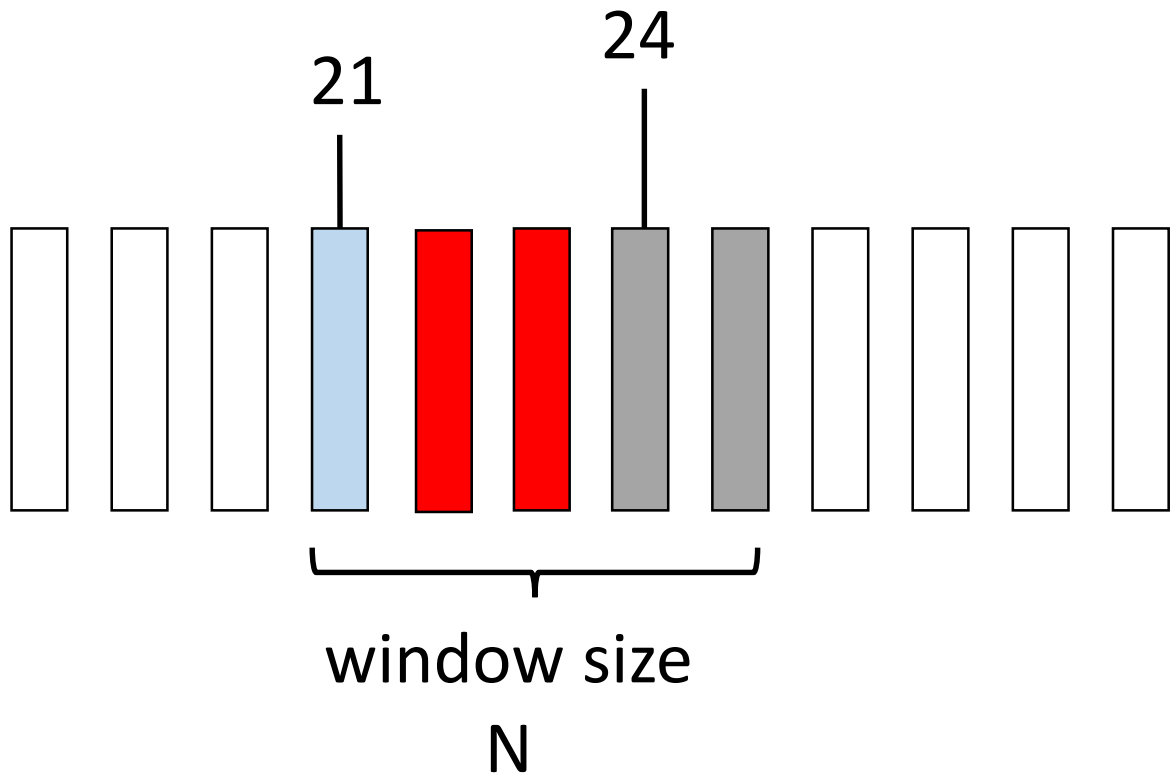
- now
  - base = 20
  - nextseqnum = 24
  - N = 5
- What if receive packet 20?

Selective Repeat Overview: Receiver



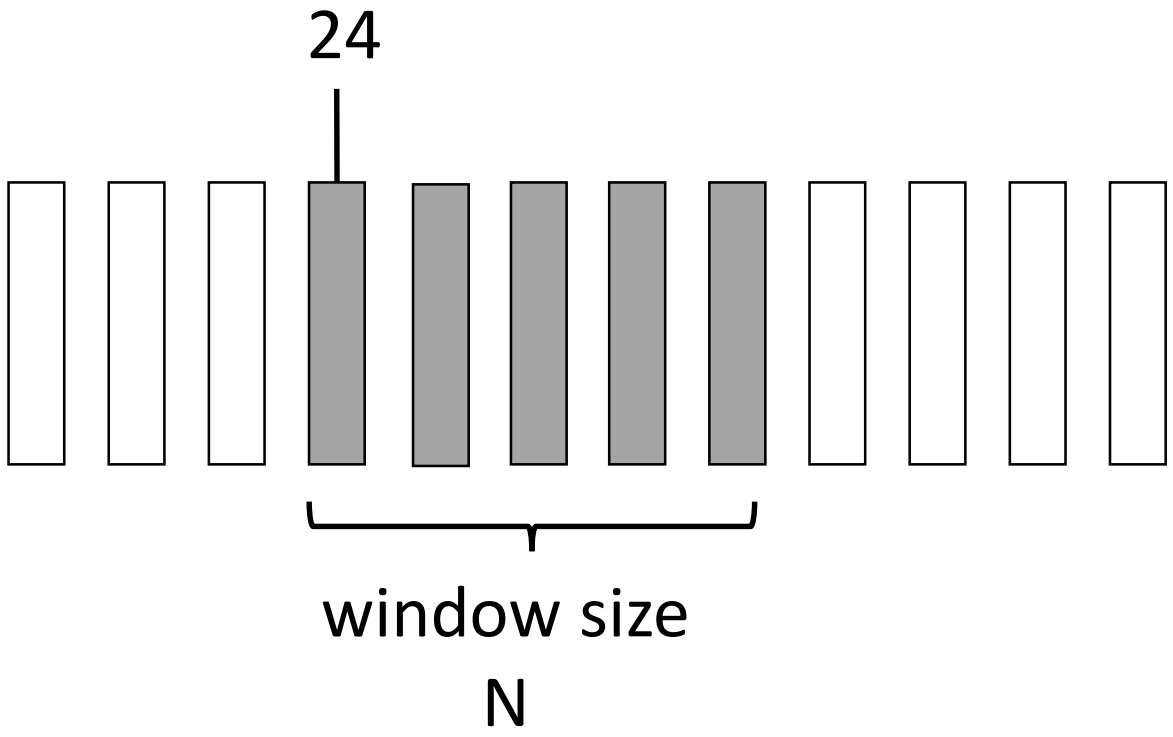
- now
  - base = 20
  - nextseqnum = 24
  - N = 5
- ACK it, deliver it! Advance window

Selective Repeat Overview: Receiver



- now
  - base = 21
  - nextseqnum = 24
  - N = 5
- What if receive packet 21?

Selective Repeat Overview: Receiver



- now
  - base = 24
  - nextseqnum = 24
  - N = 5
- ACK it, deliver [21,23], advance window

Next week

- Call back
  - Mutual Exclusions
  - Condition Variables
- Implementation Hints
  - MYSR_Packet
  - Window is full?
  - Timer
  - Terminate
  - Multi-threading

Thanks!

- If you have question, please post your question in our Piazza group