CIS 457 - Data Communications

Nathan Bowman

Images taken from Kurose and Ross book

Flow Control in TCP

TCP sender puts data in send buffer then sends it over network

TCP receiver stores data in receive buffer until ready to read

Buffers have finite amount of space

What if receiver is busy doing something else and not currently reading messages?

If sender continues pushing data onto network, receiver will eventually run out of space in buffer

This is wasteful -- packets need to be re-sent

To avoid this, TCP institutes **flow control**

Flow control matches the pace of sender and receiver to prevent overflowing buffer at receiver

Not the same as congestion control, which has a similar effect but is done for good of network rather than individual processes

Sender maintains **receive window** -- essentially tracks amount of free buffer space at receiver

Needs constant updates from receiver to do so

To simplify bookkeeping, we will assume receiver drops out-of-order packets, though we know that is not generally the case

Also, assume sender A is sending large file to receiver B, and B has nothing to send to A

B keeps track of several variables:

- `RecvBuffer` -- size of receive buffer
- `LastByteRead` -- # of last byte read from buffer
- `LastByteRcvd` -- # of last byte received

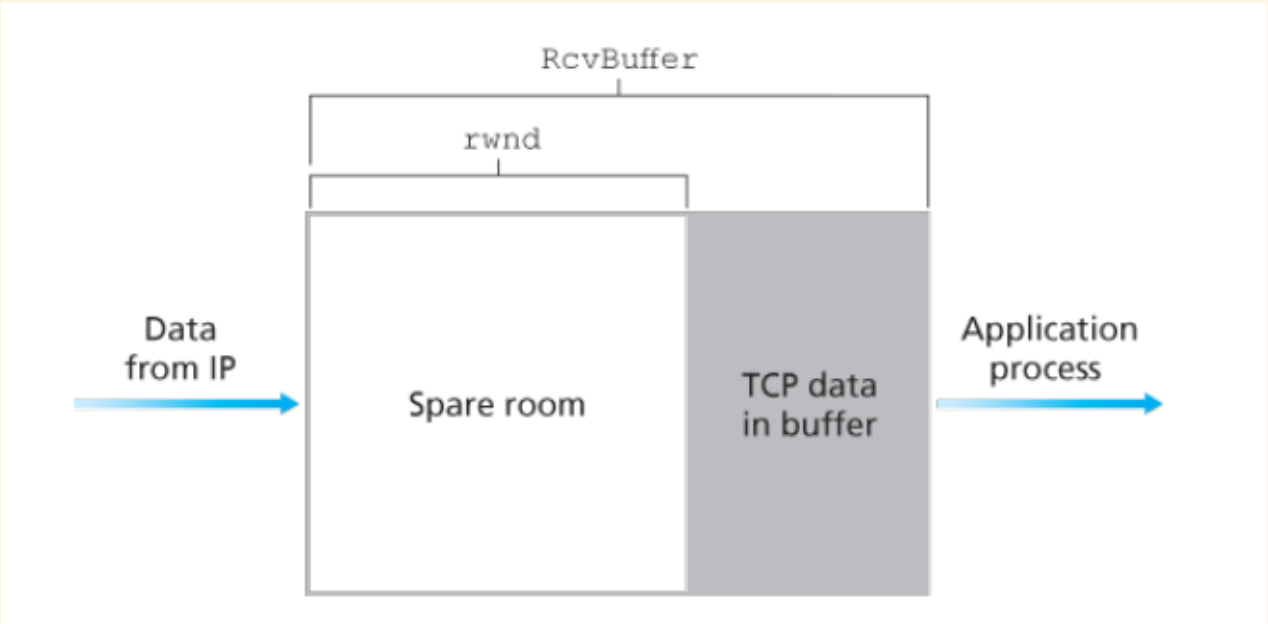
To ensure buffer is not overflowed, need to have

```
LastByteRcvd - LastByteRead <= RecvBuffer
```

Define receive window (`rwnd`) as:

```
rwnd = RecvBuffer - (LastByteRcvd - LastByteRead)
```

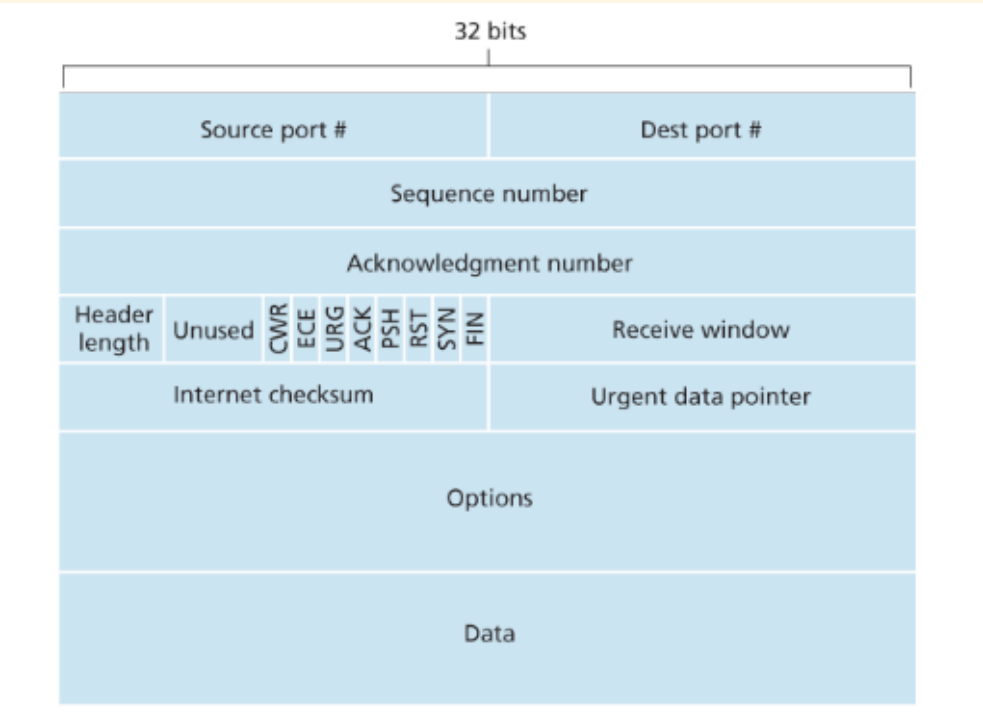
`rwnd` gives amount of free space in receive buffer


RcvBuffer
rwnd
Data
from IP
Spare room
TCP data
in buffer
Application
process

Note that A is the one that needs to know `rwnd`, and it has *none* of this information

B computes `rwnd` as described and sends to A

`rwnd` included in every segment sent from B to A


32 bits
Source port #
Dest port #
Sequence number
Acknowledgment number
Header length
Unused
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
Receive window
Internet checksum
Urgent data pointer
Options
Data

Not enough for A to know only how much space B has -- also must account for outstanding segments

Host A tracks a few variables of its own:

- `LastByteSent`
- `LastByteAcked`

Amount of data that could be already in connection on its way to B is

```
LastByteSent - LastByteAcked
```

Need to assume worst case:

- Any information sent but not ACK'd was not accounted for in `rwnd`
- B may wait indefinitely before reading (i.e., `rwnd` will not grow)

Host A must always ensure that

```
LastByteSent - LastByteAcked <= rwnd
```

As long as that is the case, buffer at B will not overflow

As long as

```
LastByteSent - LastByteAcked == rwnd
```

A cannot sent any more segments

This is enough to ensure B has sufficient buffer space

However, one pesky detail comes up:

Assume buffer is full, so B sends `rwnd = 0`

Host A will stop sending data, but how will it know when to start again?

Host B will not be sending ACKs, so A will not receive updates about `rwnd`

Protocol specifies that A sends small "dummy" packets when `rwnd` is 0

Host A will still get regular updates when B ACKs these packets

Eventually, B will free up space, `rwnd` will go above 0, and A can send again

Note that UDP has no concept of flow control

Sender will send segments whenever it is ready, and if receiver is not fast enough packets will be dropped