6.033 Spring 2017

Lecture #10

- **Reliable Transport**
- **Window-based Congestion Control**

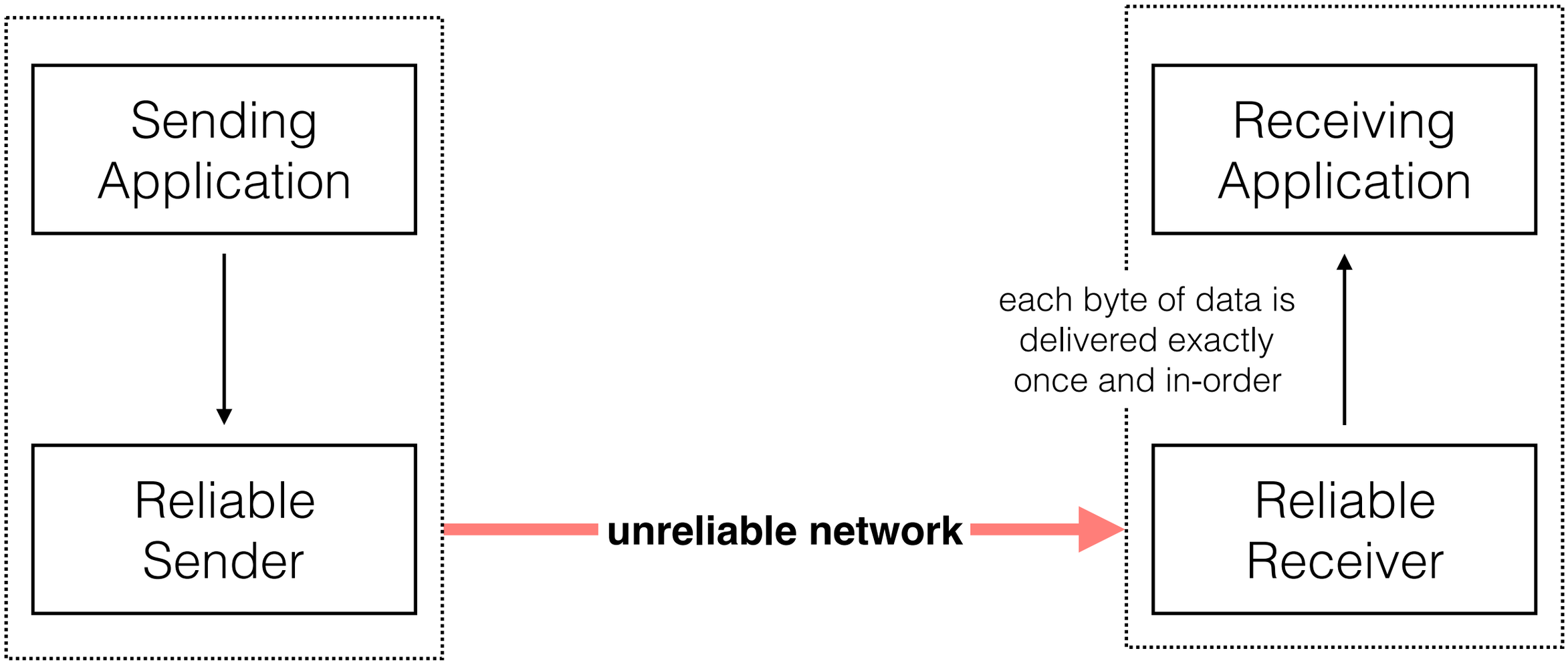
Internet of Problems

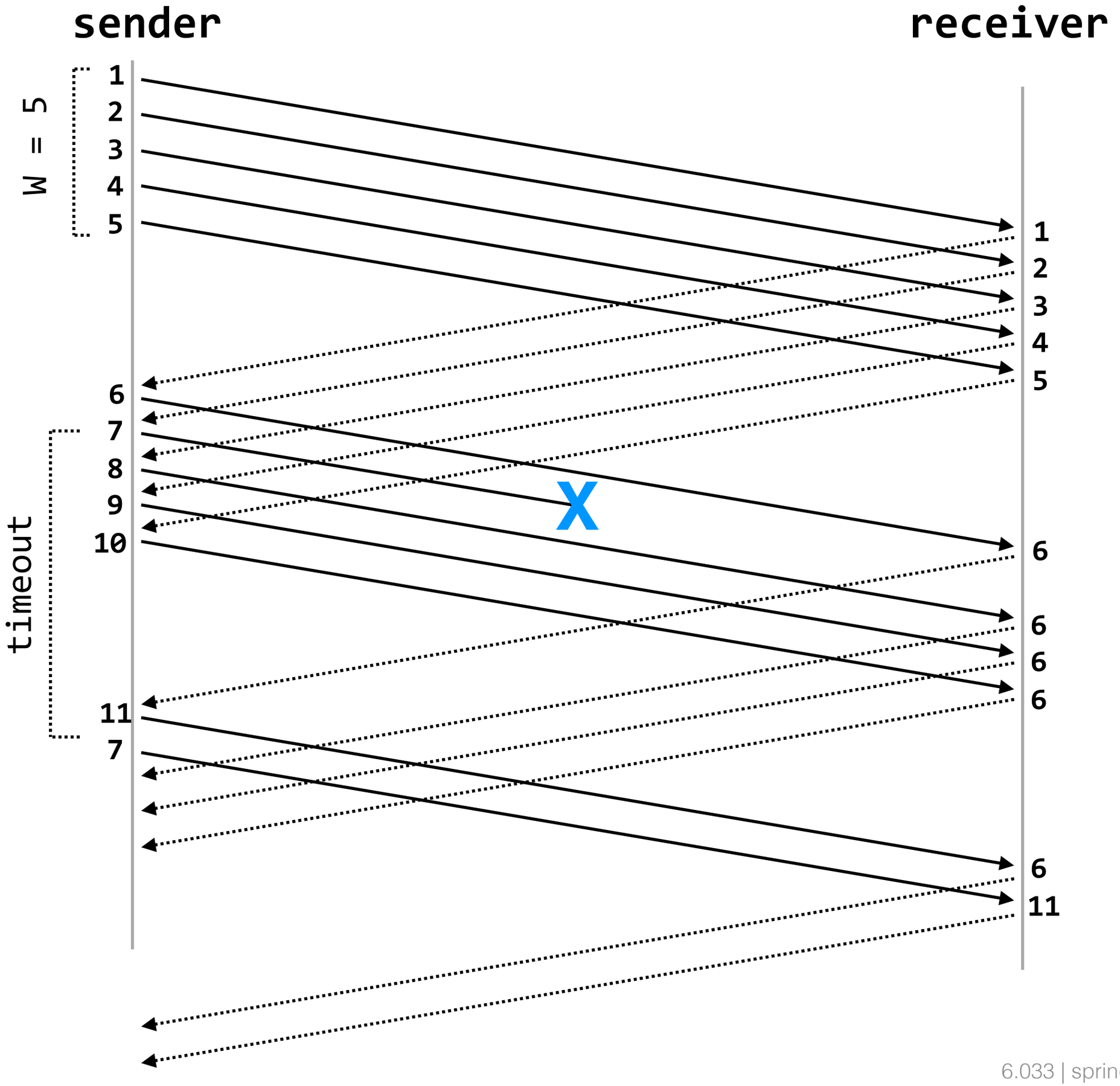
How do we **route** (and address) scalably, while dealing with issues of policy and economy? → **BGP**

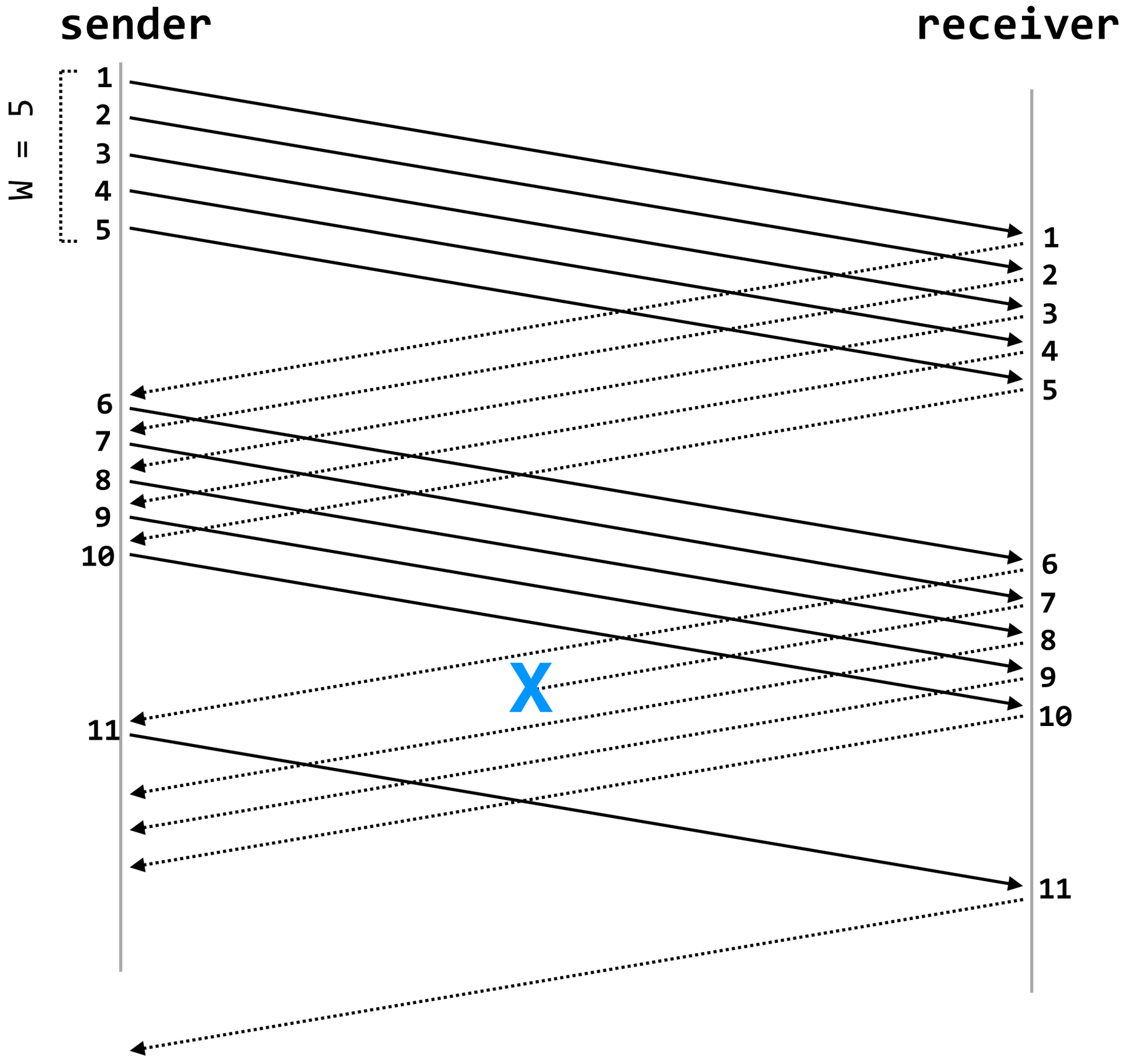
How do we **transport** data scalably, while dealing with varying application demands? → **TCP**

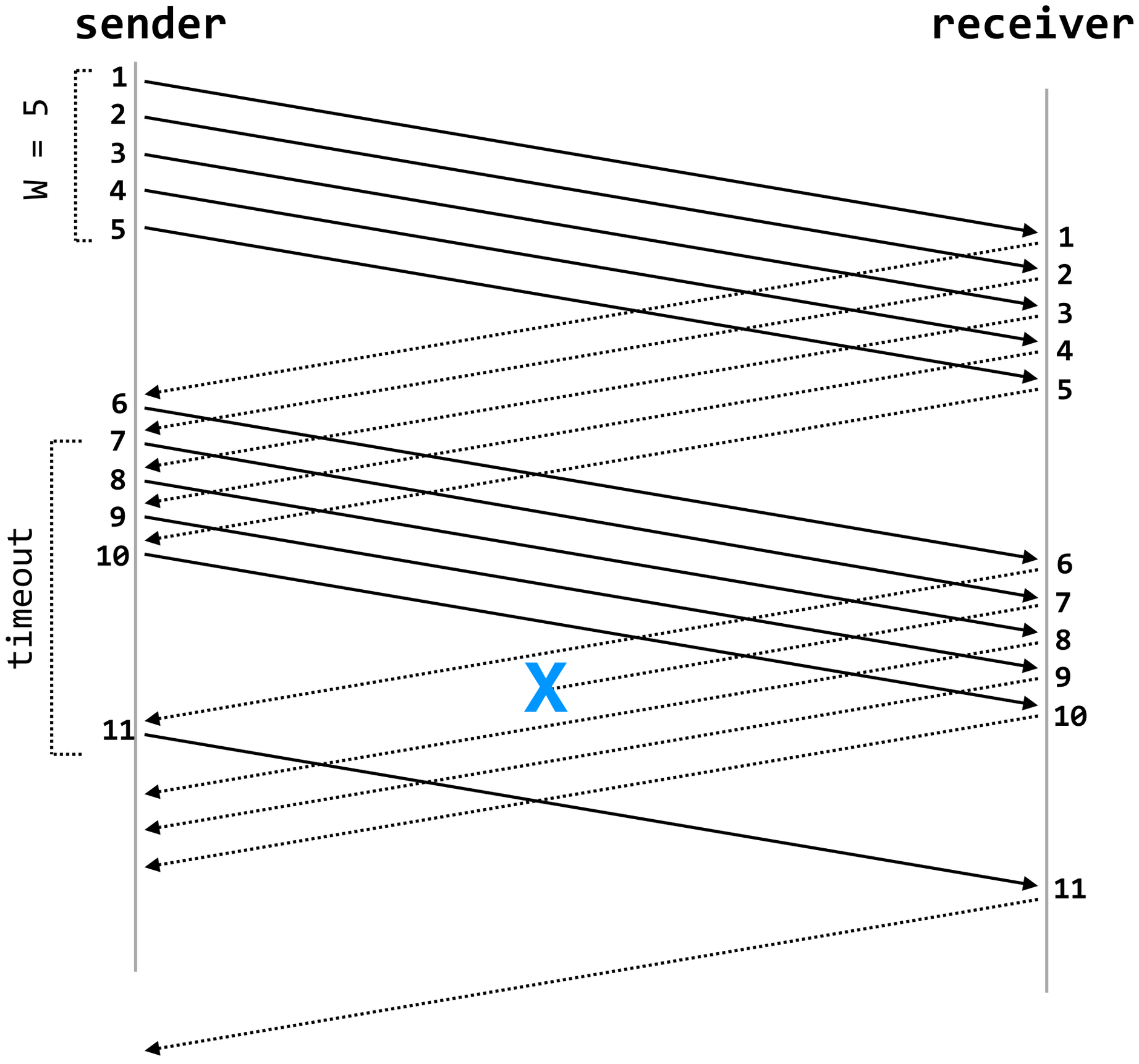
How do we **adapt** new applications and technologies to an inflexible architecture?

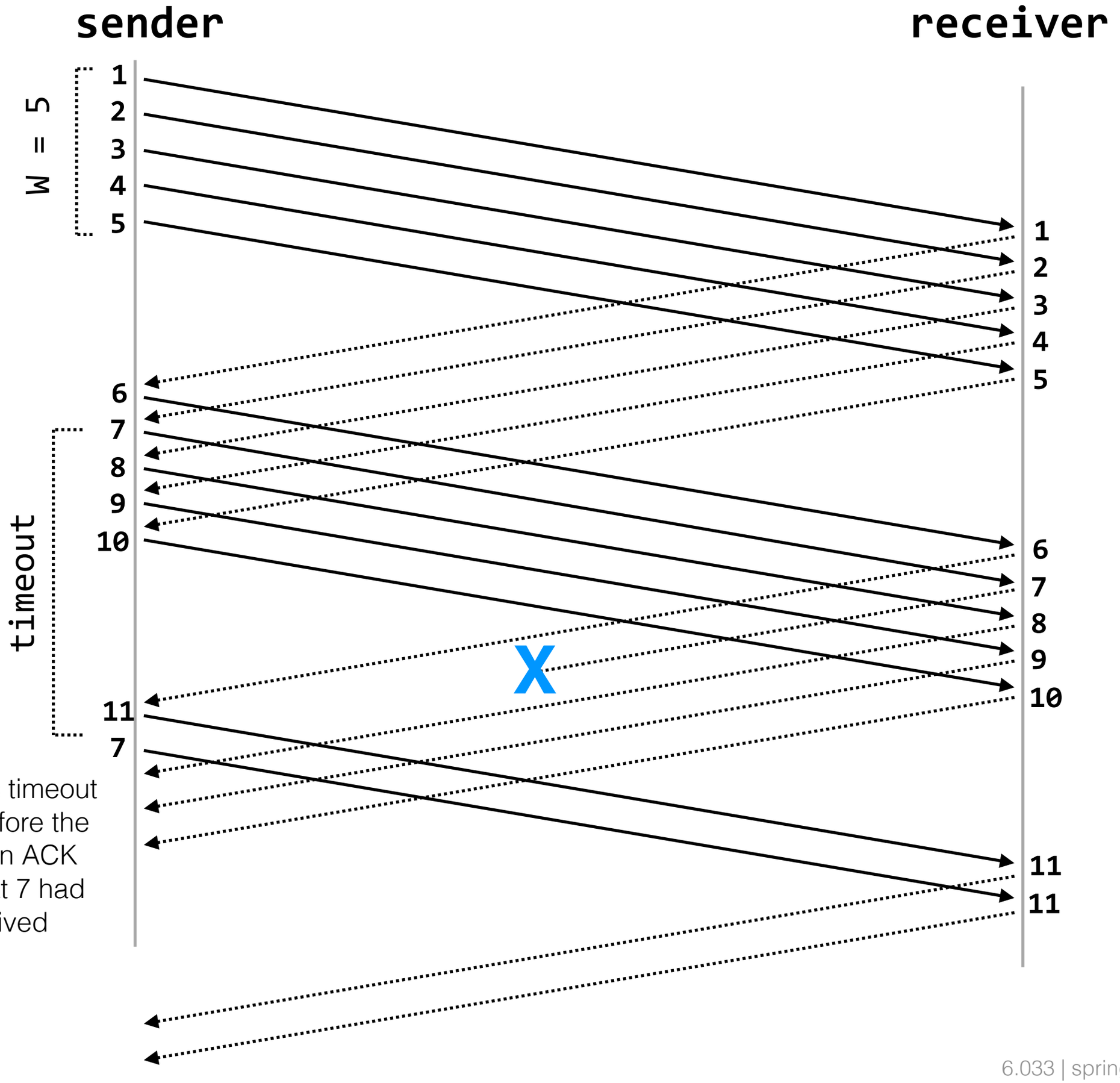
Reliable Transport



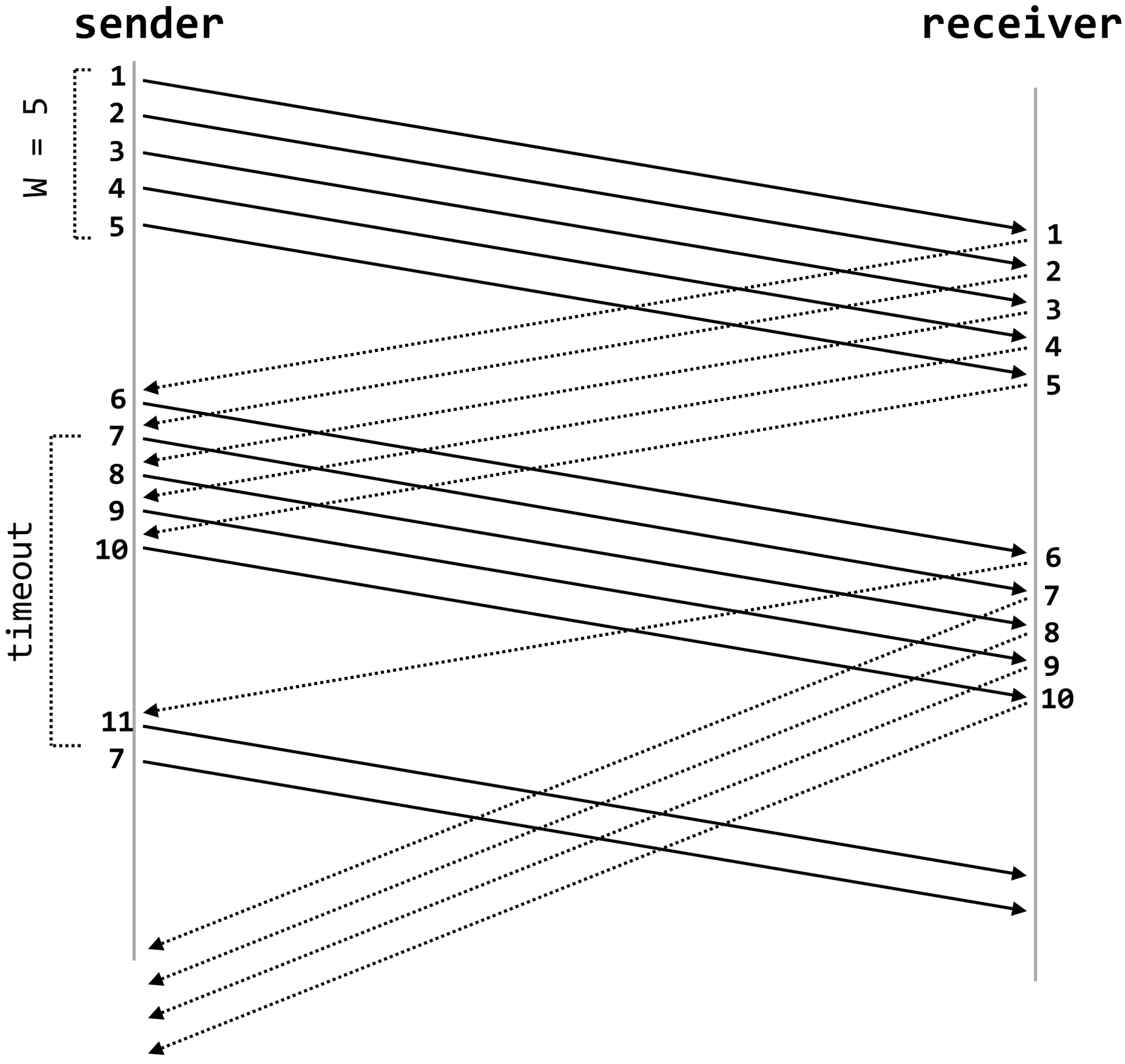

sender
receiver
W = 5
timeout
1
2
3
4
5
6
7
8
9
10
11
X
1
2
3
4
5
6
6
6
6
6
11


sender
receiver
W = 5
1
2
3
4
5
6
7
8
9
10
11
X
1
2
3
4
5
6
7
8
9
10
11


sender
receiver
W = 5
timeout
1
2
3
4
5
6
7
8
9
10
11
X
1
2
3
4
5
6
7
8
9
10
11



notice that this timeout happened before the sender got an ACK indicating that 7 had been received

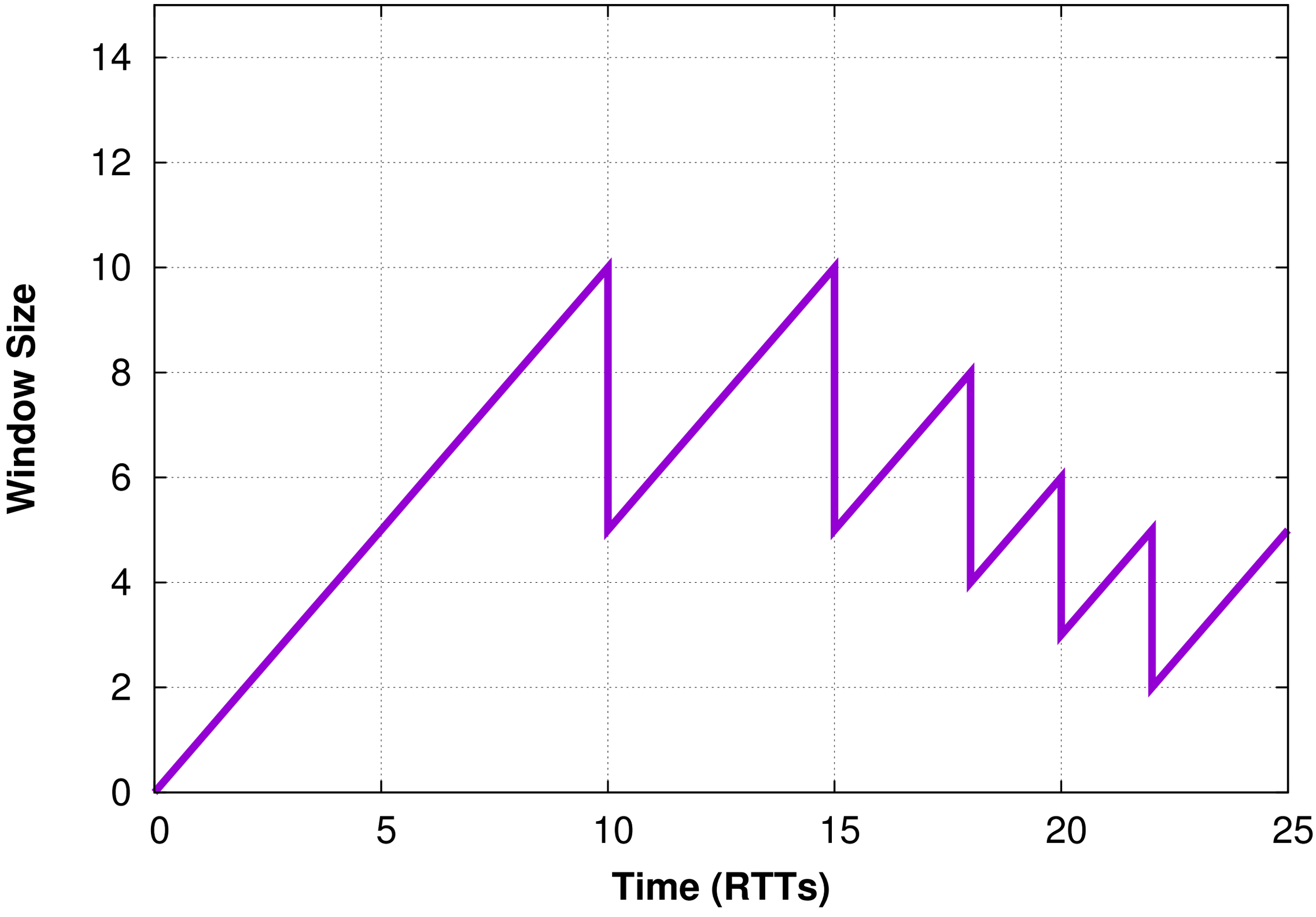

sender
receiver
W = 5
timeout
1
2
3
4
5
6
7
8
9
10
11
7
1
2
3
4
5
6
7
8
9
10

question: what is the correct value for `W`?

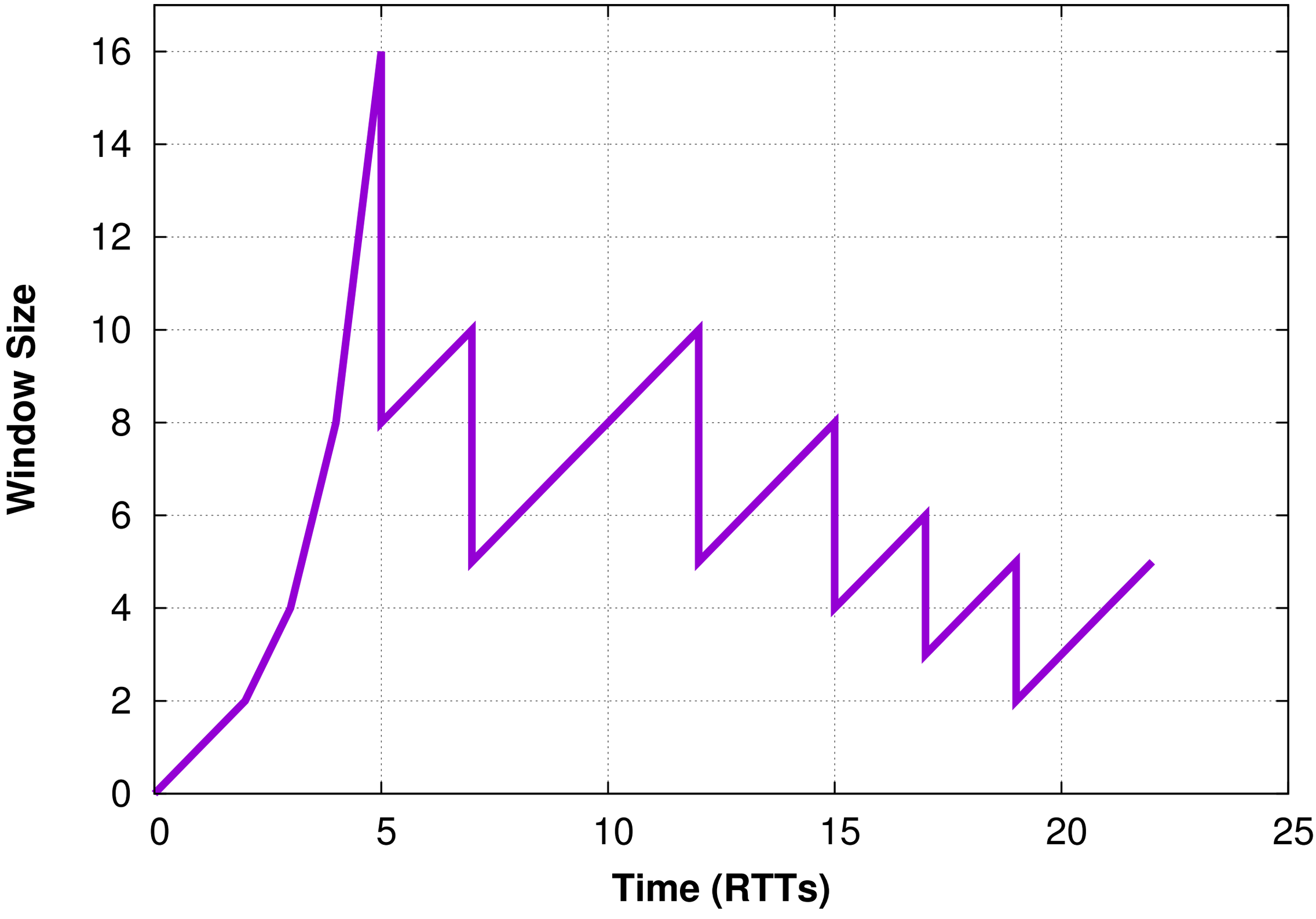
too small → underutilized network
too large → congestion

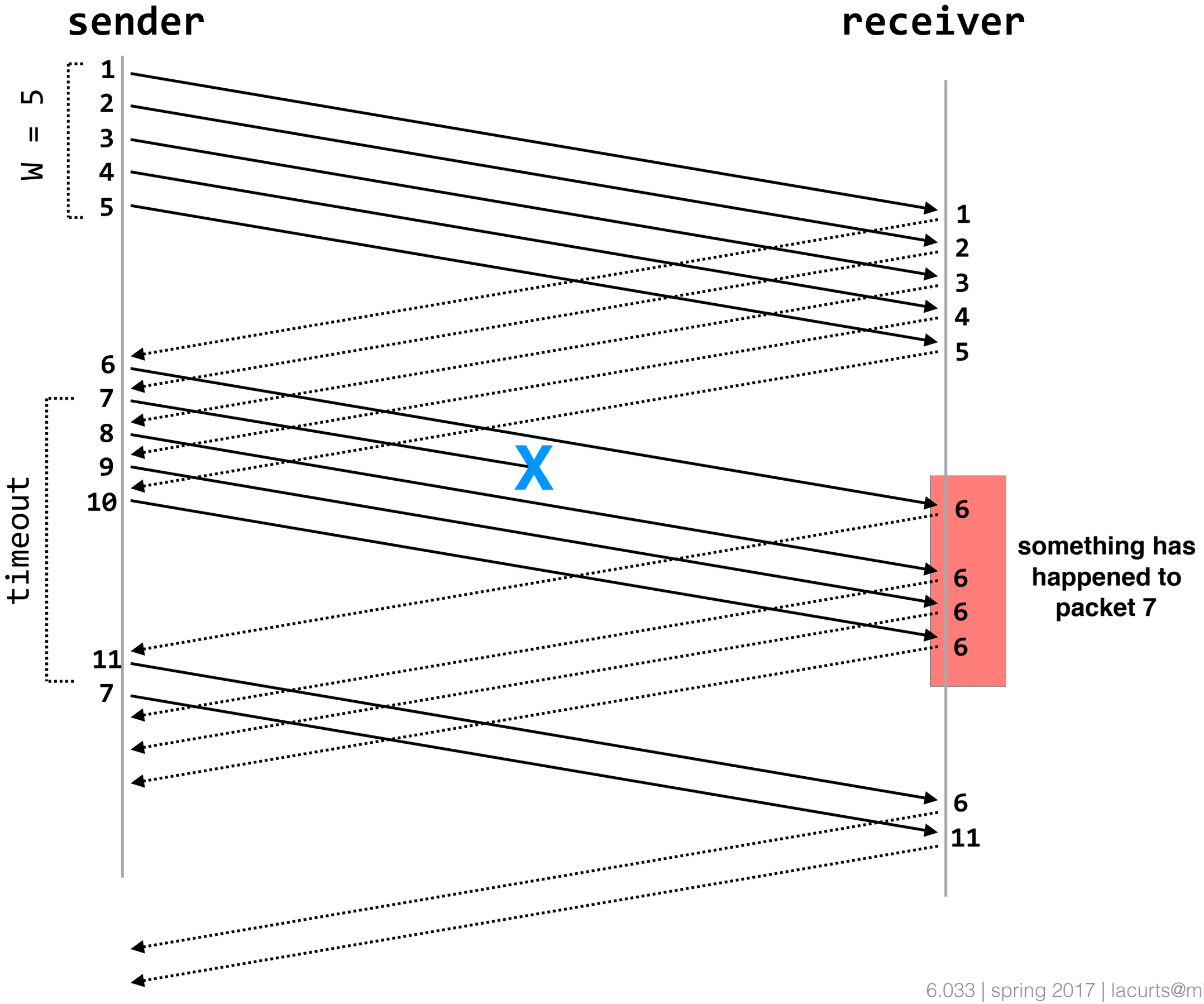
question: how can a single reliable sender, using a sliding-window protocol, set its window size to maximize utilization — but prevent congestion and unfairness — given that there are many other end points using the network, all with different, changing demands?

AIMD

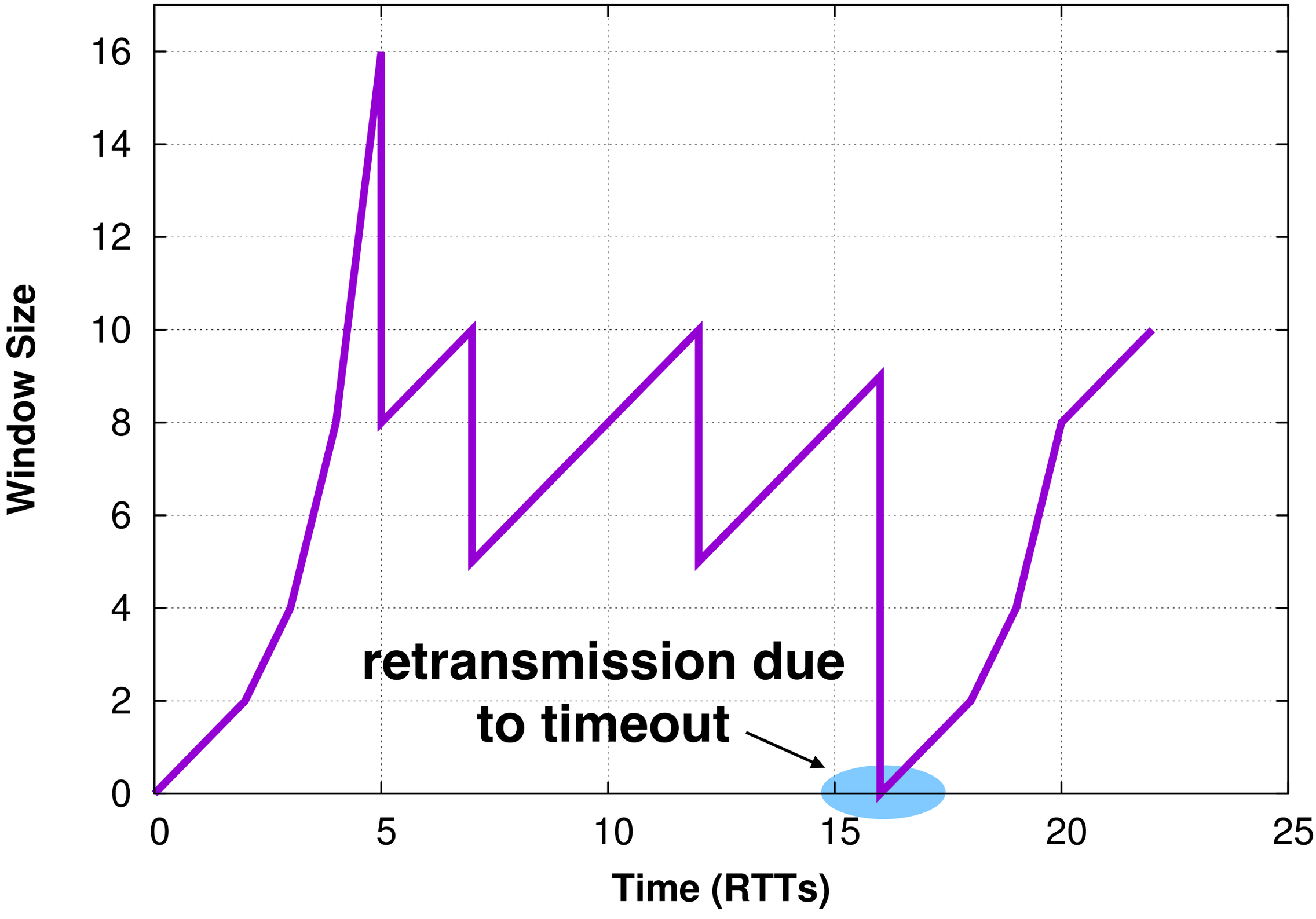


AIMD + Slow Start




sender
receiver
W = 5
timeout
1
2
3
4
5
6
7
8
9
10
11
X
1
2
3
4
5
6
6
6
6
6
6
11
something has happened to packet 7

AIMD + Slow Start



- **TCP** provides **reliable transport** along with **congestion control**: senders increase their window additively until they experience loss, and then back off multiplicatively. Senders also use slow-start and fast-retransmit/fast-recovery to quickly increase the window and recover from loss.

- TCP has been a massive success, but **senders don't react to congestion until queues are already full**. Is there a better way?