6.1800 Spring 2024

Lecture #11: Reliable Transport

adding reliability while also keeping things efficient and fair

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024

6.1800 in the news

SPECIAL SEMINAR

Reconstructing the History of Incompatible Computing

Lars Brinkhoff and Oscar Vermeulen

ITS Reconstruction Project

MIT Museum Collections Workshop (3rd floor)

(MIT Museum, 314 Main St, Gambrill Center, Cambridge, MA)

April 1, 2024

2:30-4:30 pm

NOTE: The seminar is free and open to all but an MIT ID (or an admission ticket) is required for entry to the museum.

6.1800 in the news

ITS, and the software developed on it, were technically and culturally influential far beyond their core user community. Remote "guest" or "tourist" access was easily available via the early ARPAnet, allowing many interested parties to informally try out features of the operating system and application programs. The wide-open ITS philosophy and collaborative online community were a major influence on the hacker culture, as described in Steven Levy's book *Hackers*,[3] and were the direct forerunners of the free and open-source software, open-design, and Wiki movements.

https://en.wikipedia.org/wiki/Incompatible_Timesharing_System

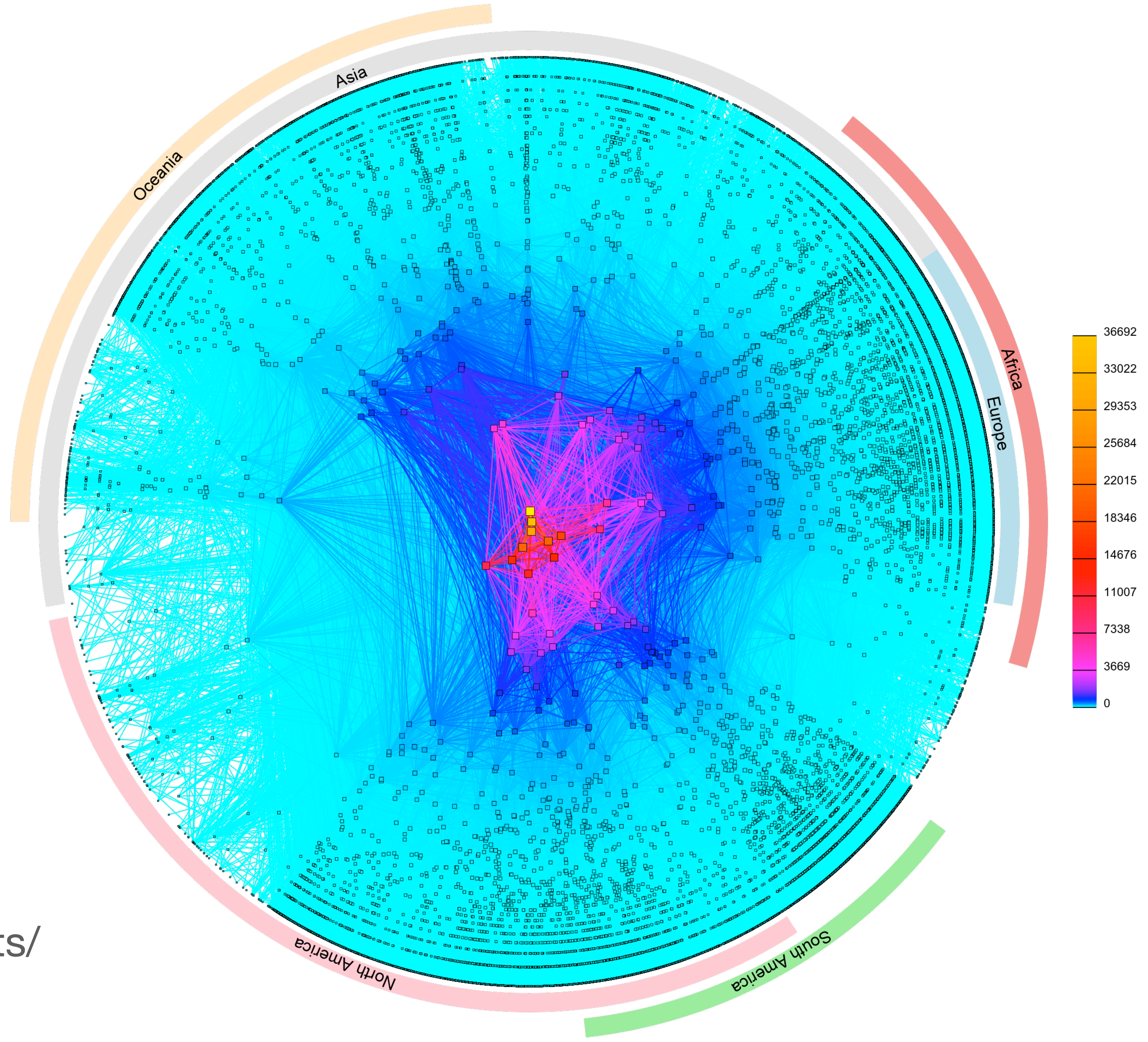
1970s: ARPAnet — hosts.txt

1978: flexibility and layering — distance-vector routing

early 80s: growth → change — **TCP**, UDP; OSPF, EGP, DNS

late 80s: growth → problems — congestion collapse (which led to **congestion control**); policy routing; CIDR

1993: commercialization



CAIDA's IPv4 AS Core, January 2020 (https://www.caida.org/projects/cartography/as-core/2020/)

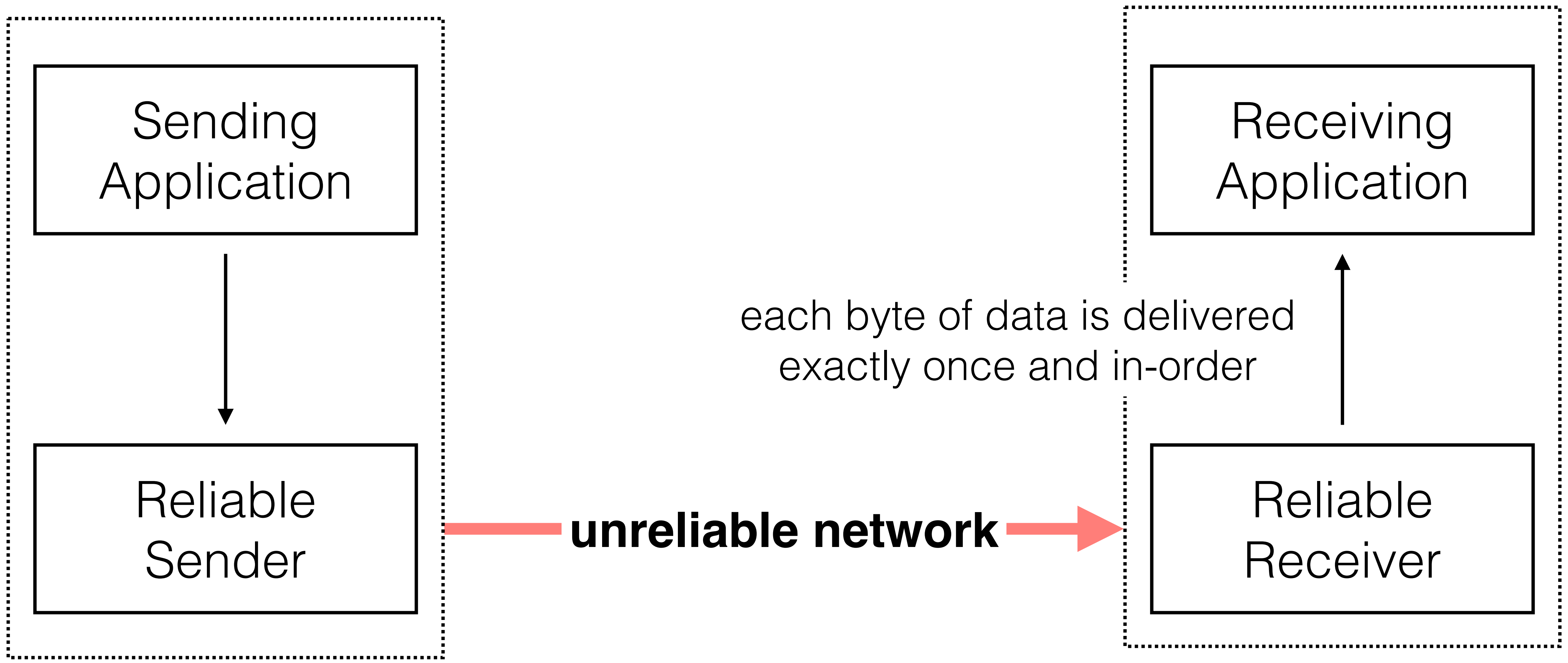
application	the things that actually generate traffic
transport	sharing the network, reliability (or not) *examples: TCP, UDP*
network	naming, addressing, routing *examples: IP*
link	communication between two directly-connected nodes *examples: ethernet, bluetooth, 802.11 (wifi)*

today: moving up to the transport layer to discuss **reliable transport**

our (first) goal today is to create a **reliable transport protocol**, which delivers each byte of data **exactly once, in-order**, to the receiving application

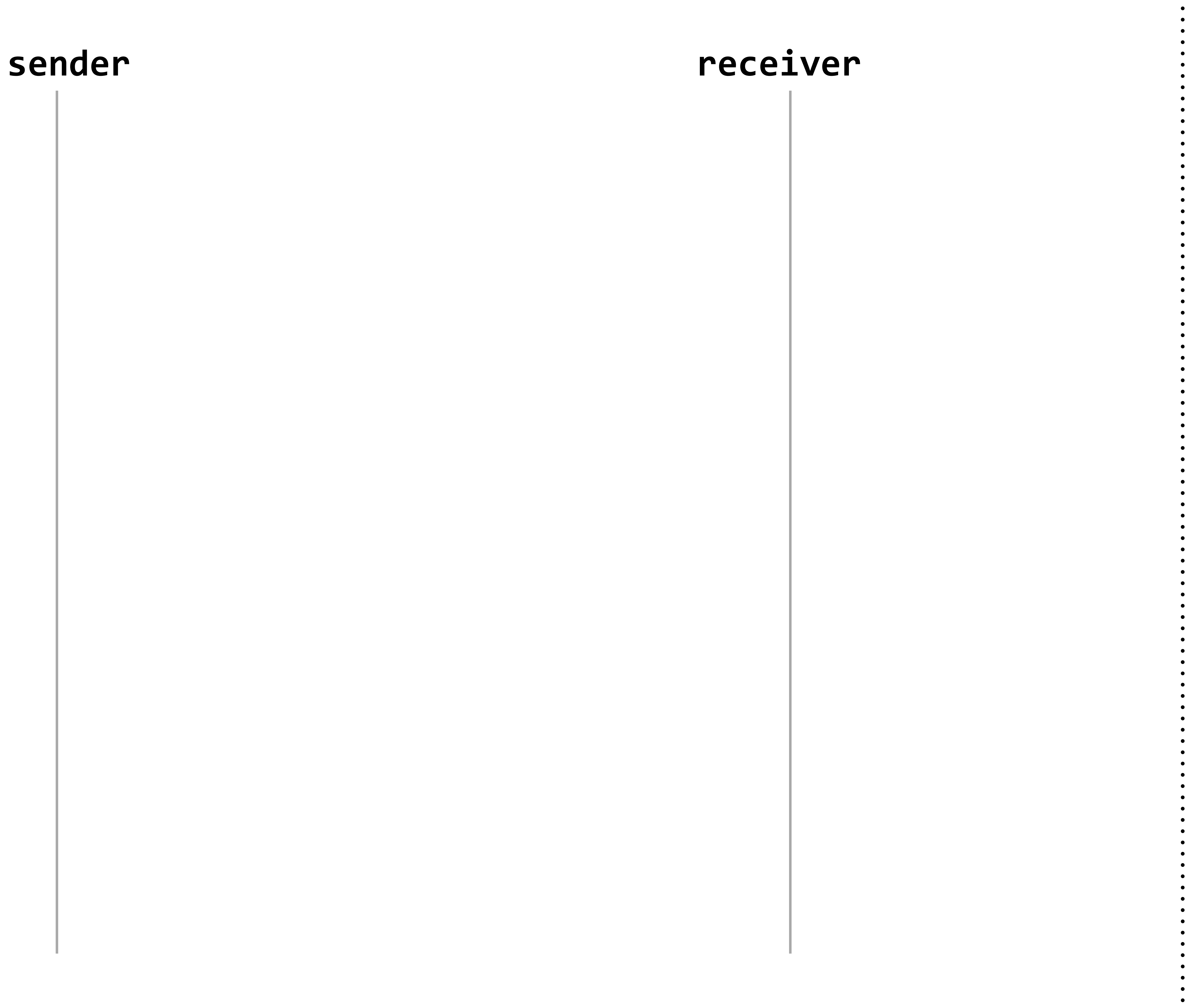
`application`	`the things that actually generate traffic`
`transport`	`sharing the network, reliability (or not)` *examples: TCP, UDP*
`network`	`naming, addressing, routing` *examples: IP*
`link`	`communication between two directly-connected nodes` *examples: ethernet, bluetooth, 802.11 (wifi)*

our (first) goal today is to create a **reliable transport protocol**, which delivers each byte of data **exactly once, in-order**, to the receiving application

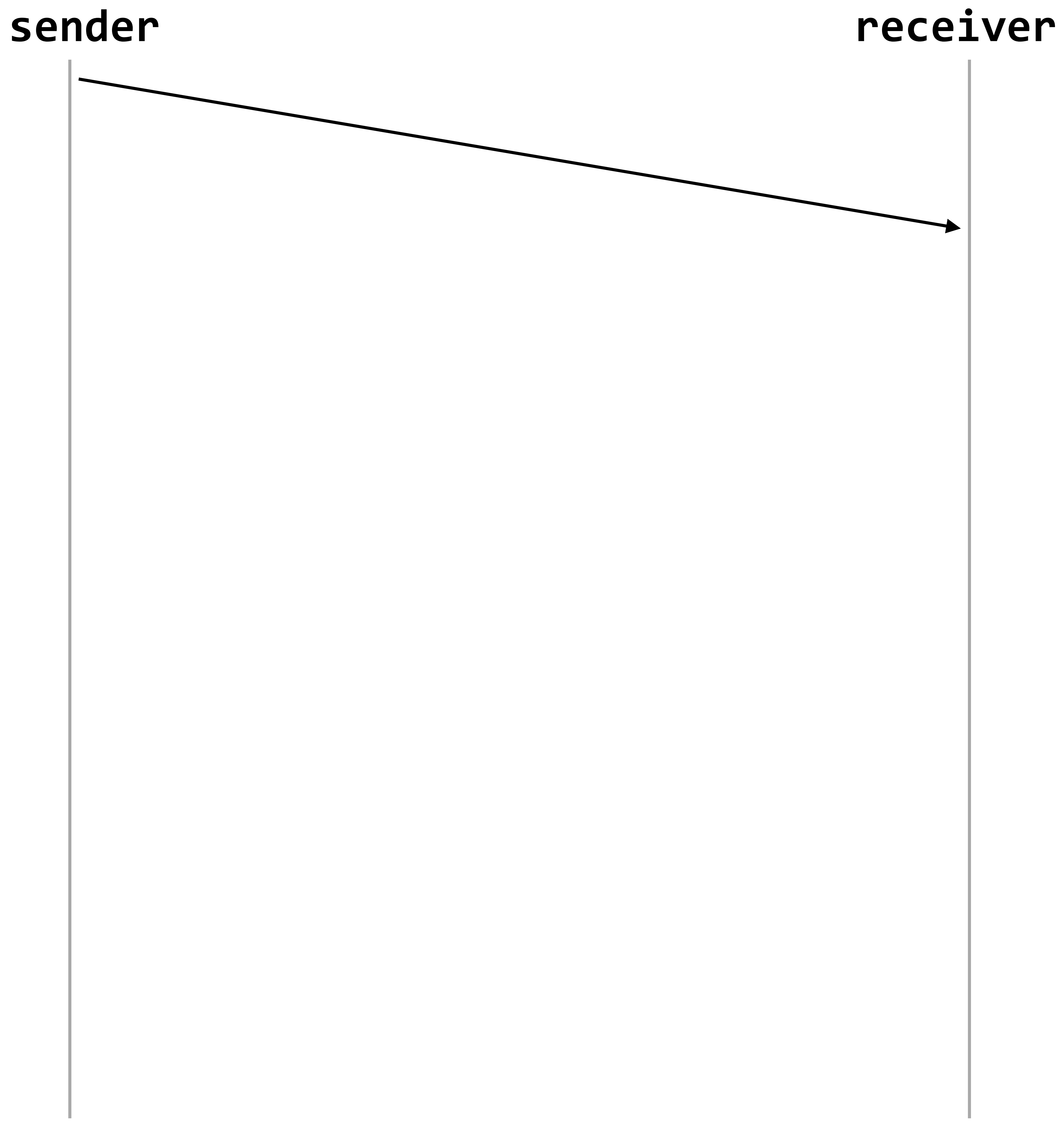


`application`	`the things that actually generate traffic`
`transport`	`sharing the network, reliability (or not)` *examples: TCP, UDP*
`network`	`naming, addressing, routing` *examples: IP*
`link`	`communication between two directly-connected nodes` *examples: ethernet, bluetooth, 802.11 (wifi)*

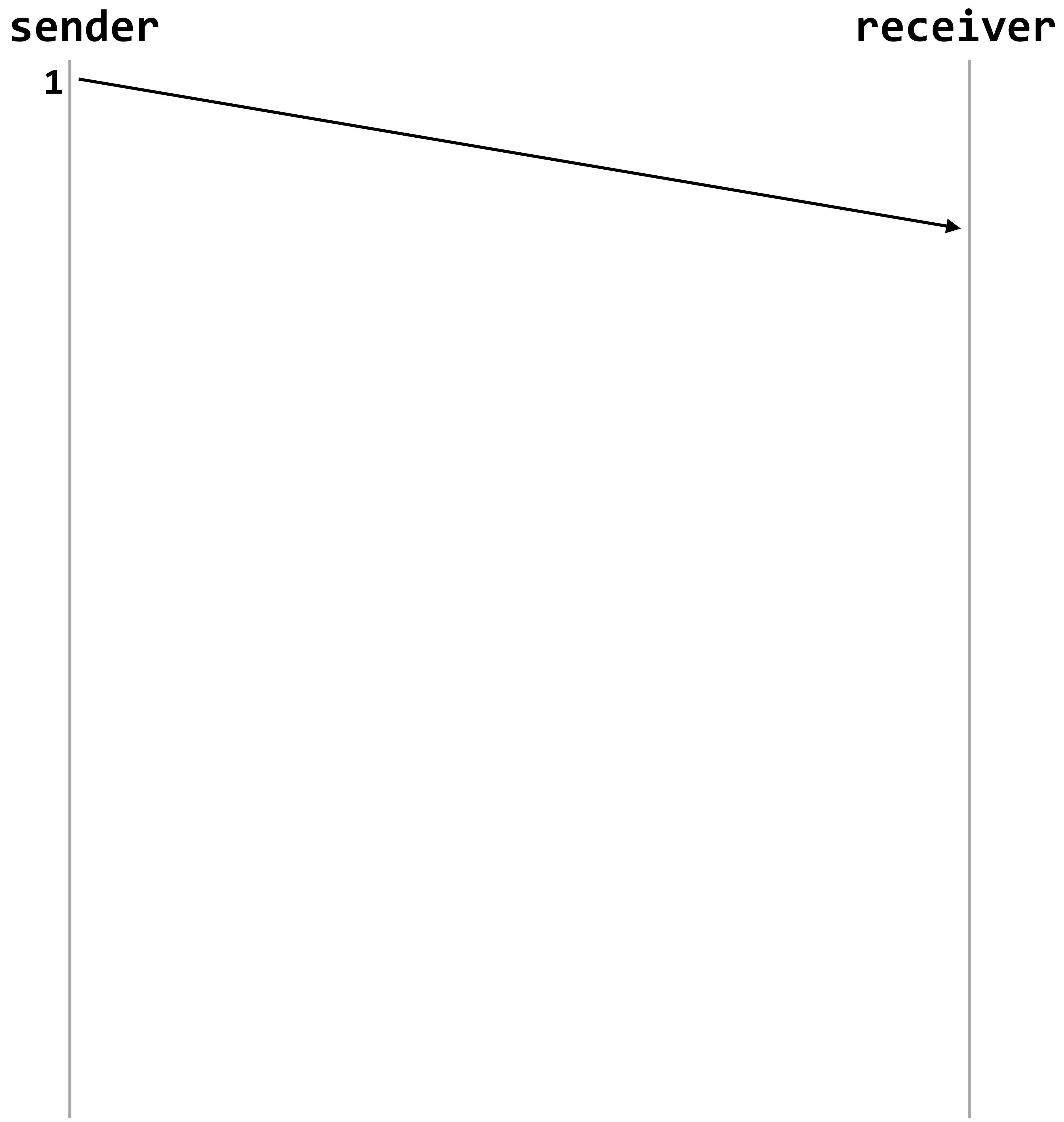
reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

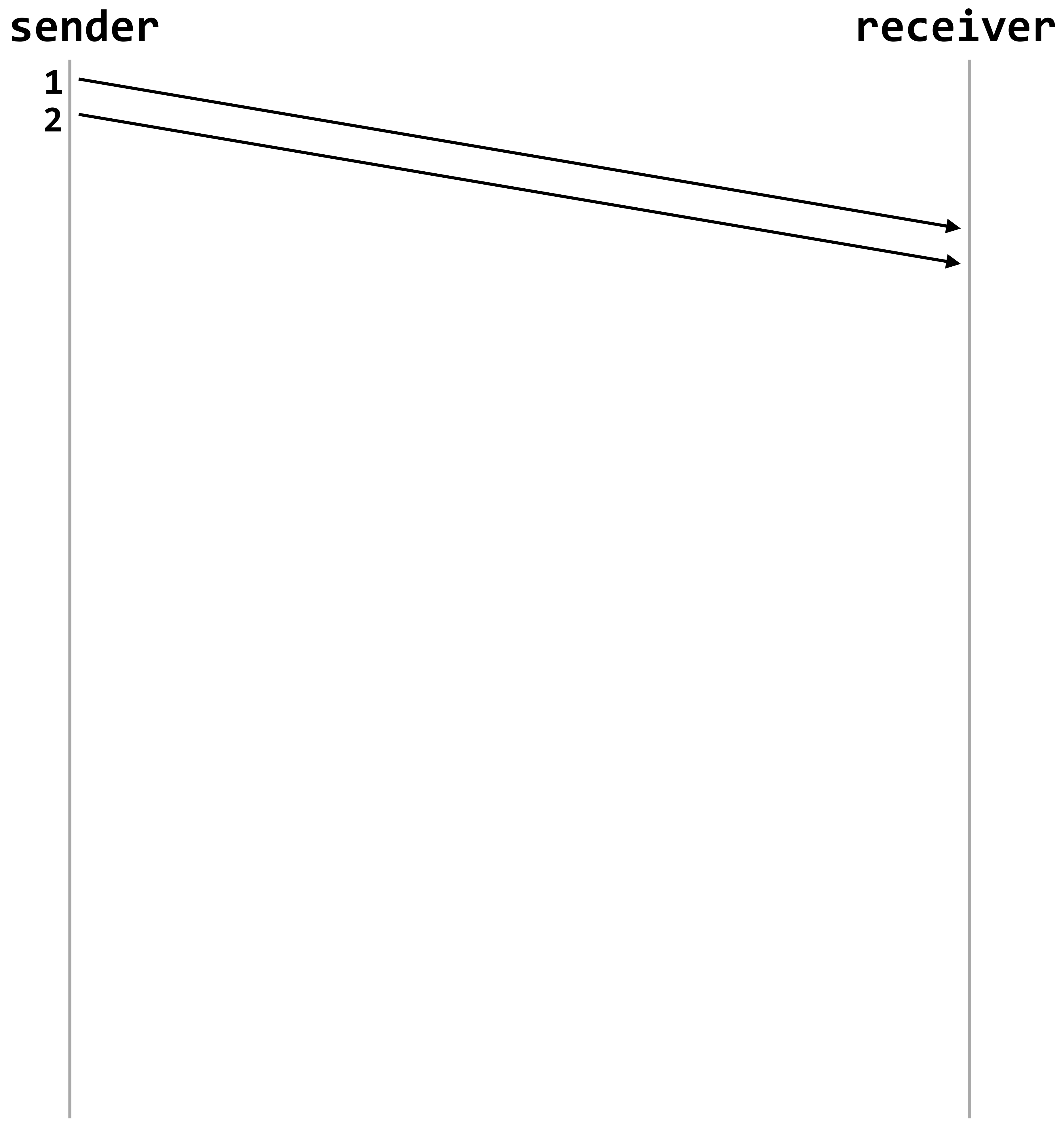


reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



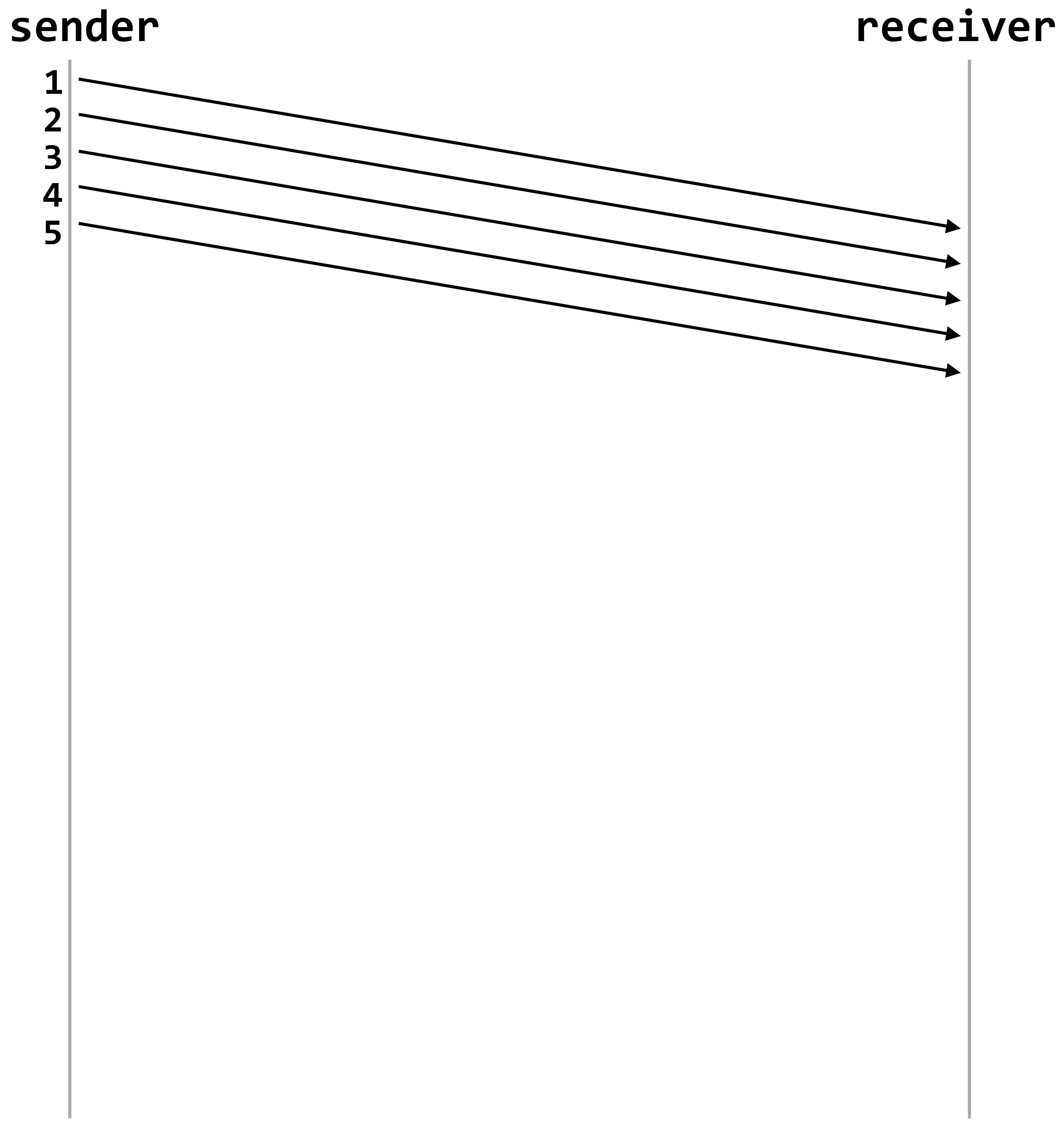
sequence numbers: used to order the packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



sequence numbers: used to order the packets

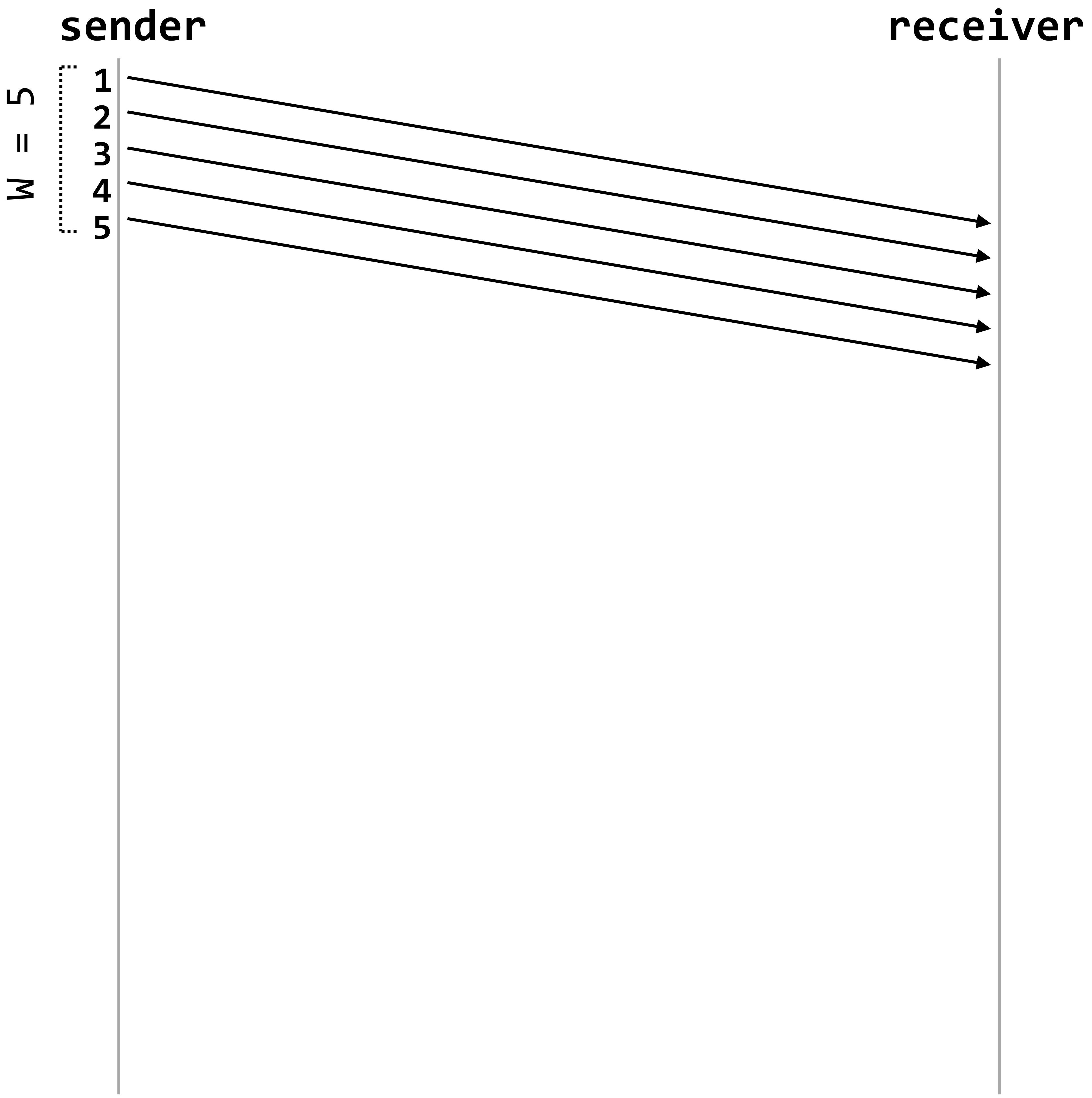
reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



sequence numbers: used to order the packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

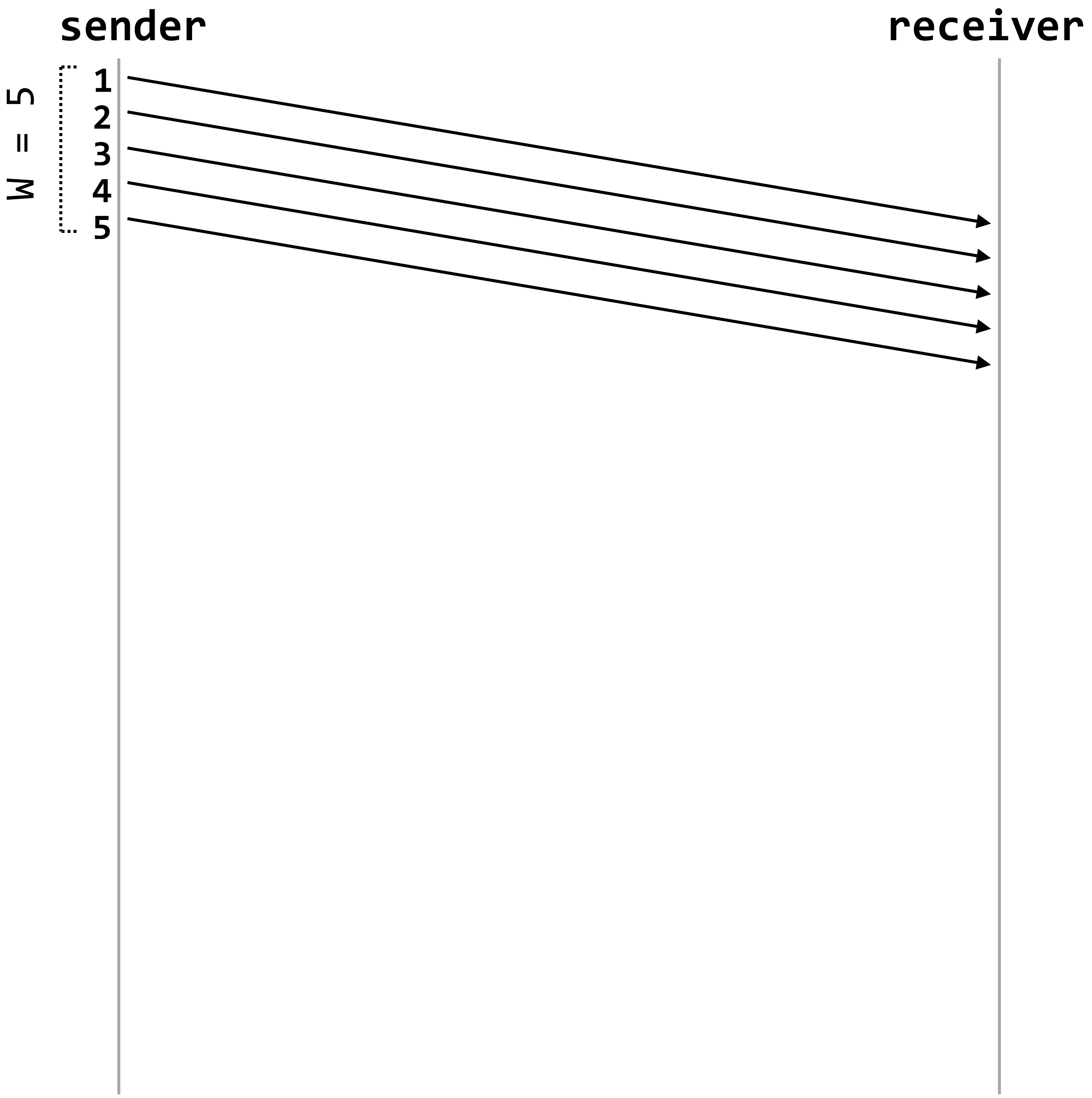
the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



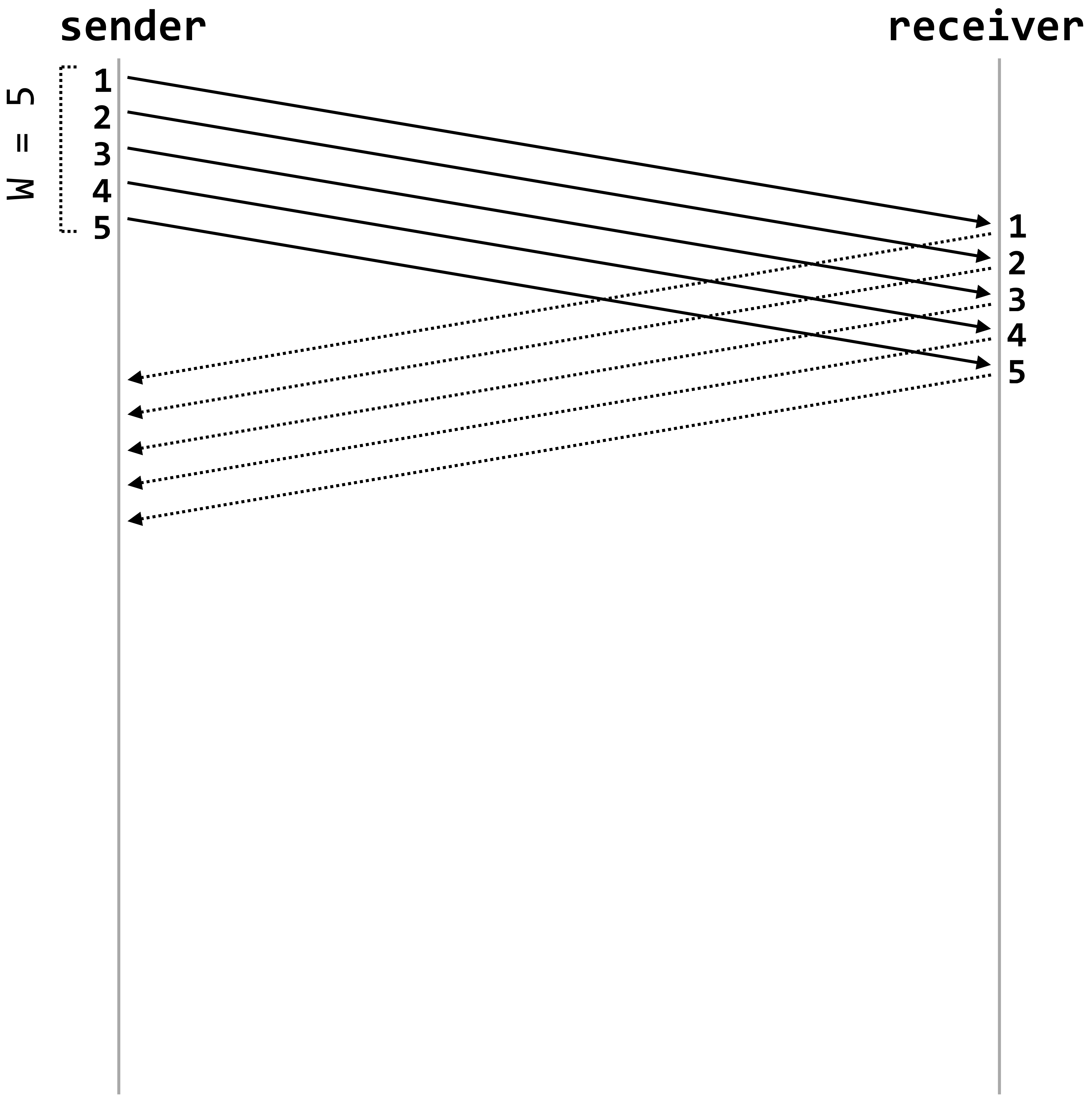
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



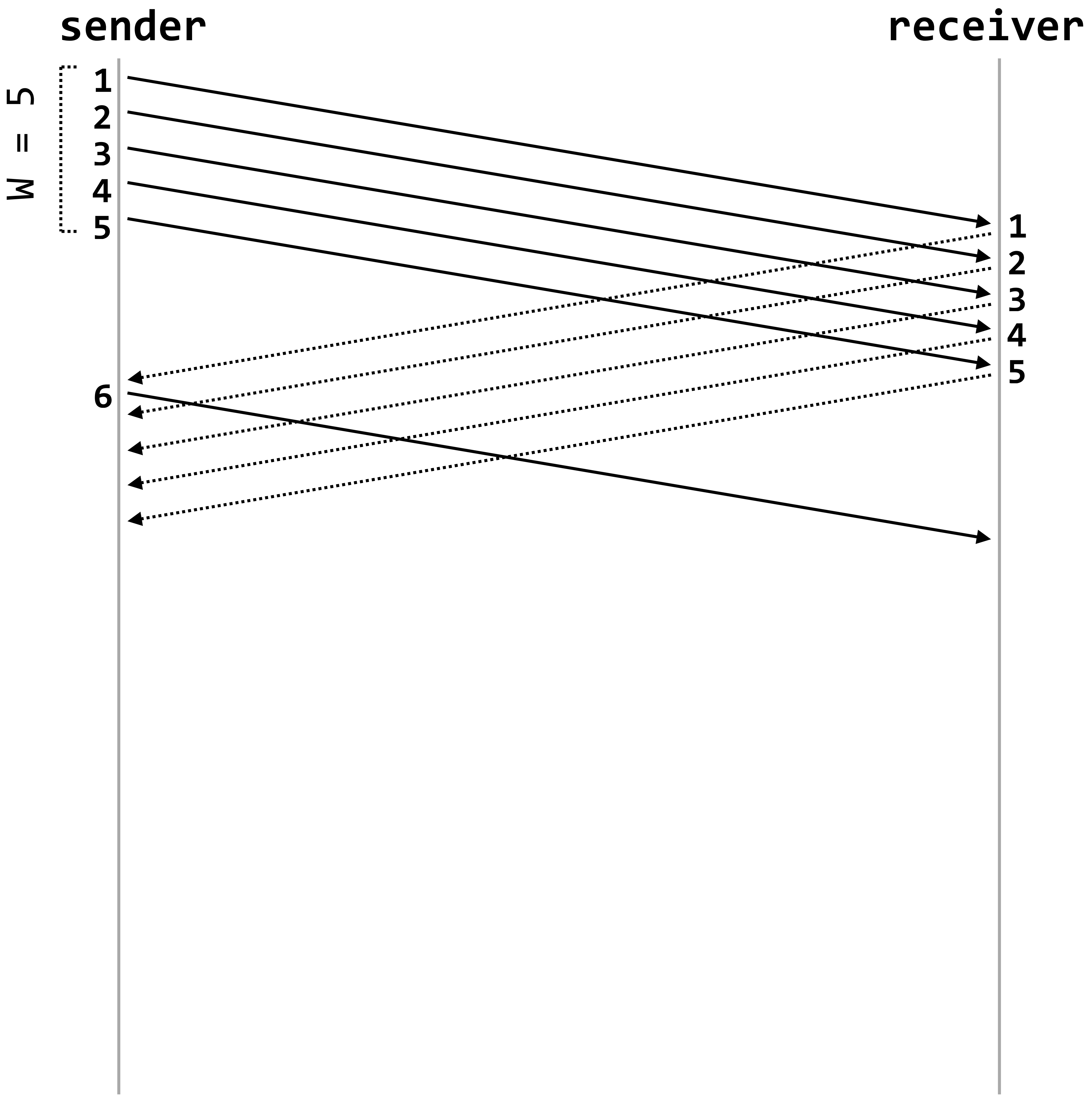
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



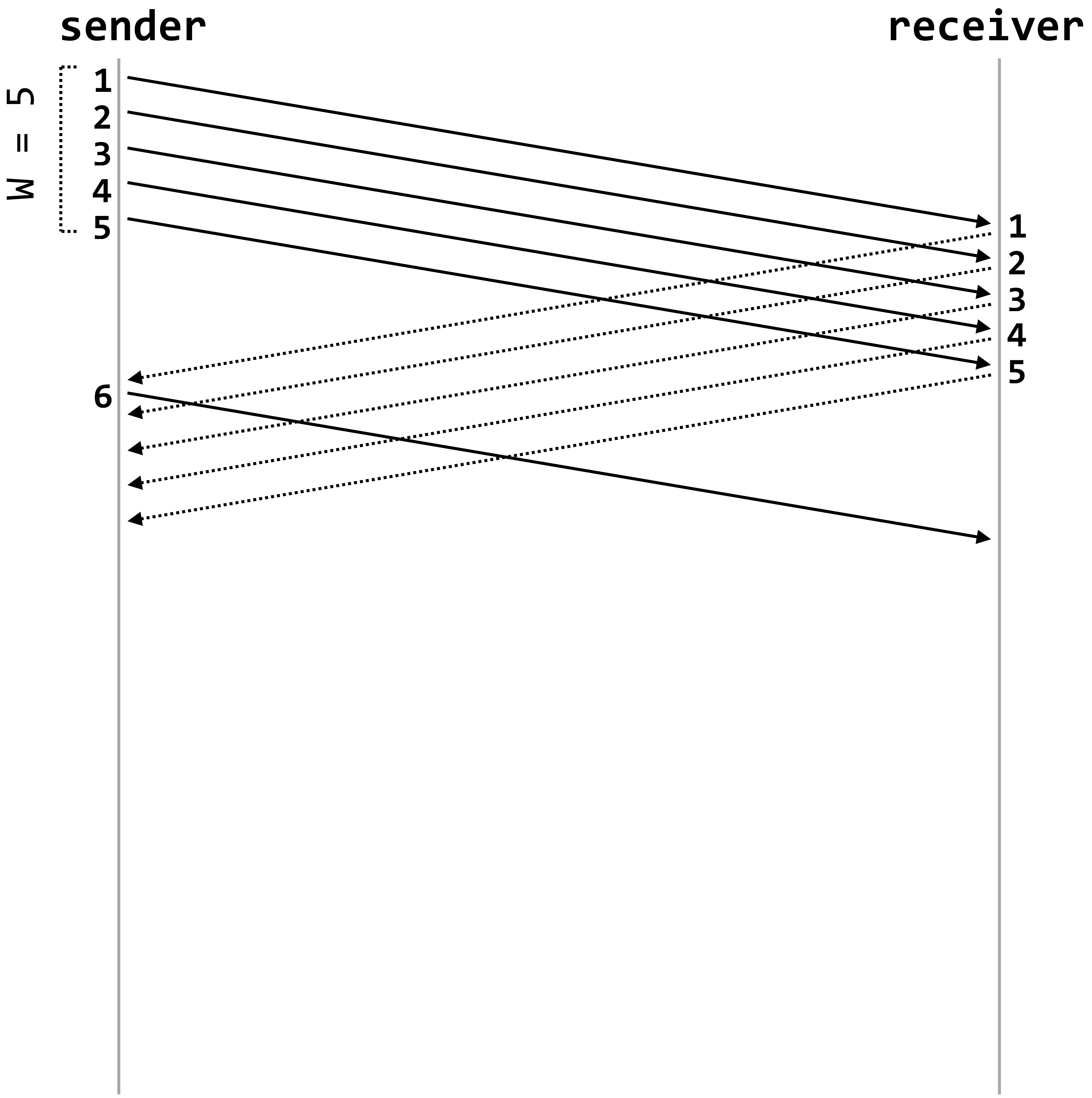
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

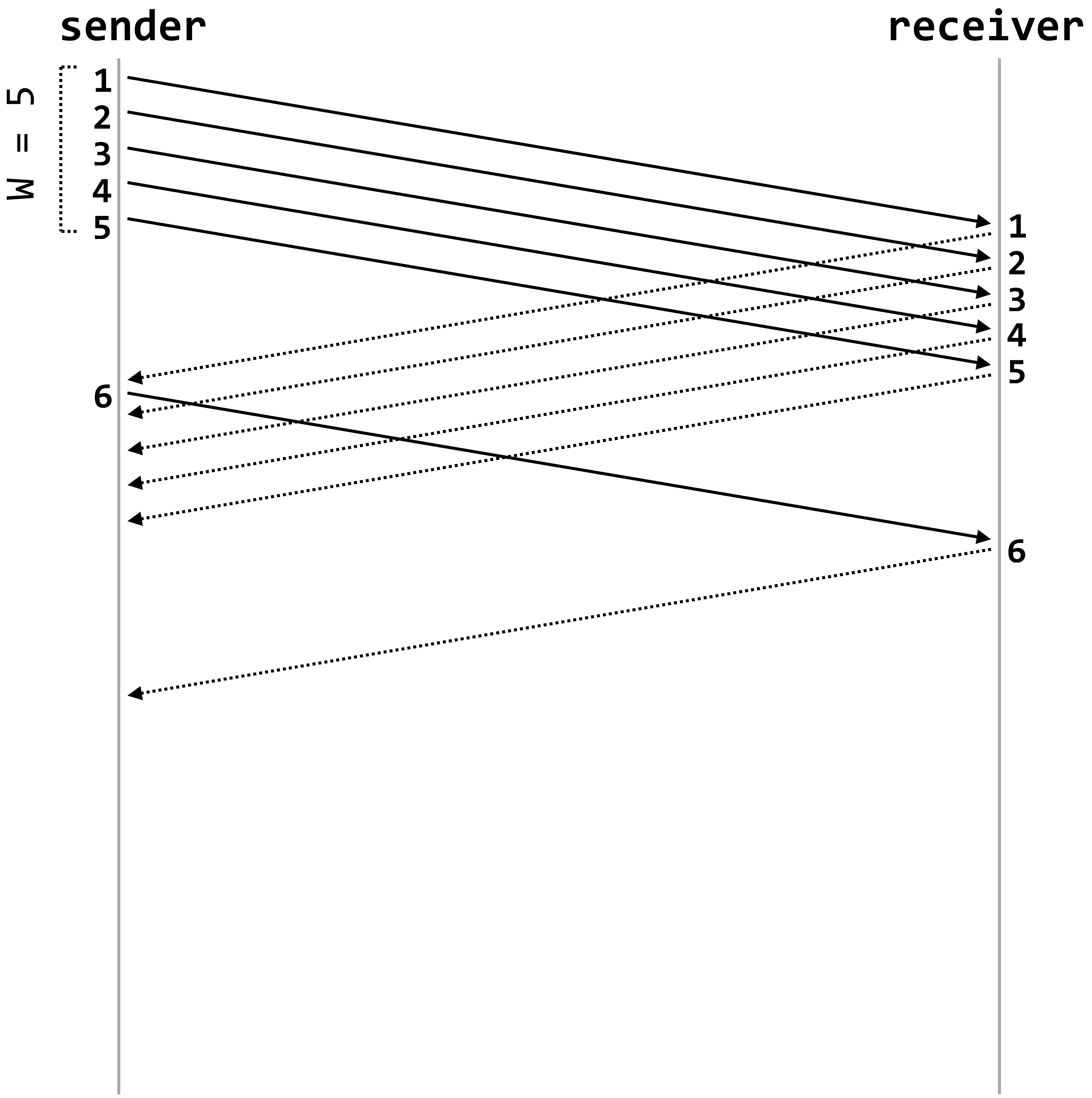
an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

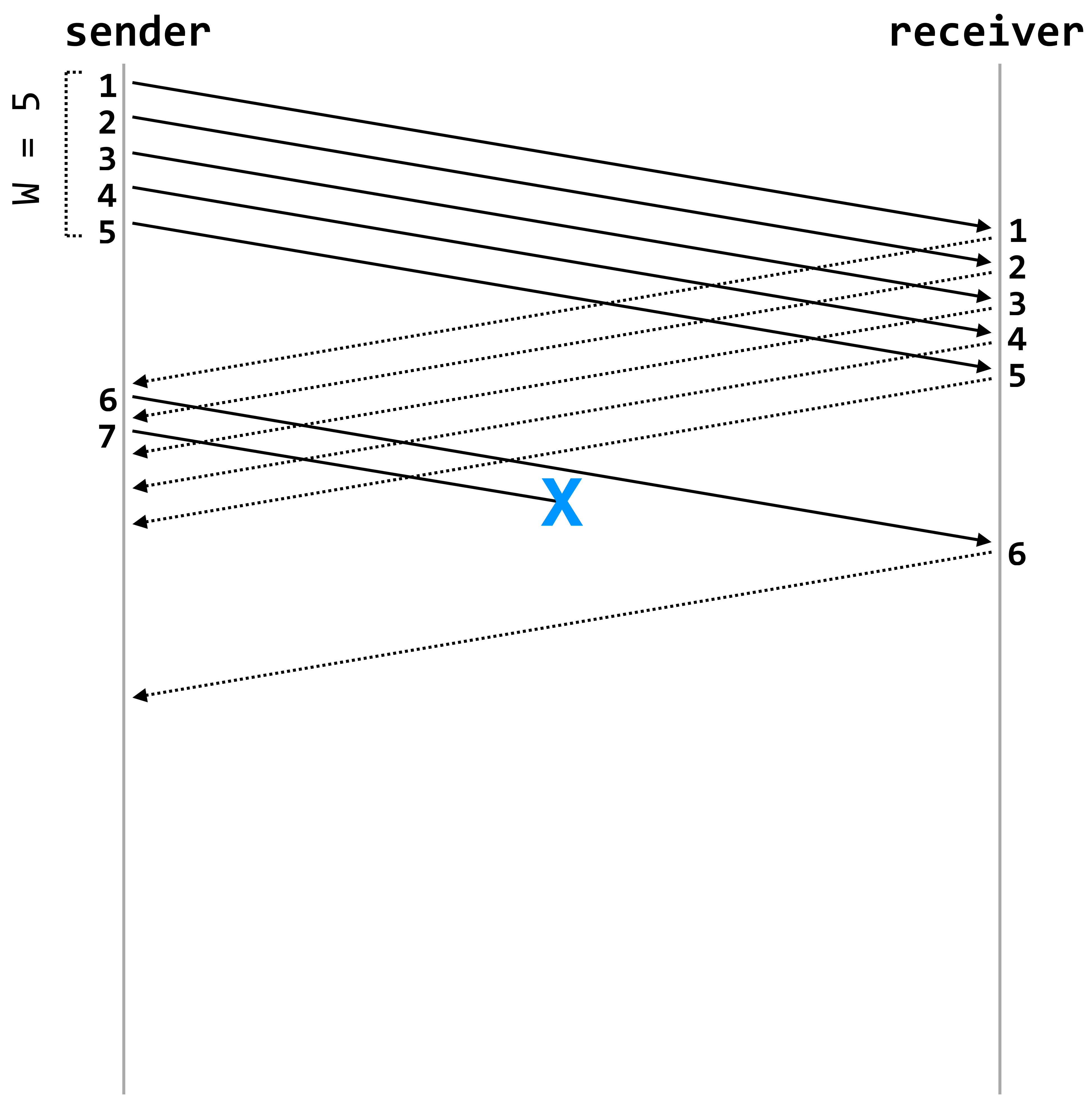
an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

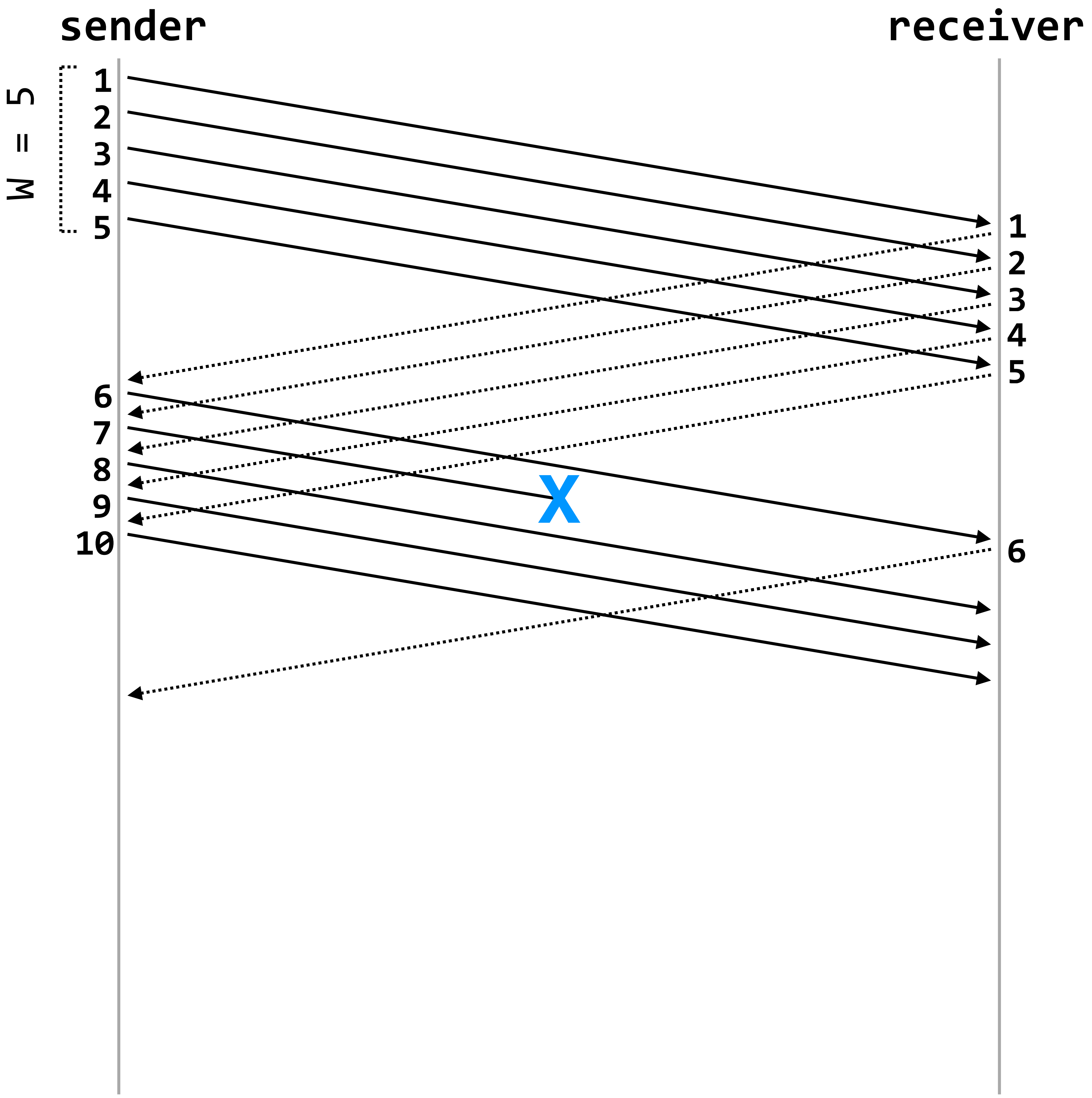
an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

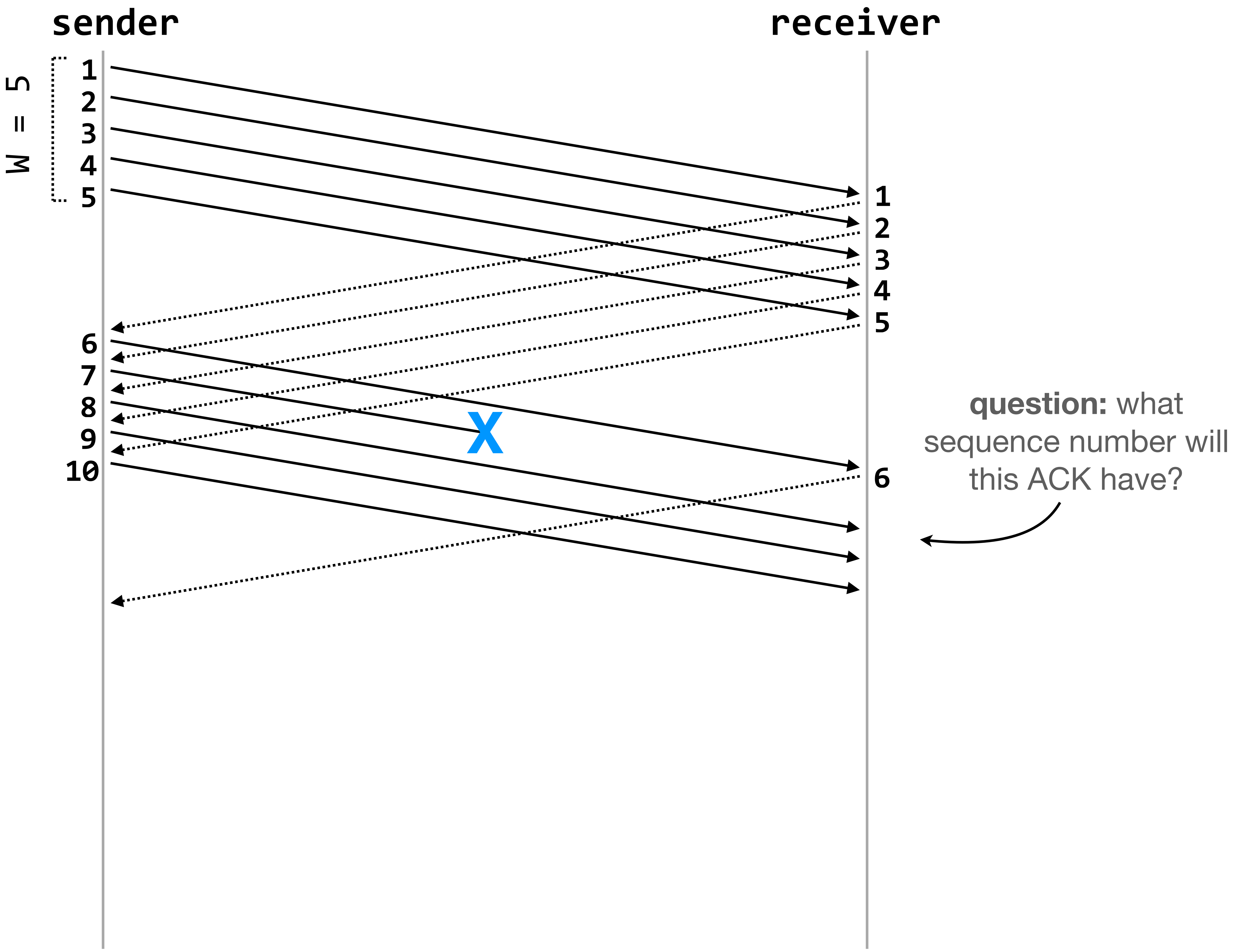
an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

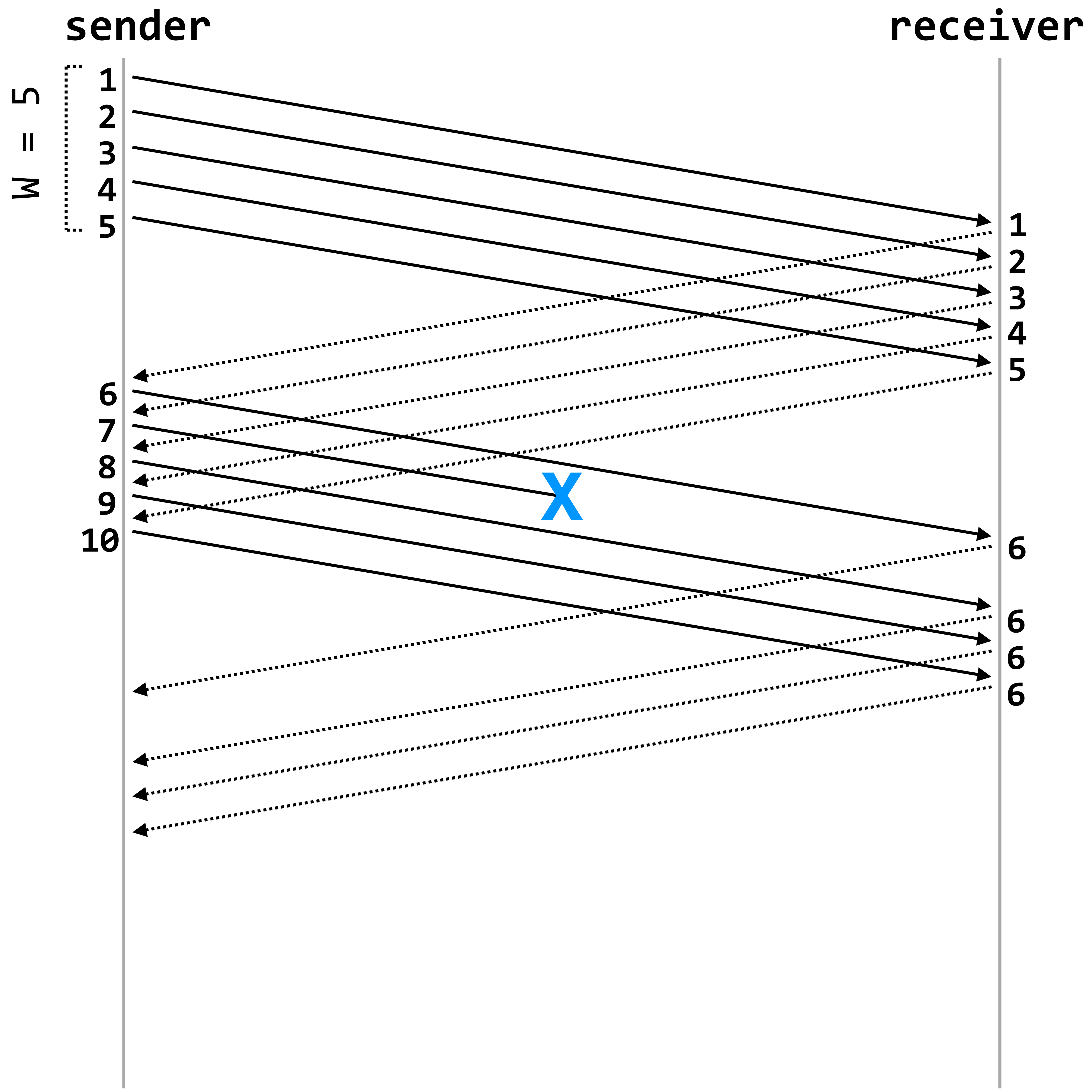
this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

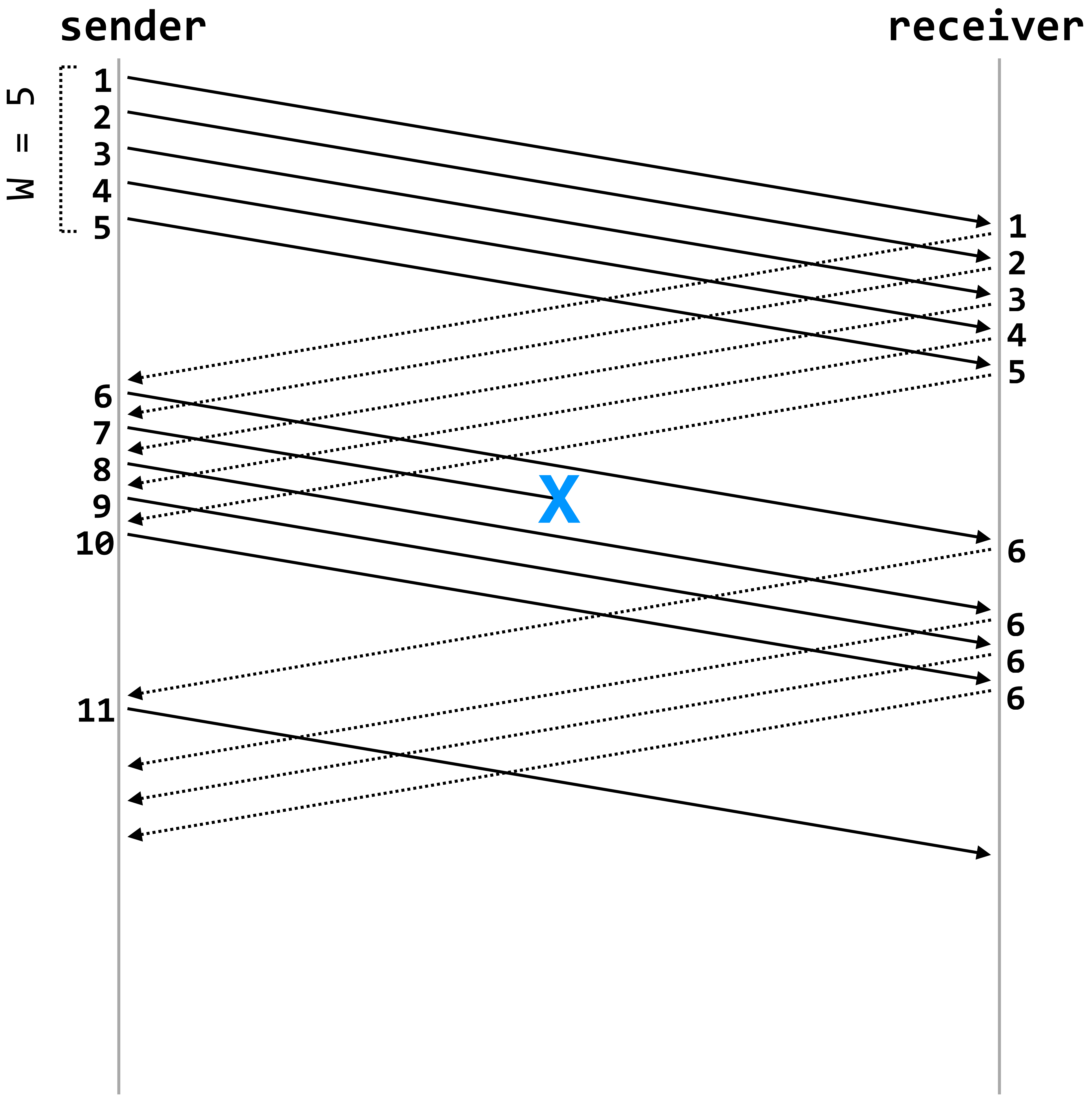
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

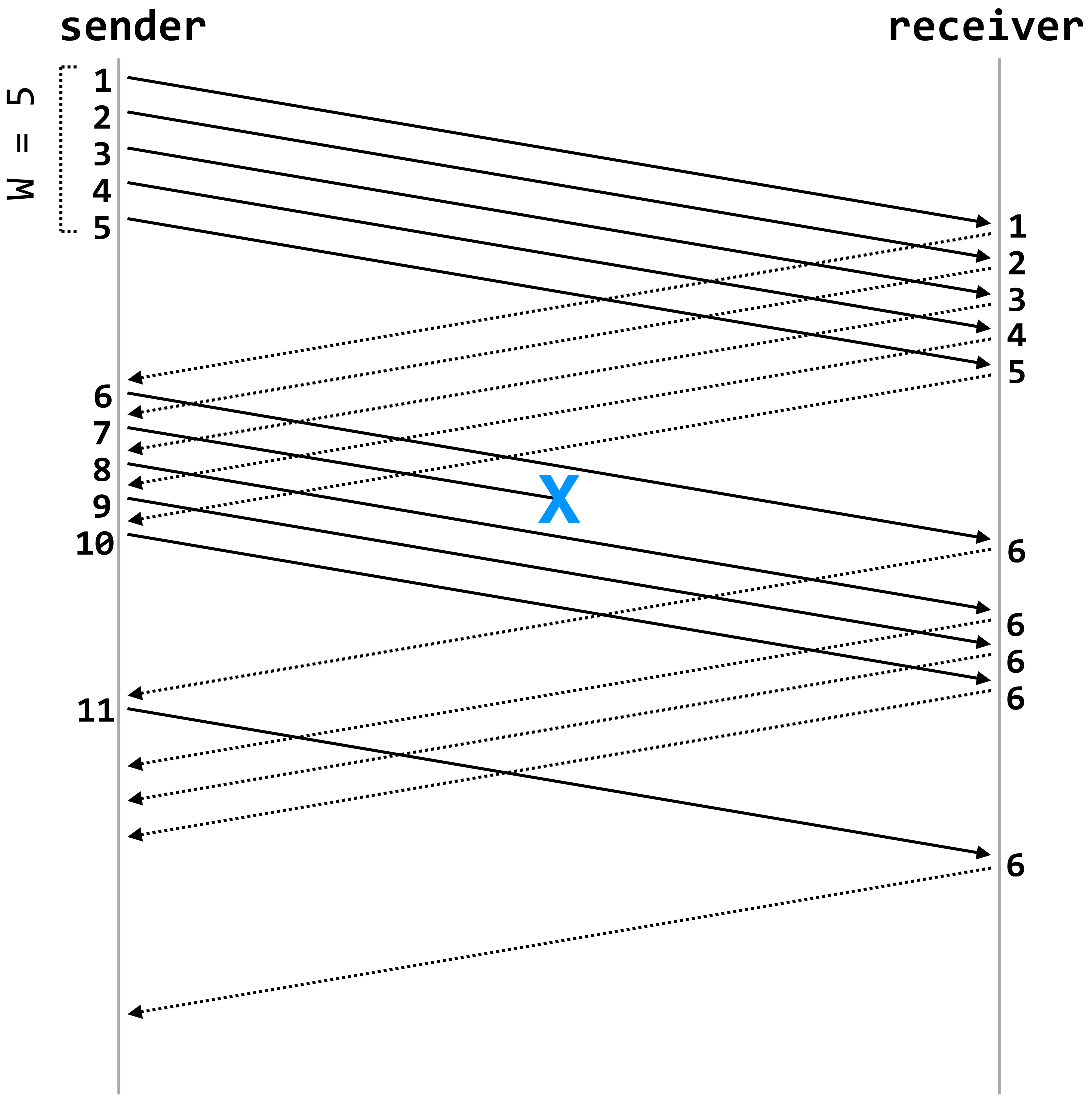
an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

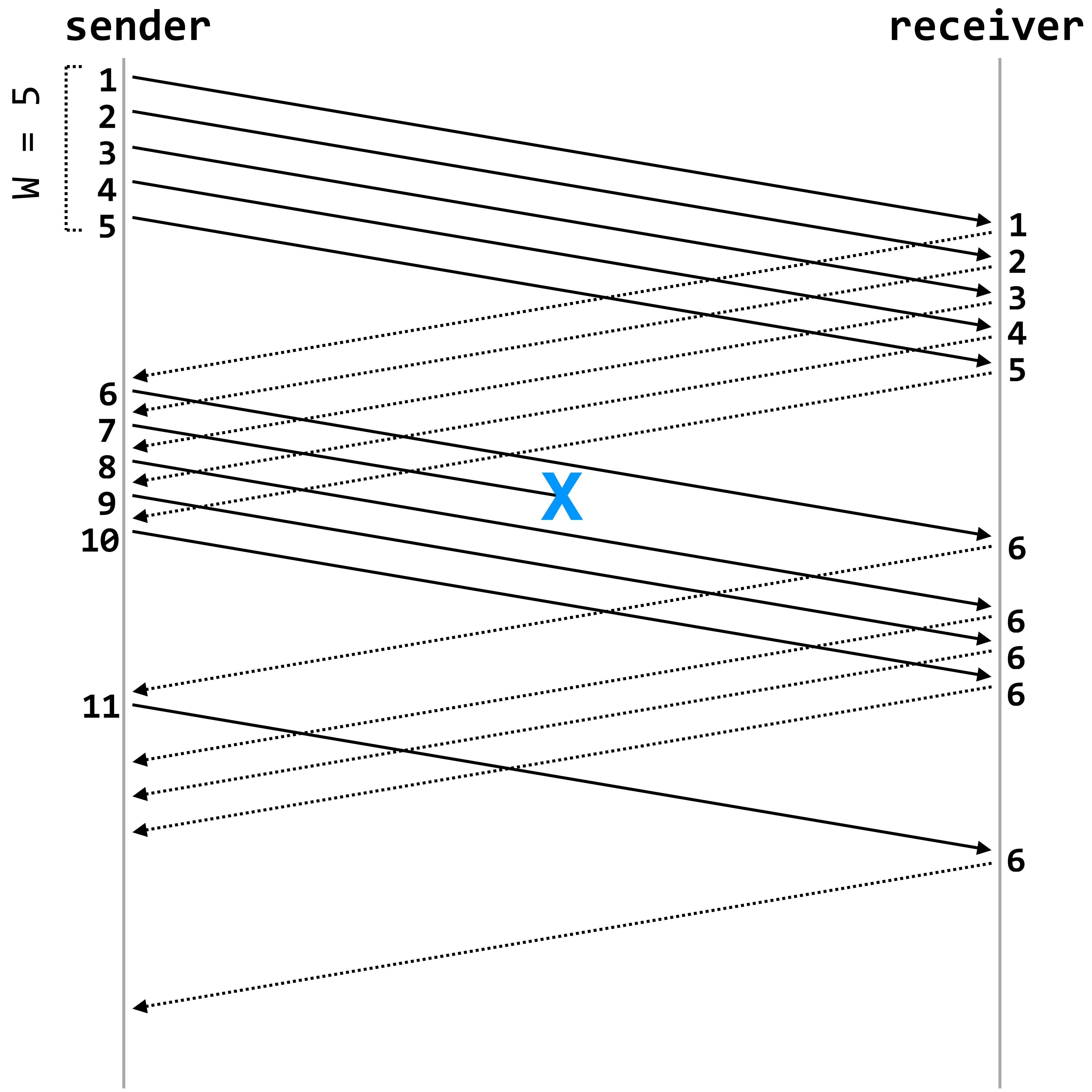
an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

question: can the sender infer that packet 7 has been lost?

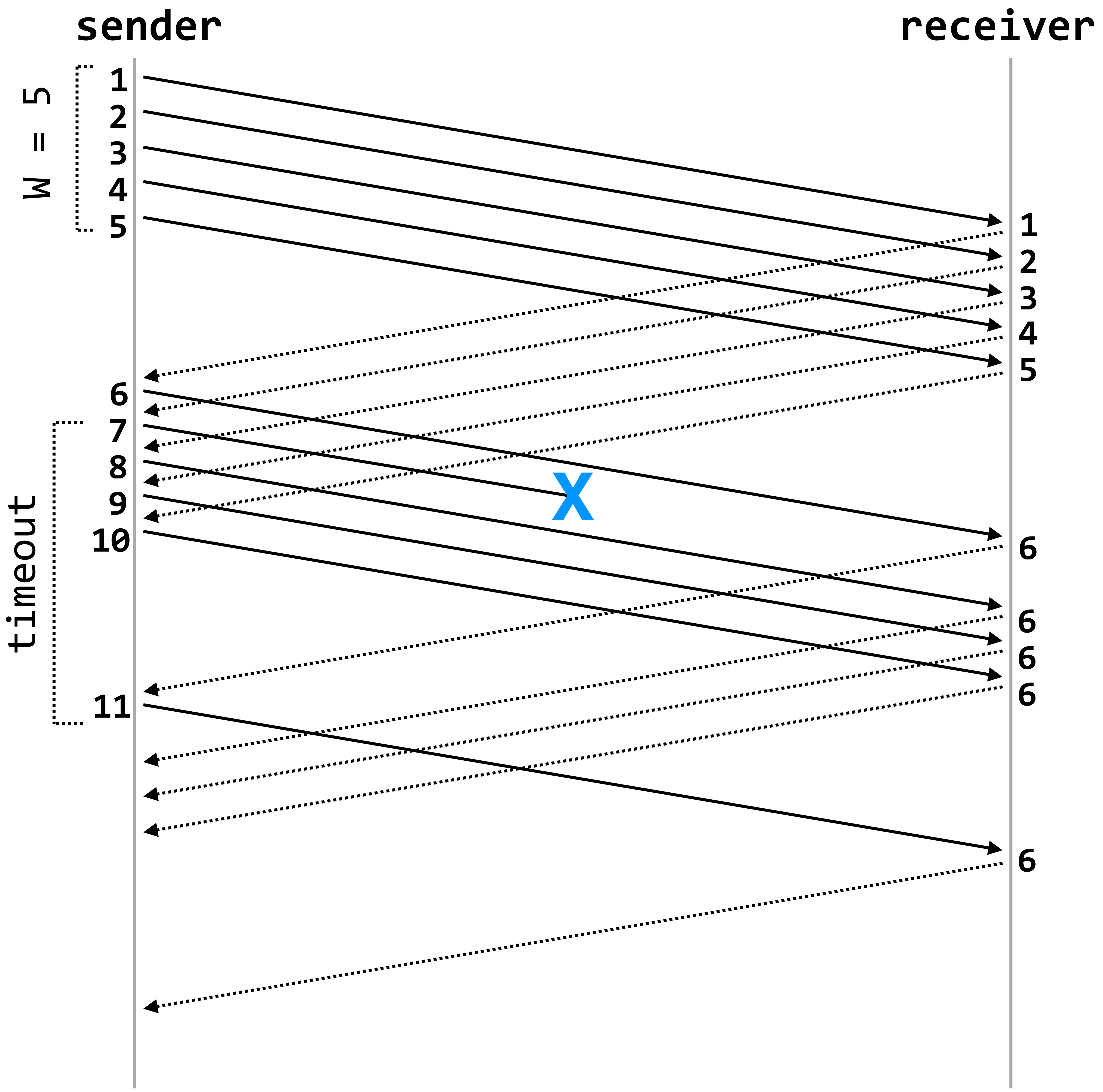
this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

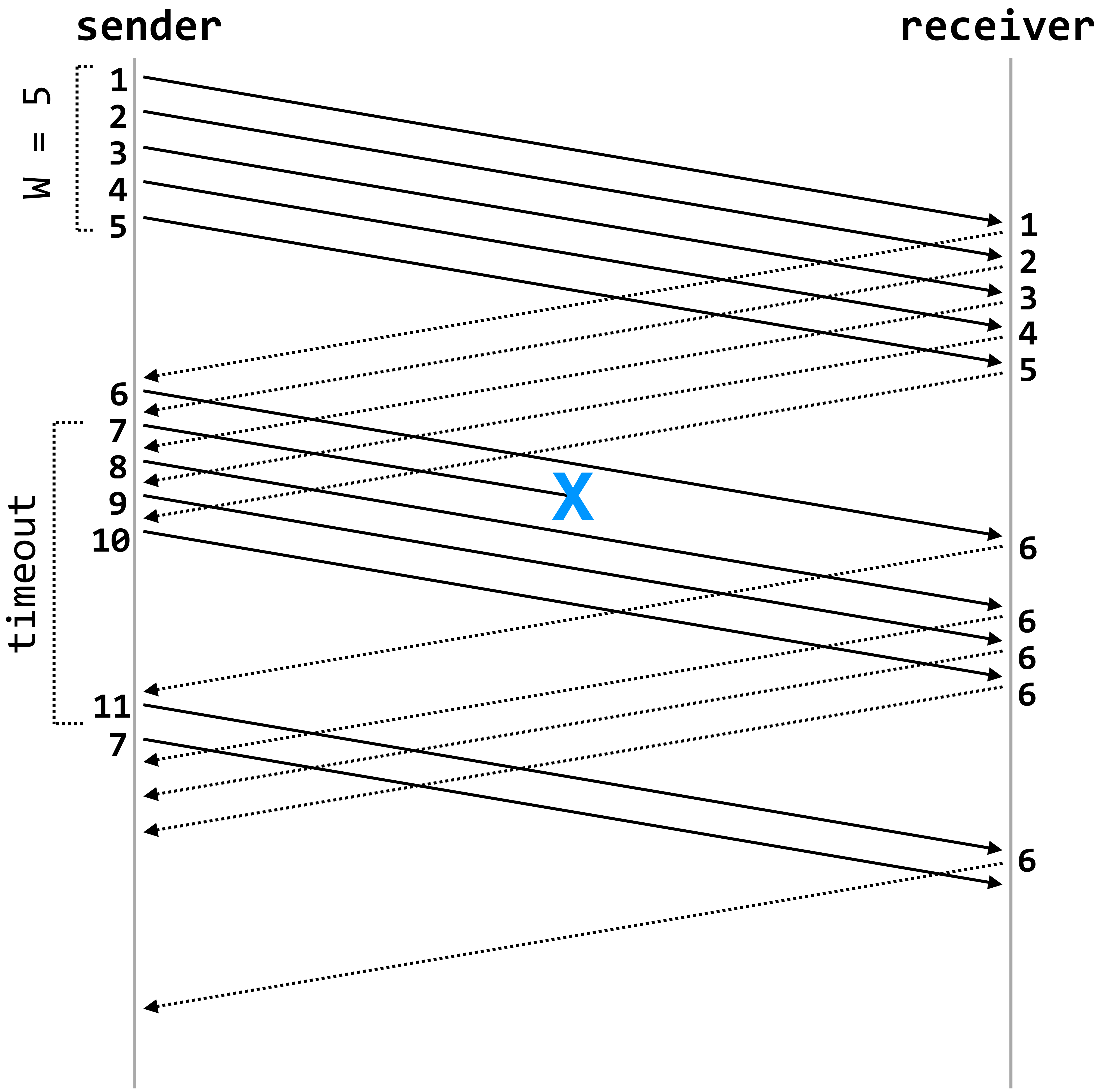
timeouts: used to retransmit packets

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

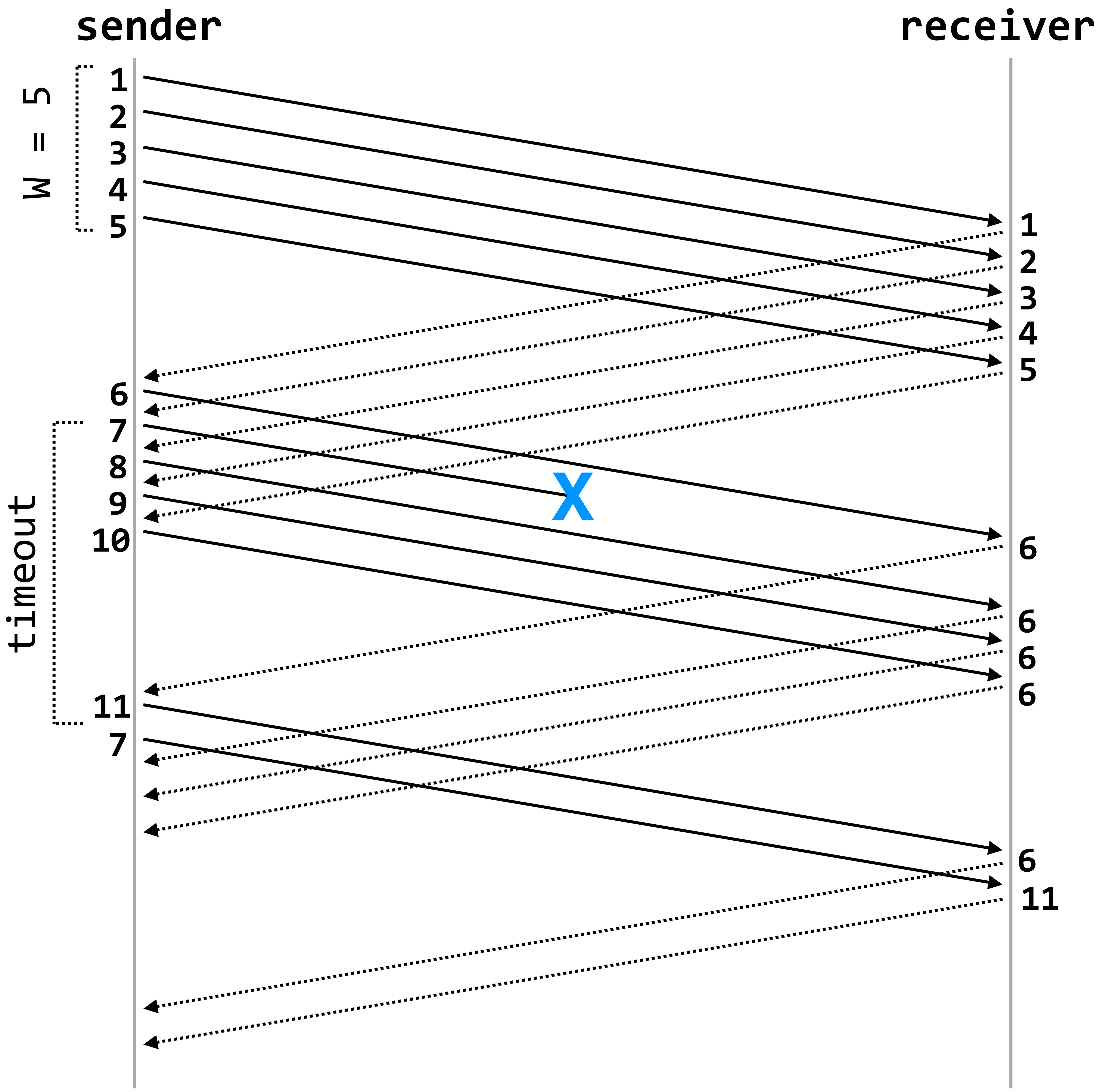
timeouts: used to retransmit packets

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

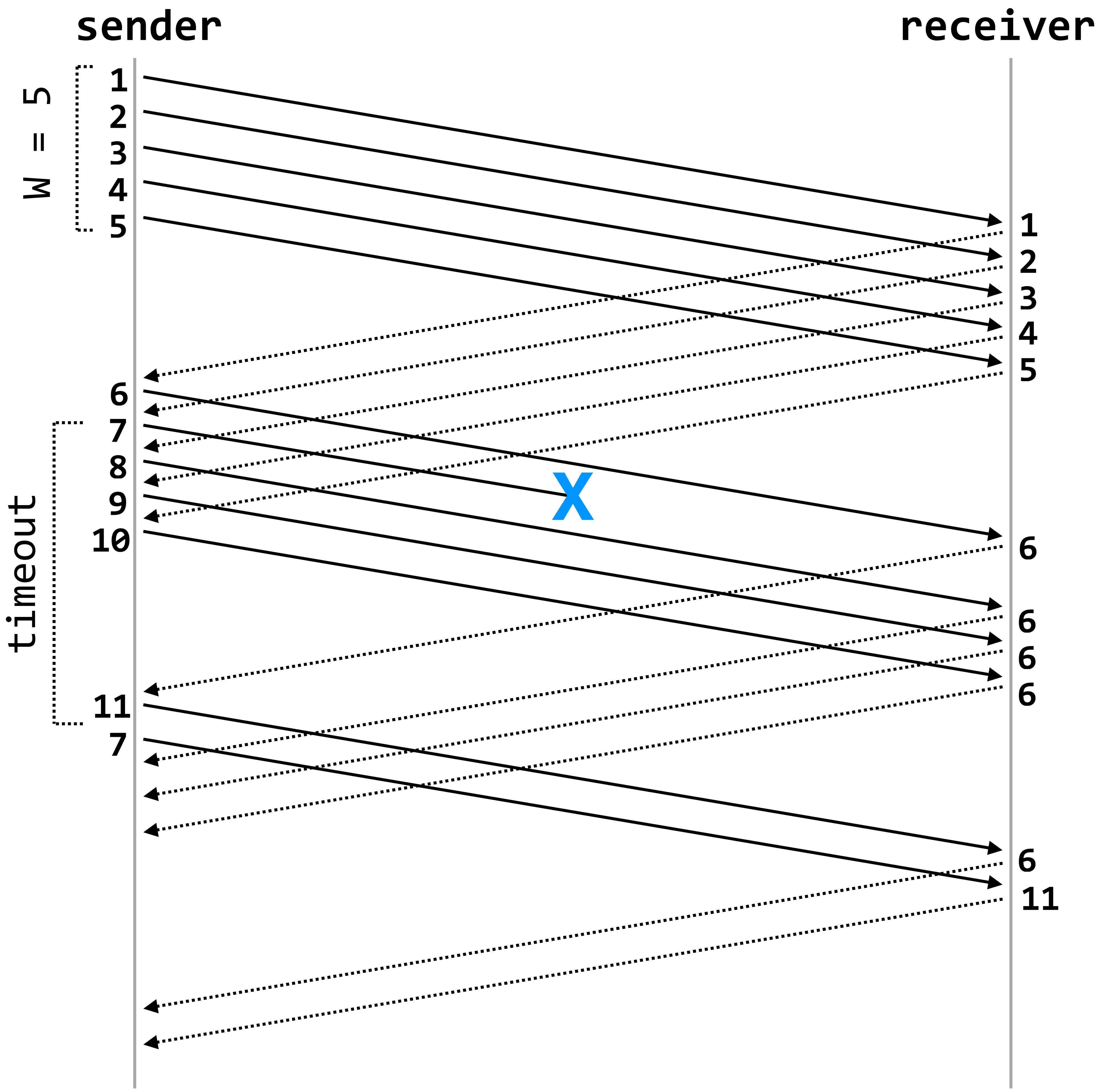
timeouts: used to retransmit packets

this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

the sender is allowed to have W outstanding packets at once, but no more



this is known as a **sliding-window protocol**

the **window** of outstanding (un-ACKed) packets **slides** along the sequence number space

sequence numbers: used to order the packets

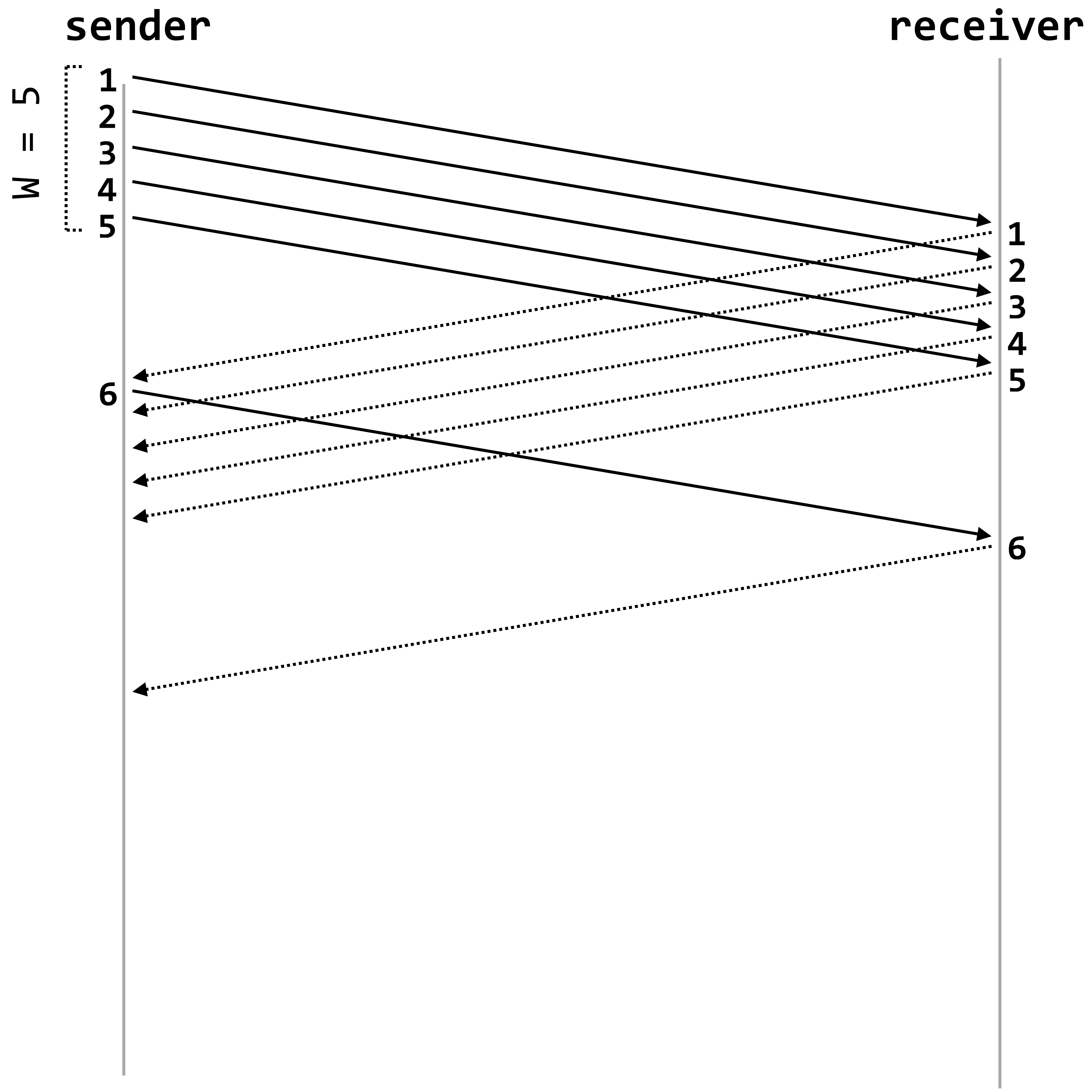
acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

note that the sender could also infer loss because it has received multiple ACKs with sequence number 6, but none with sequence number > 7; we'll come back to that

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



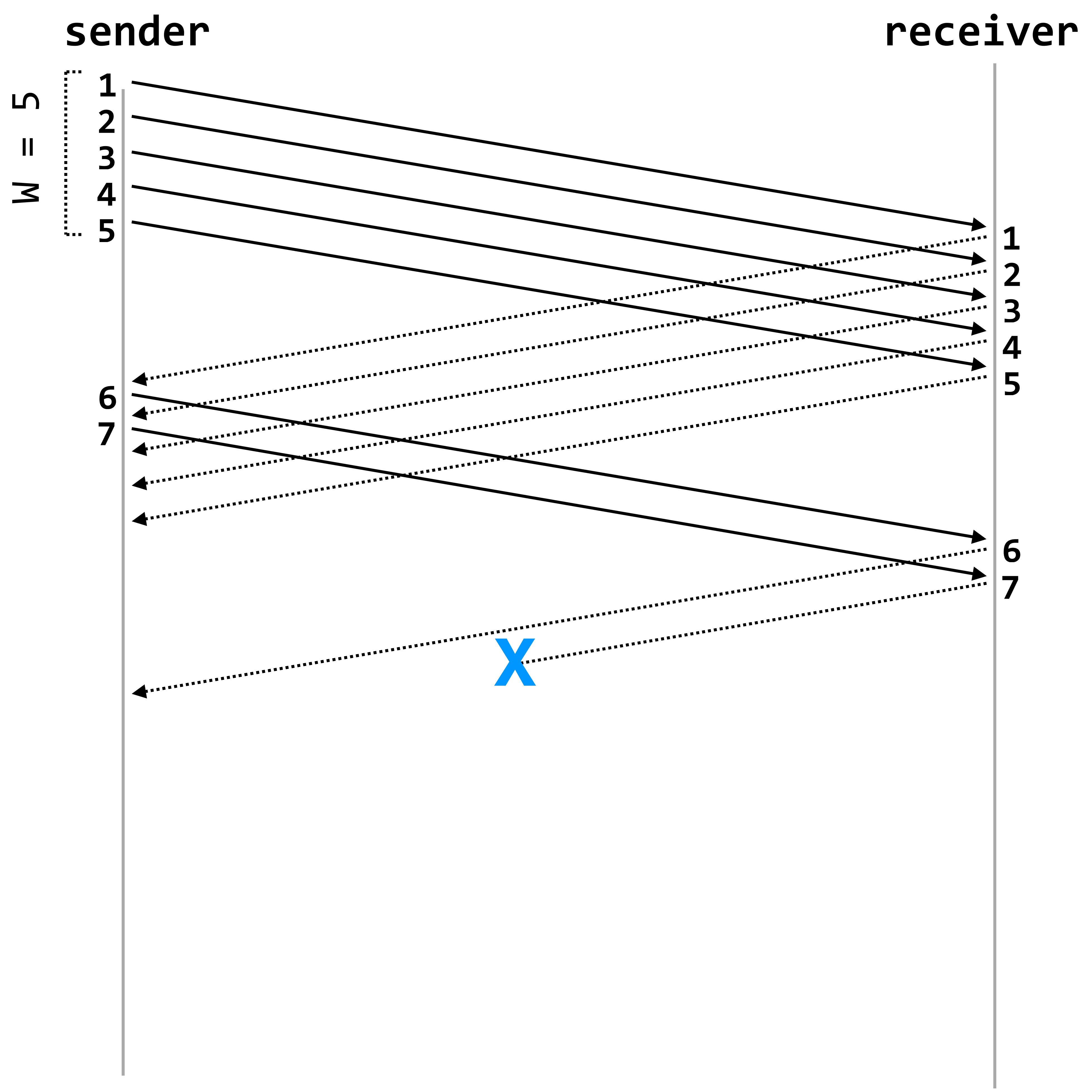
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



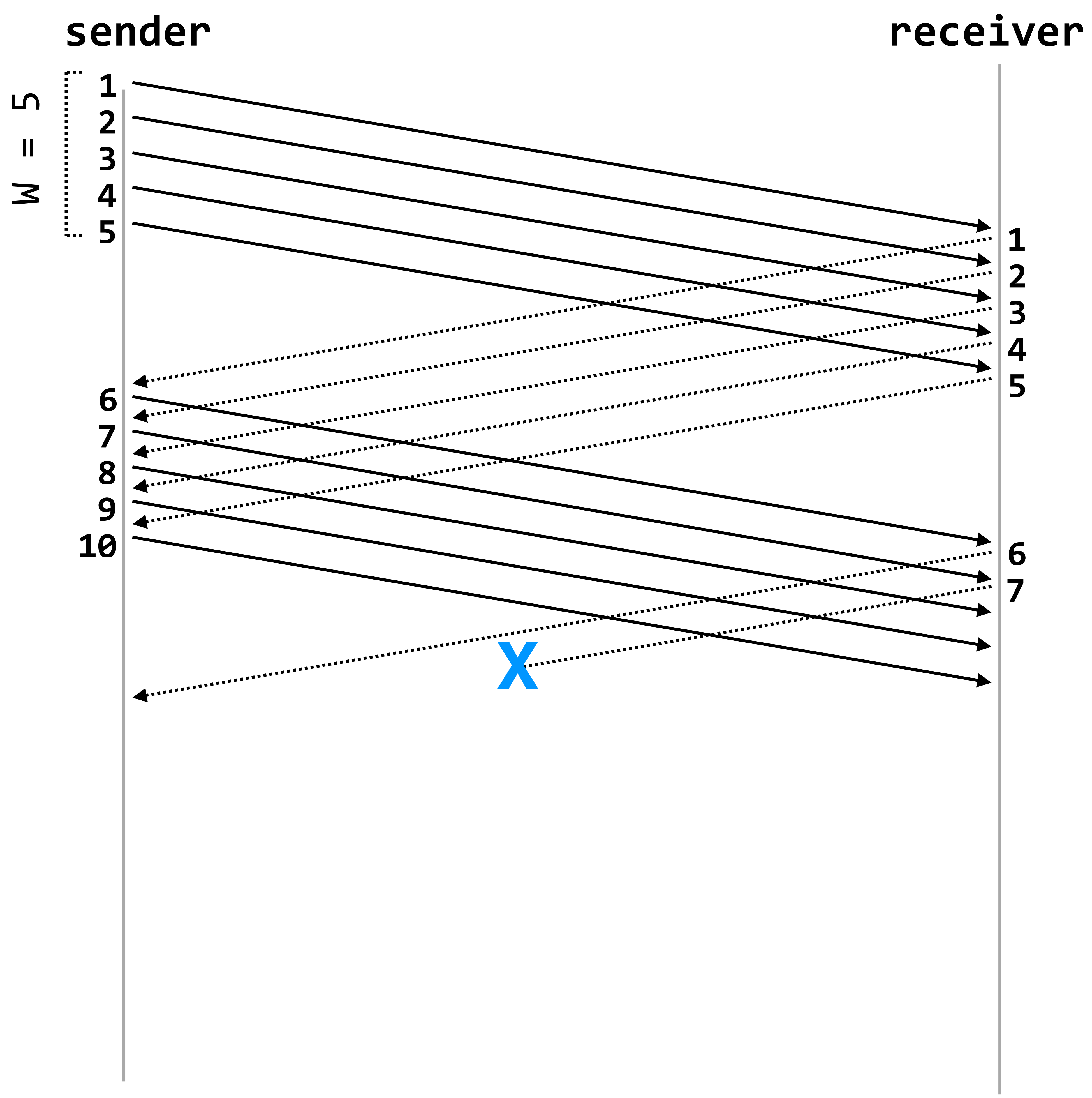
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



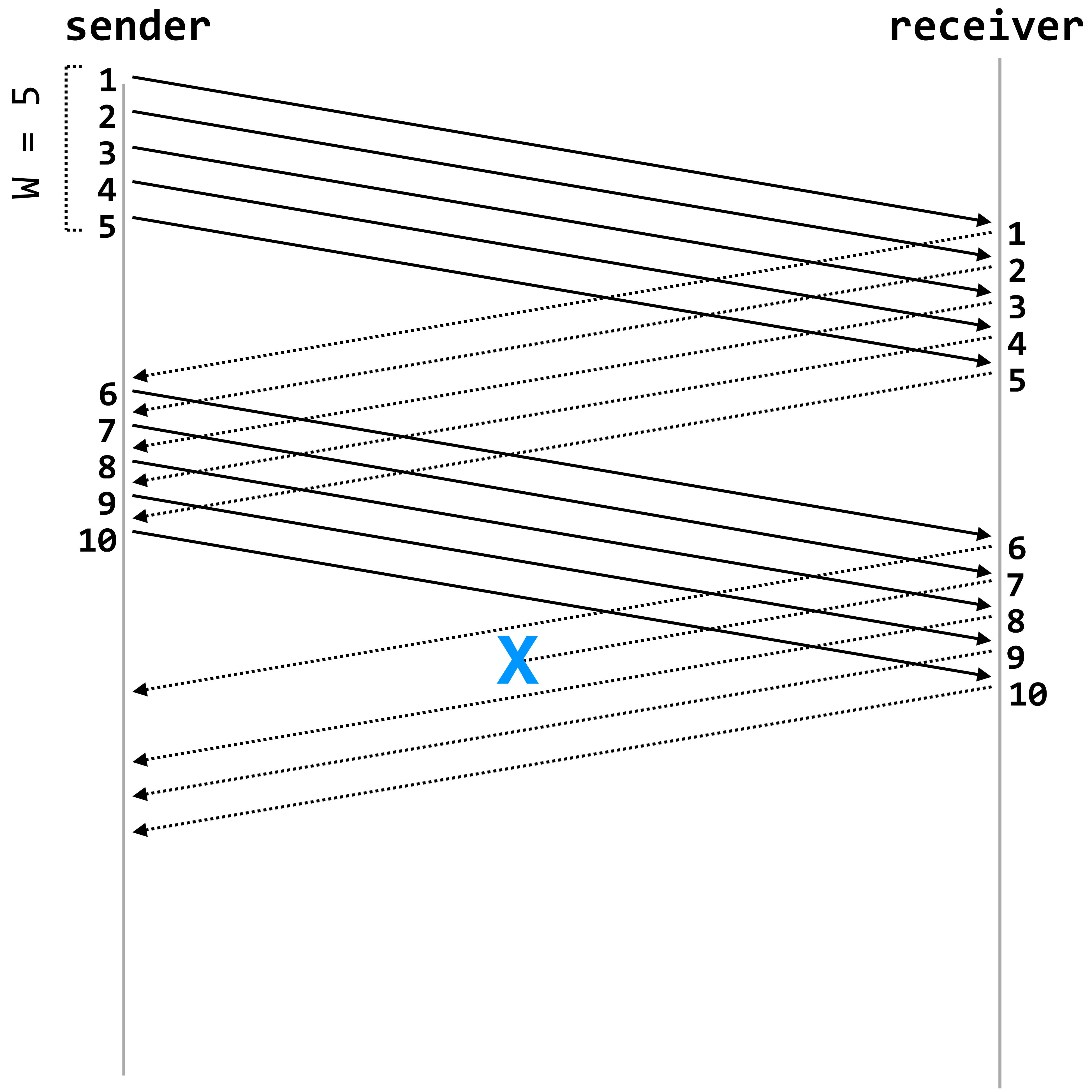
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



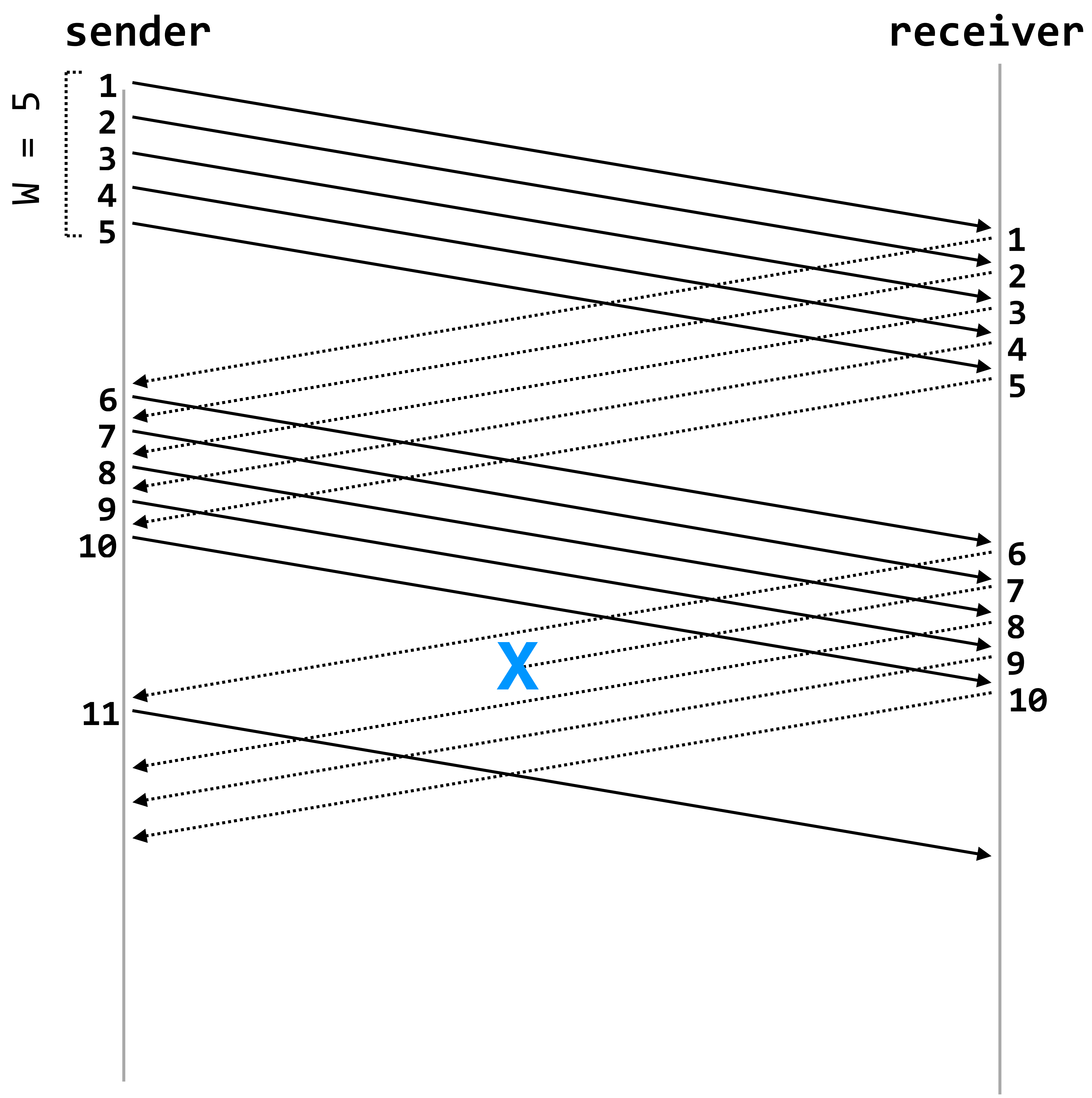
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



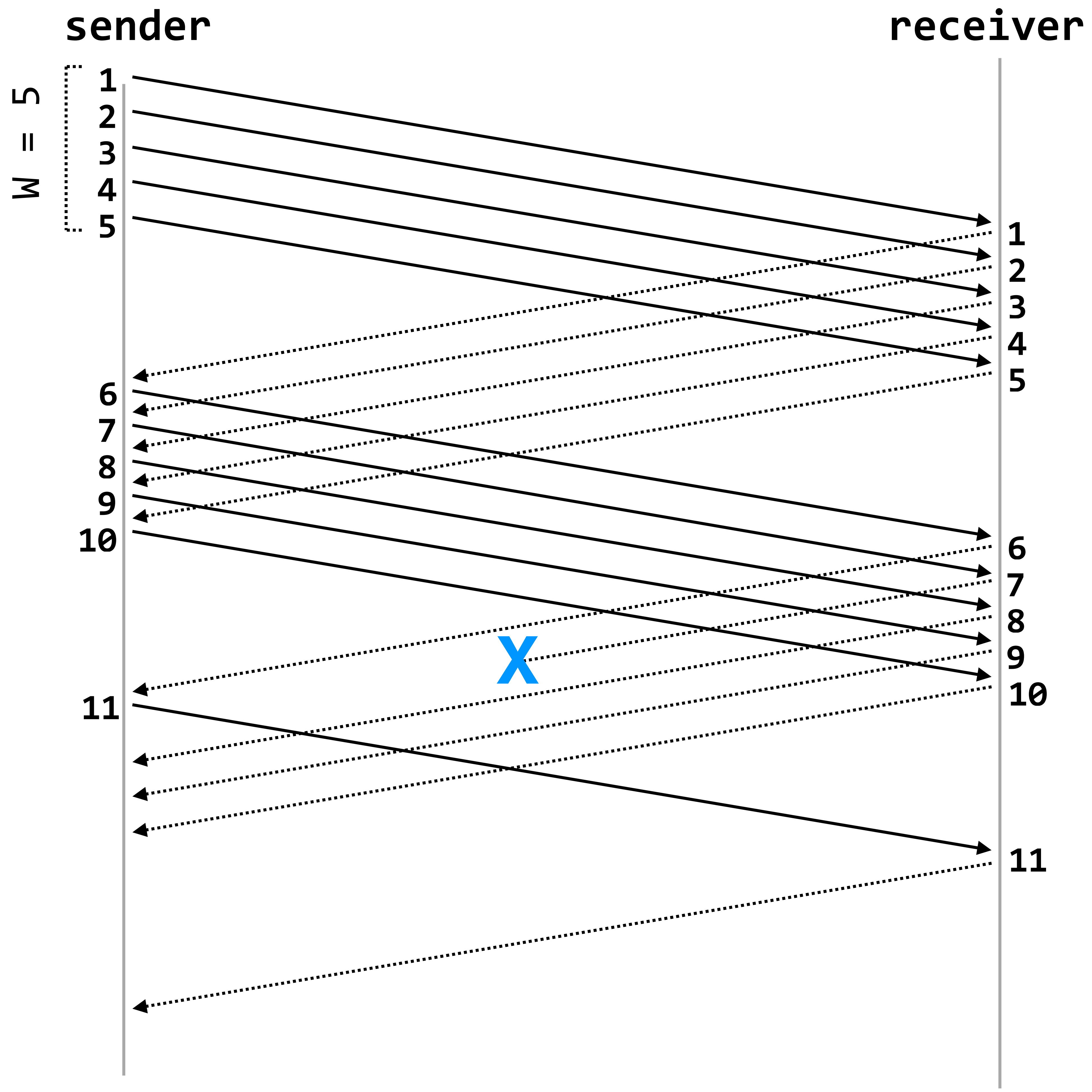
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



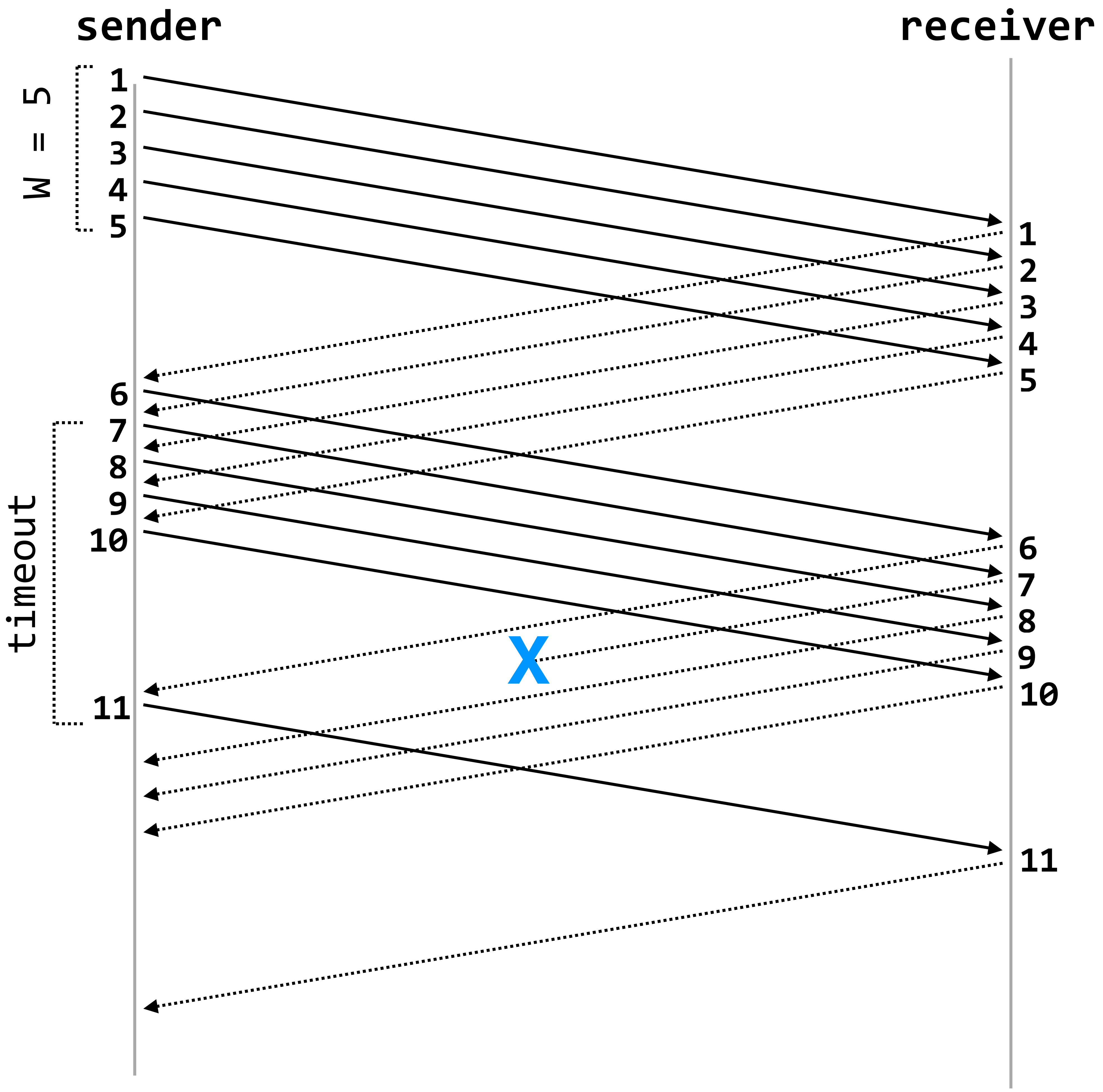
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



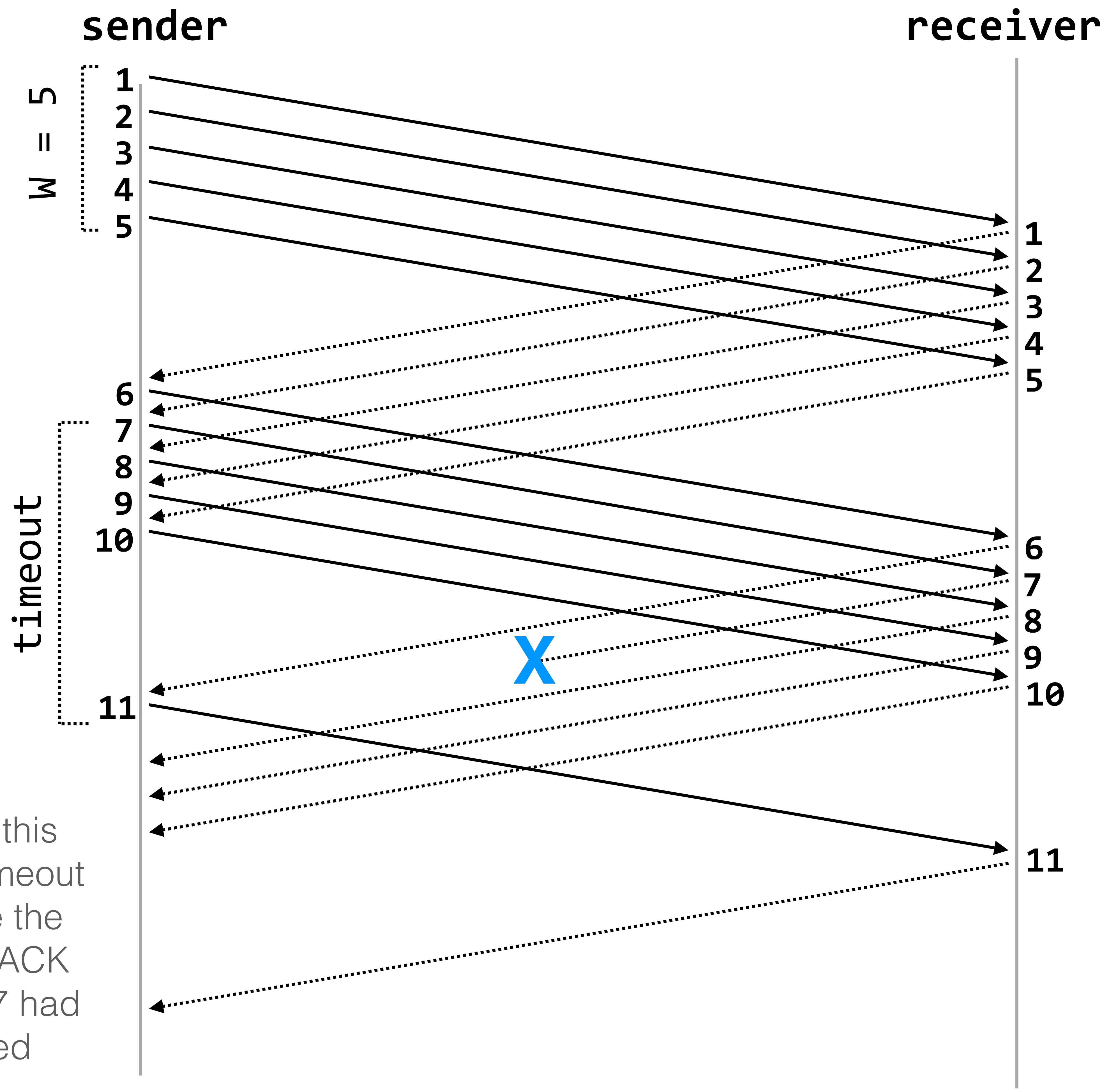
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received

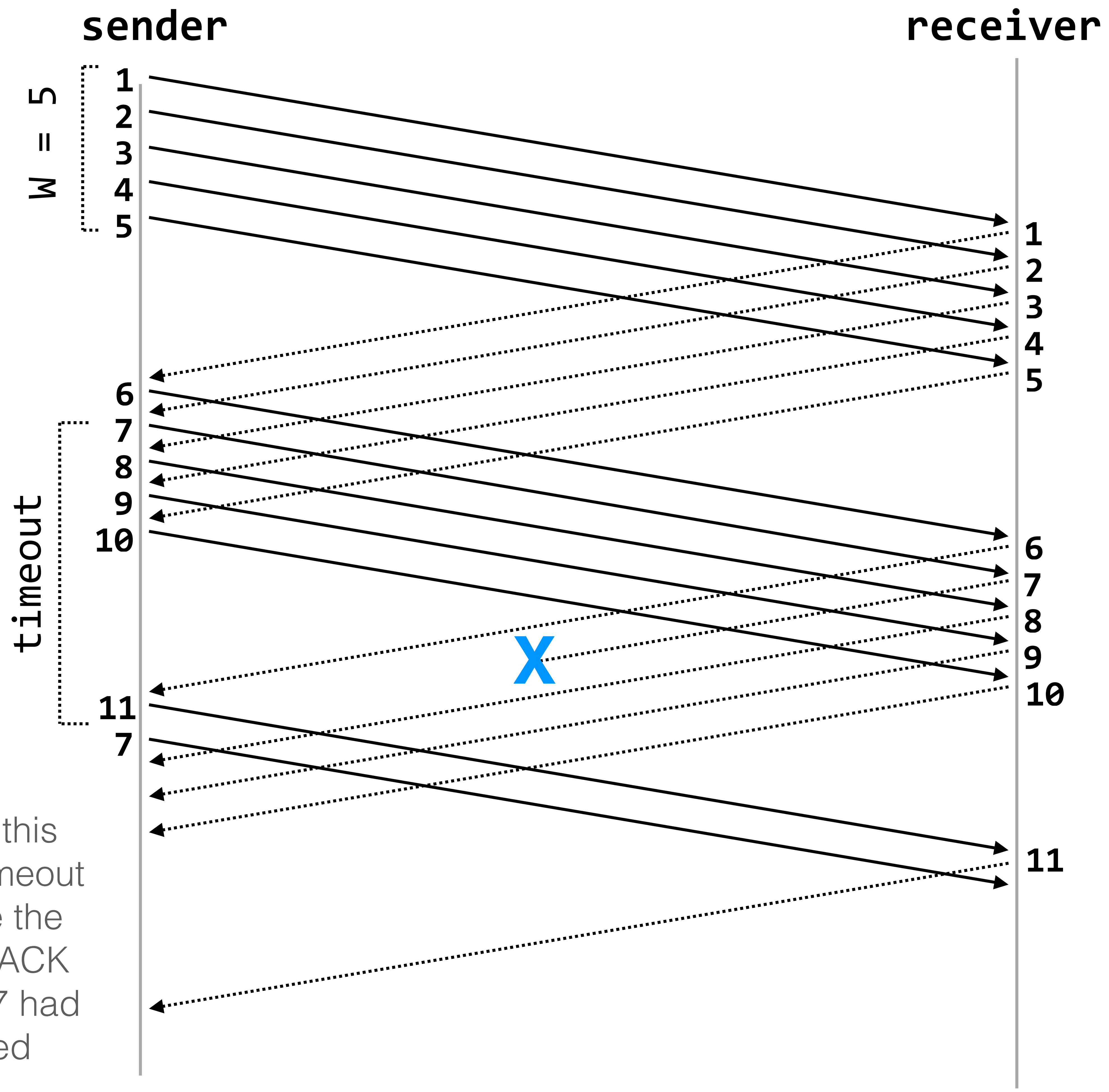
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received

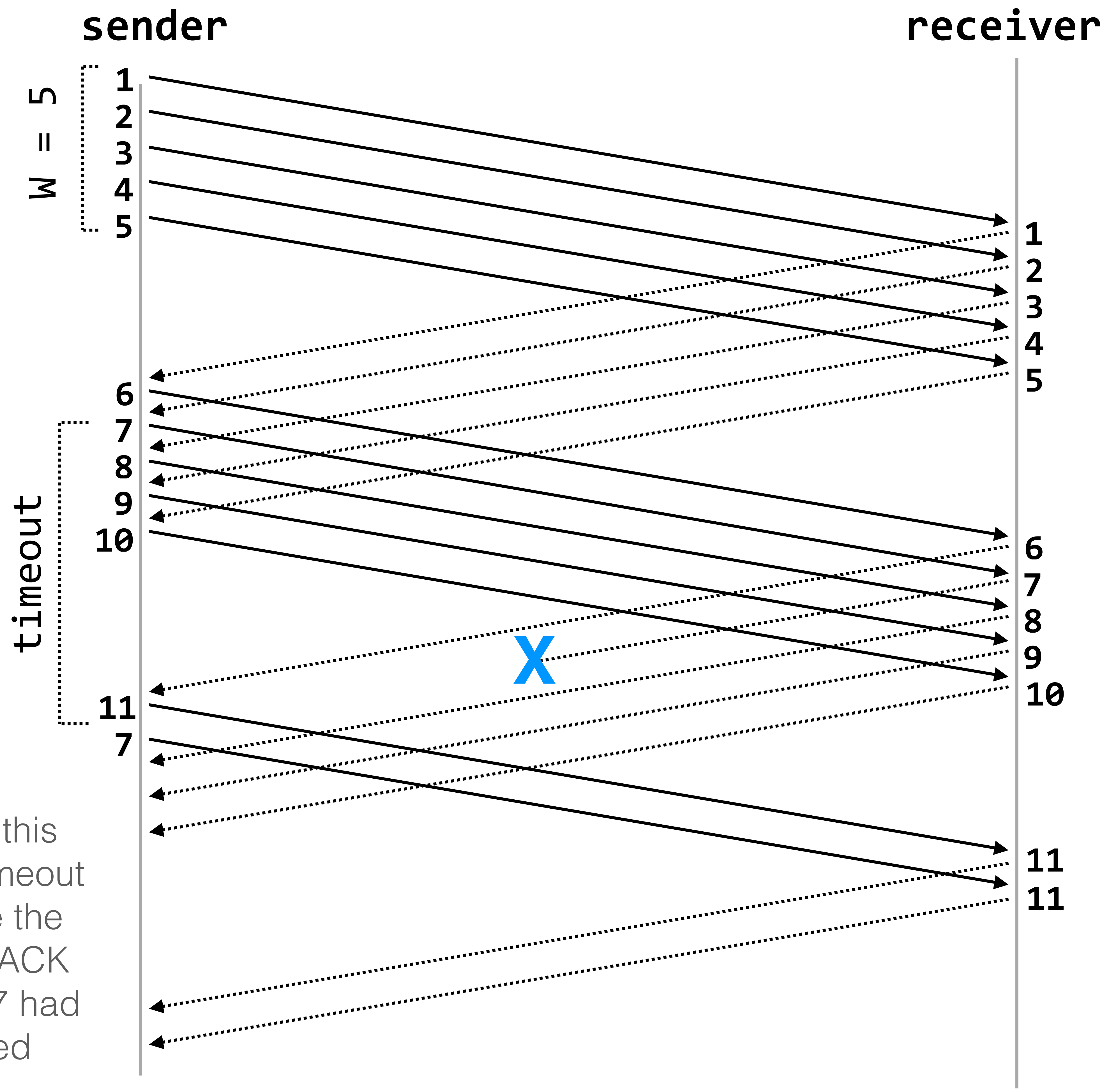
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received

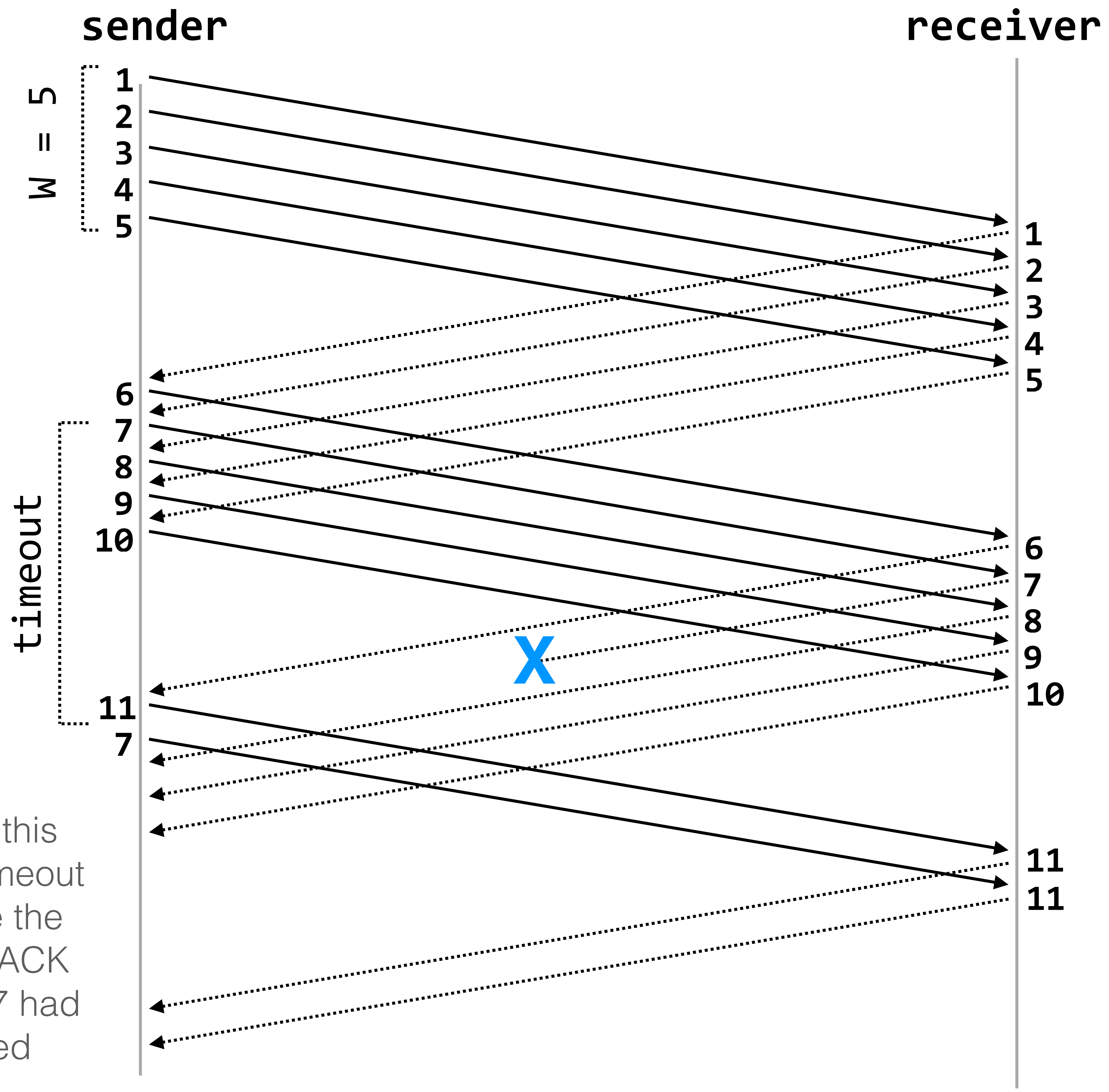
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received

spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

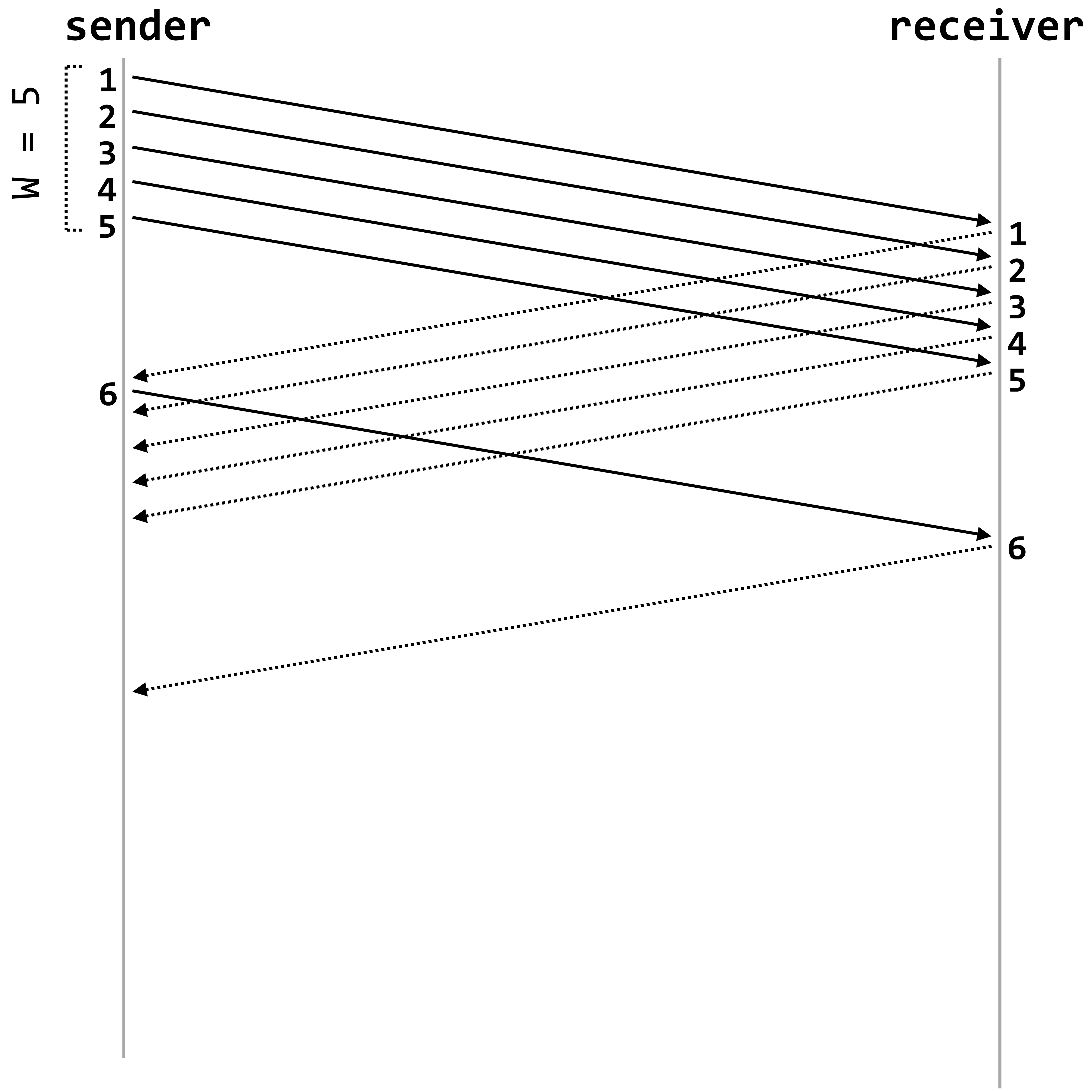
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



sequence numbers: used to order the packets

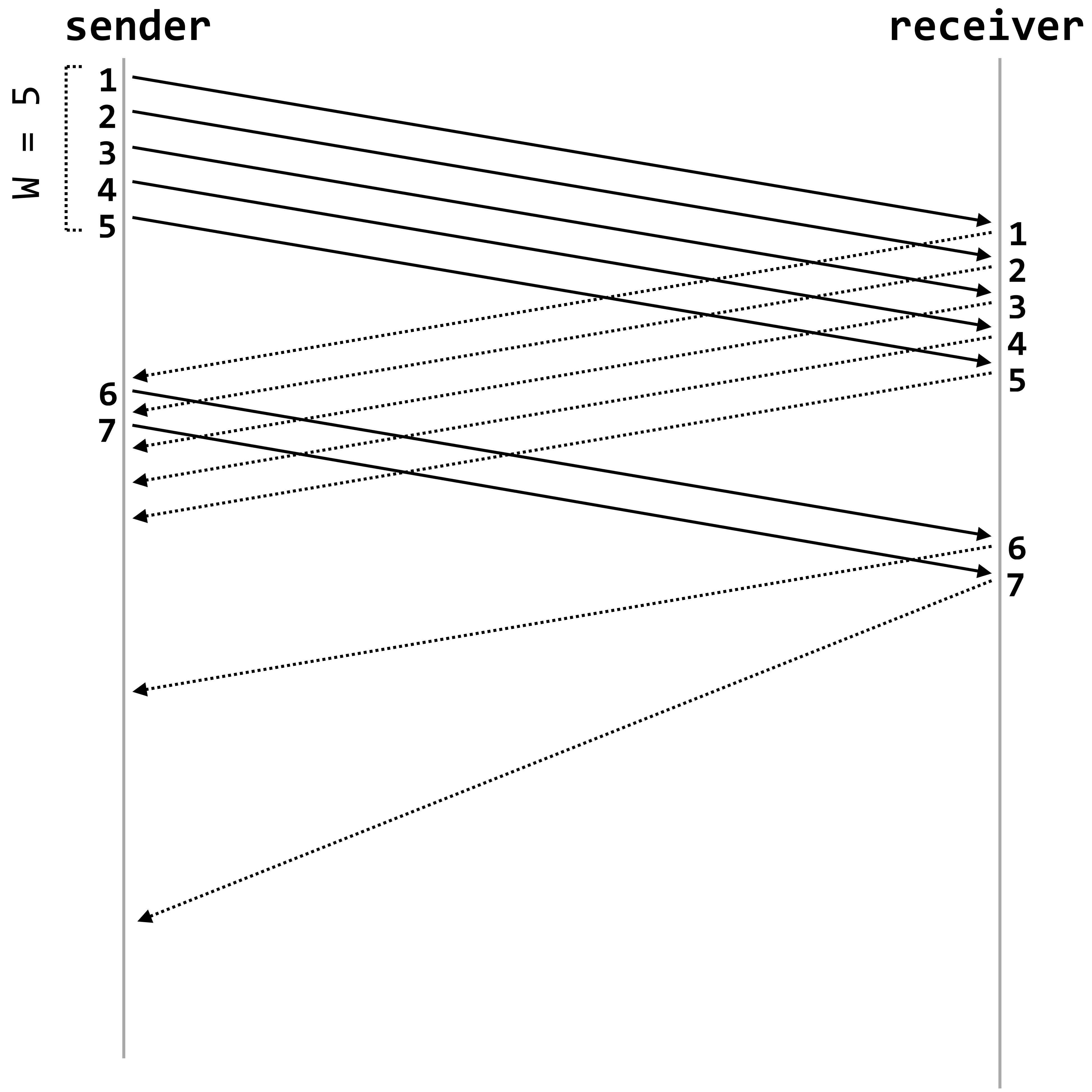
acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

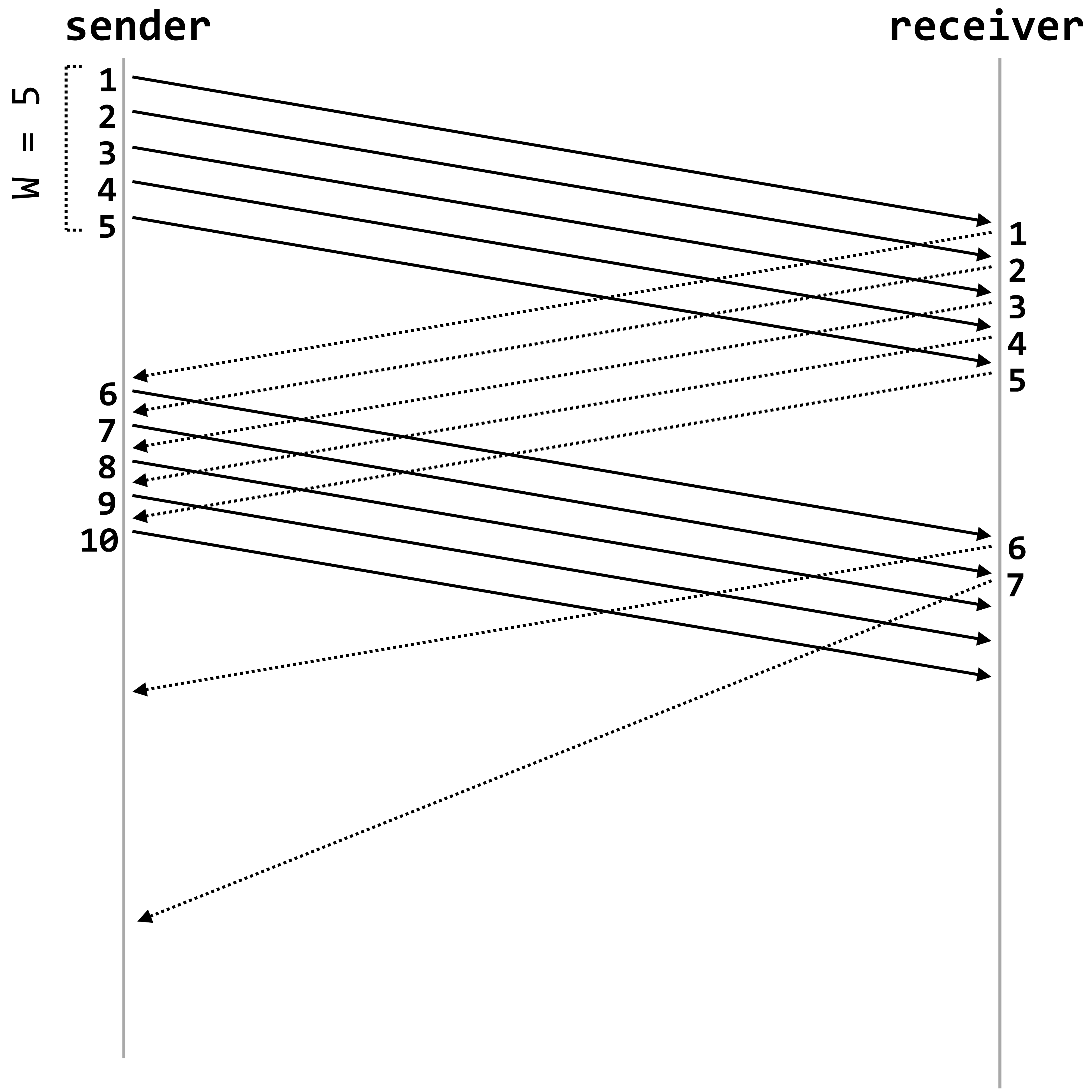
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

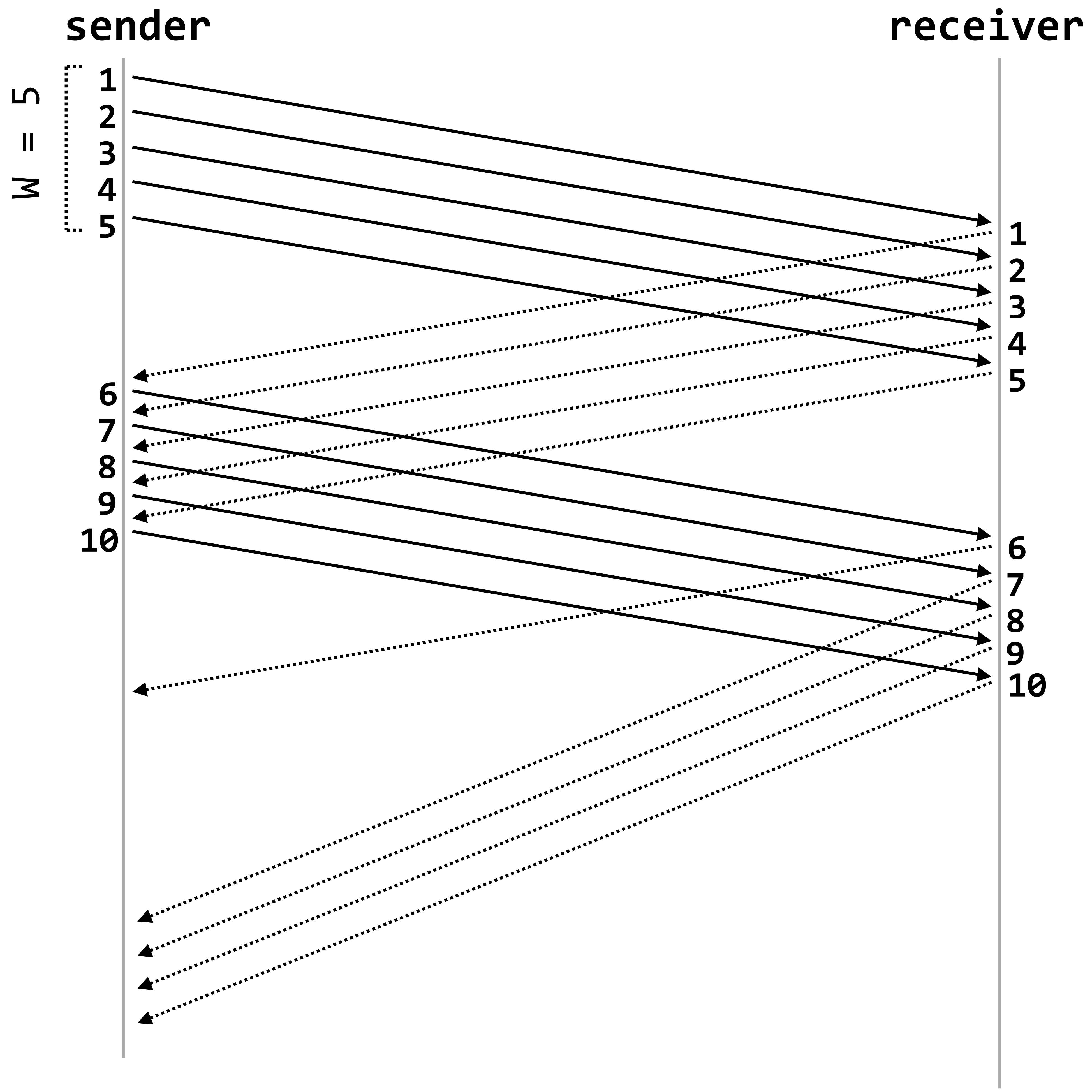
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

sequence numbers: used to order the packets

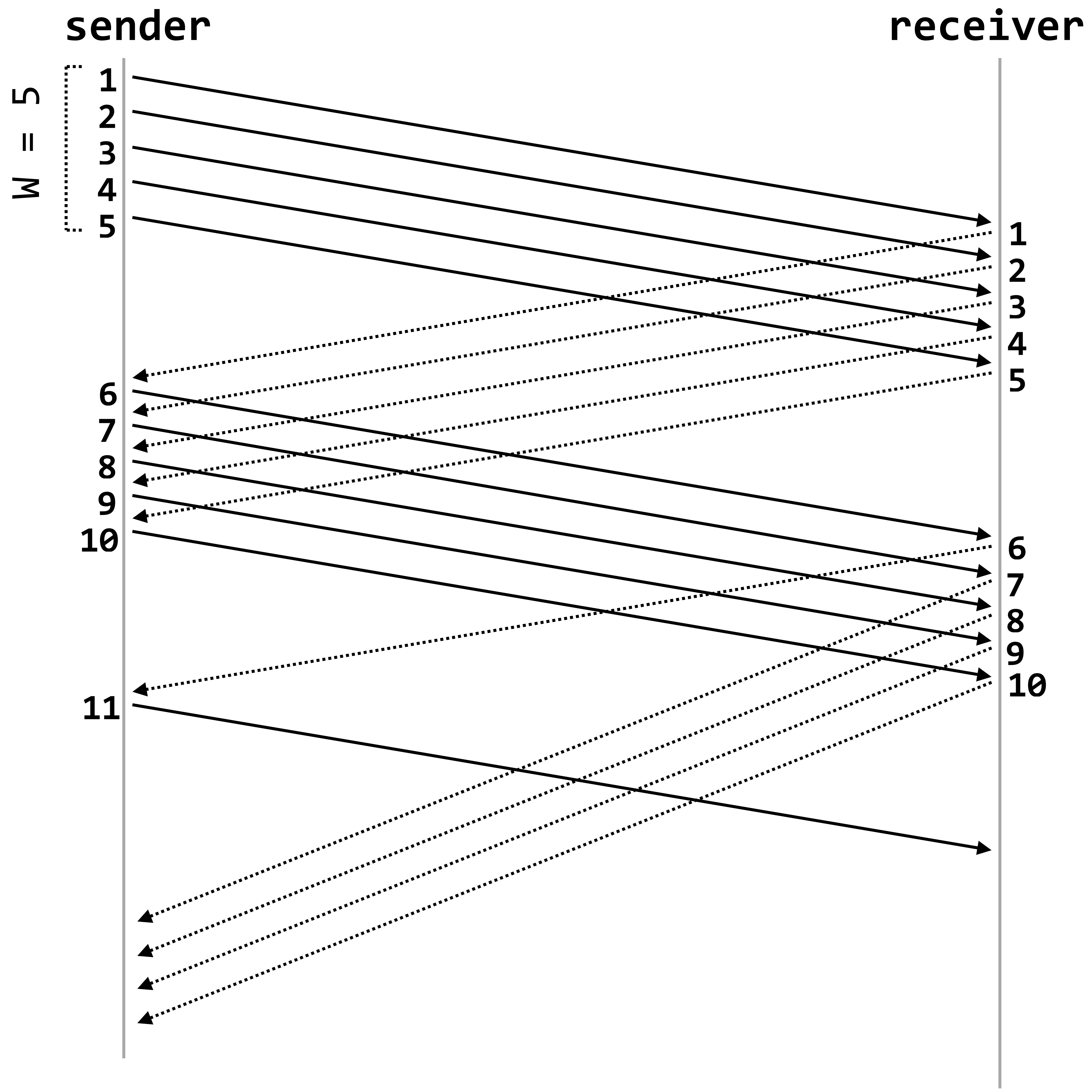
acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

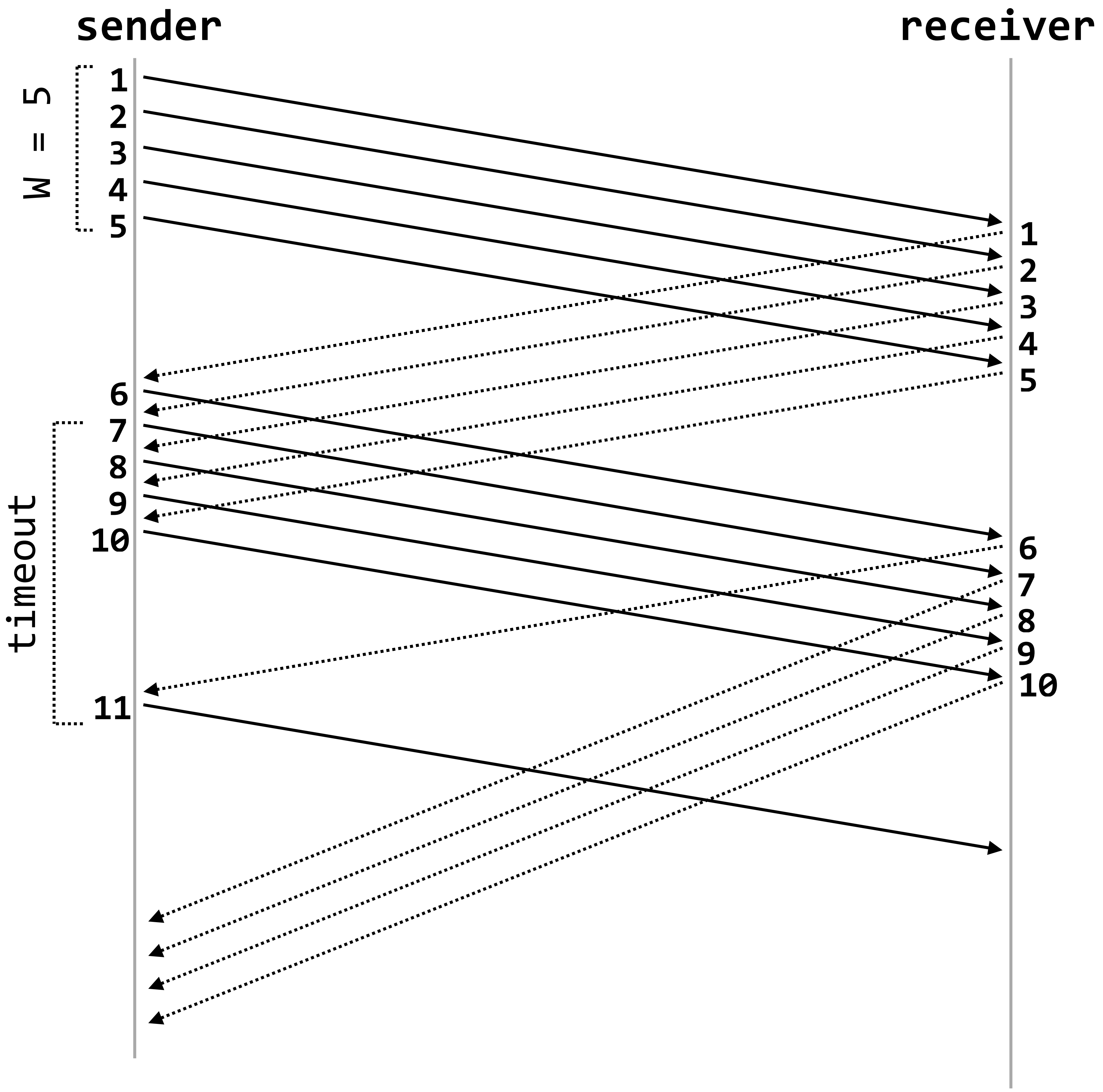
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

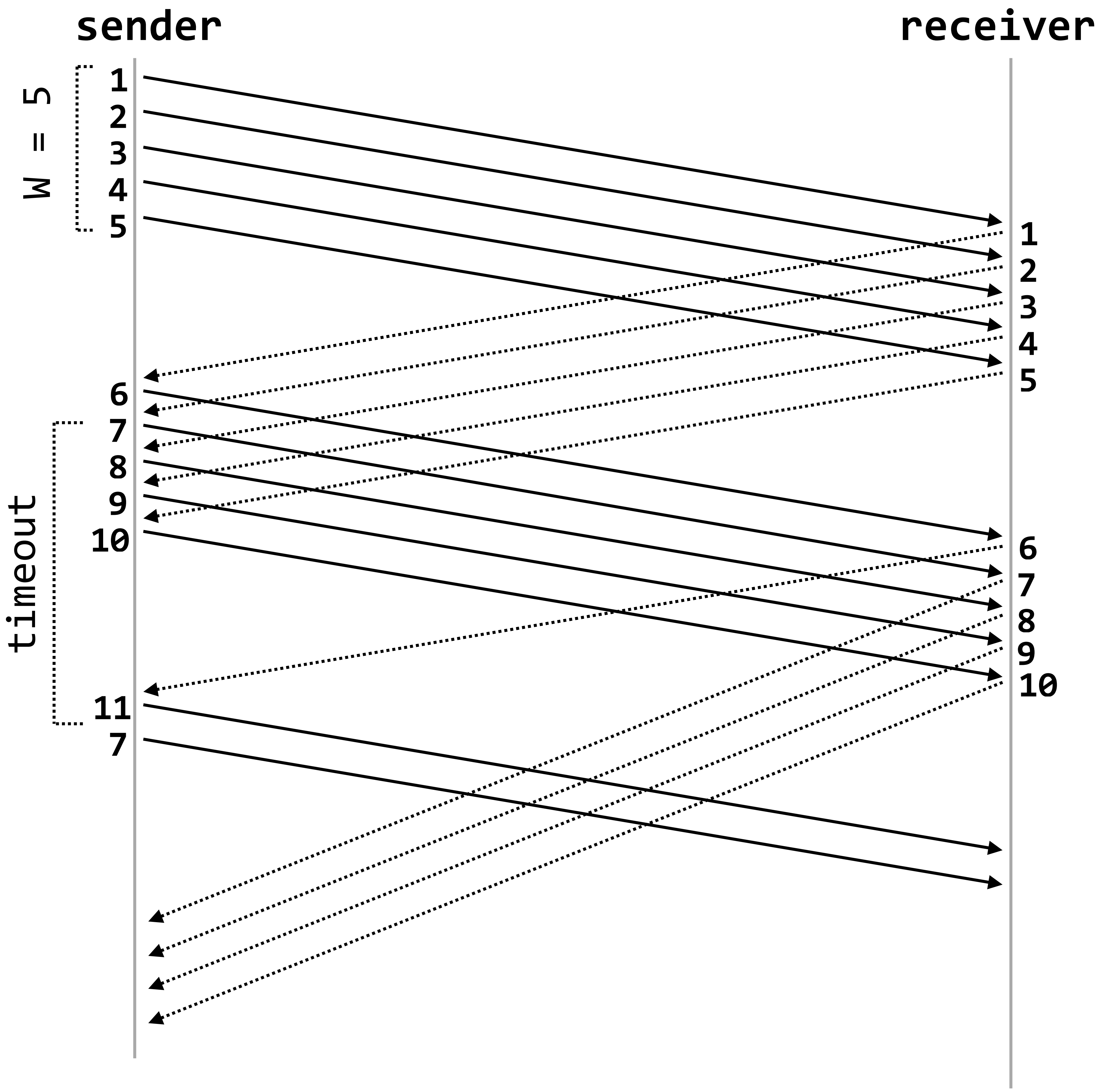
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



spurious retransmission: the sender retransmitted a packet that the receiver had already ACKed

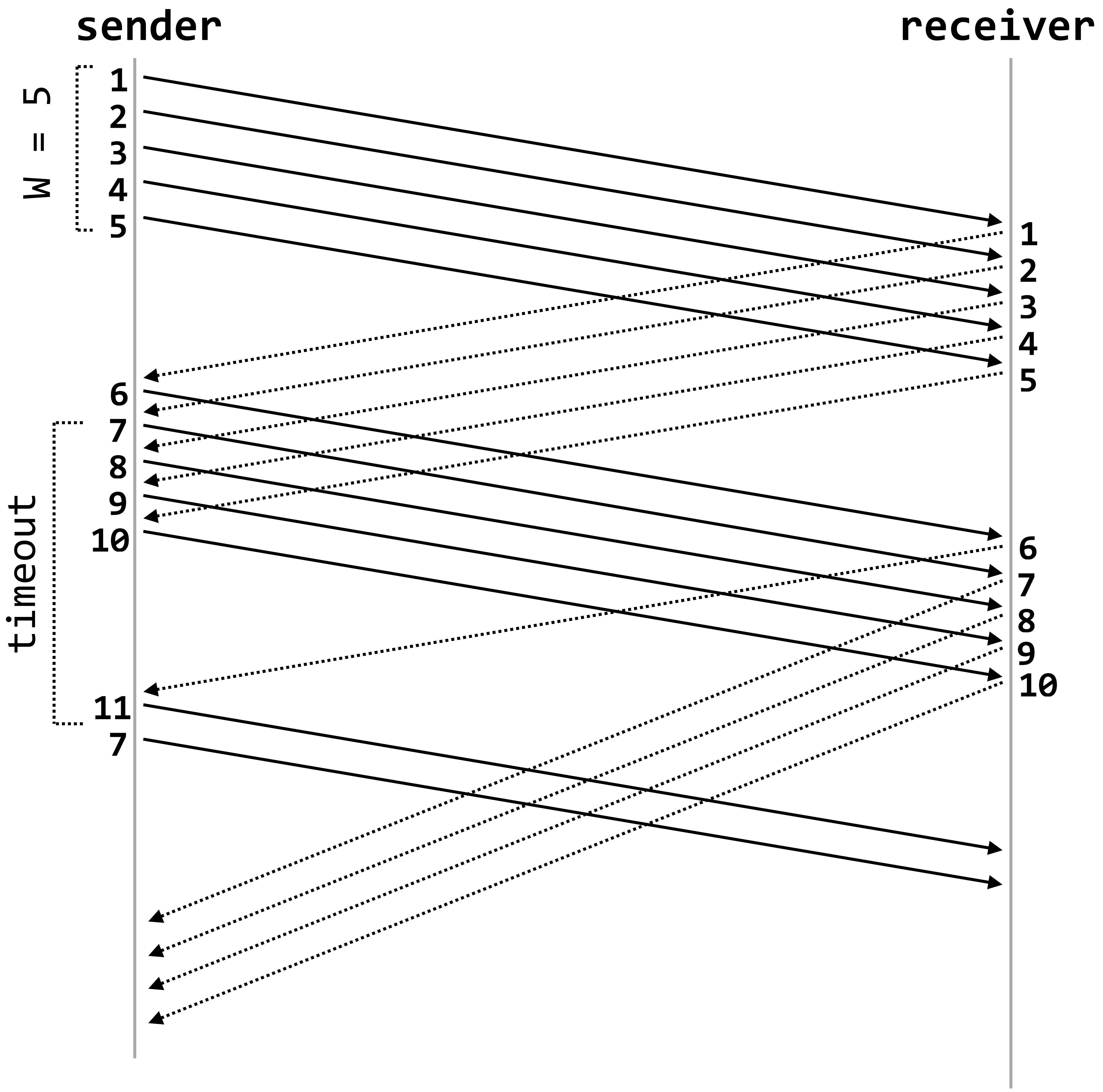
sequence numbers: used to order the packets

acknowledgments ("ACKs"): used to confirm that a packet has been received

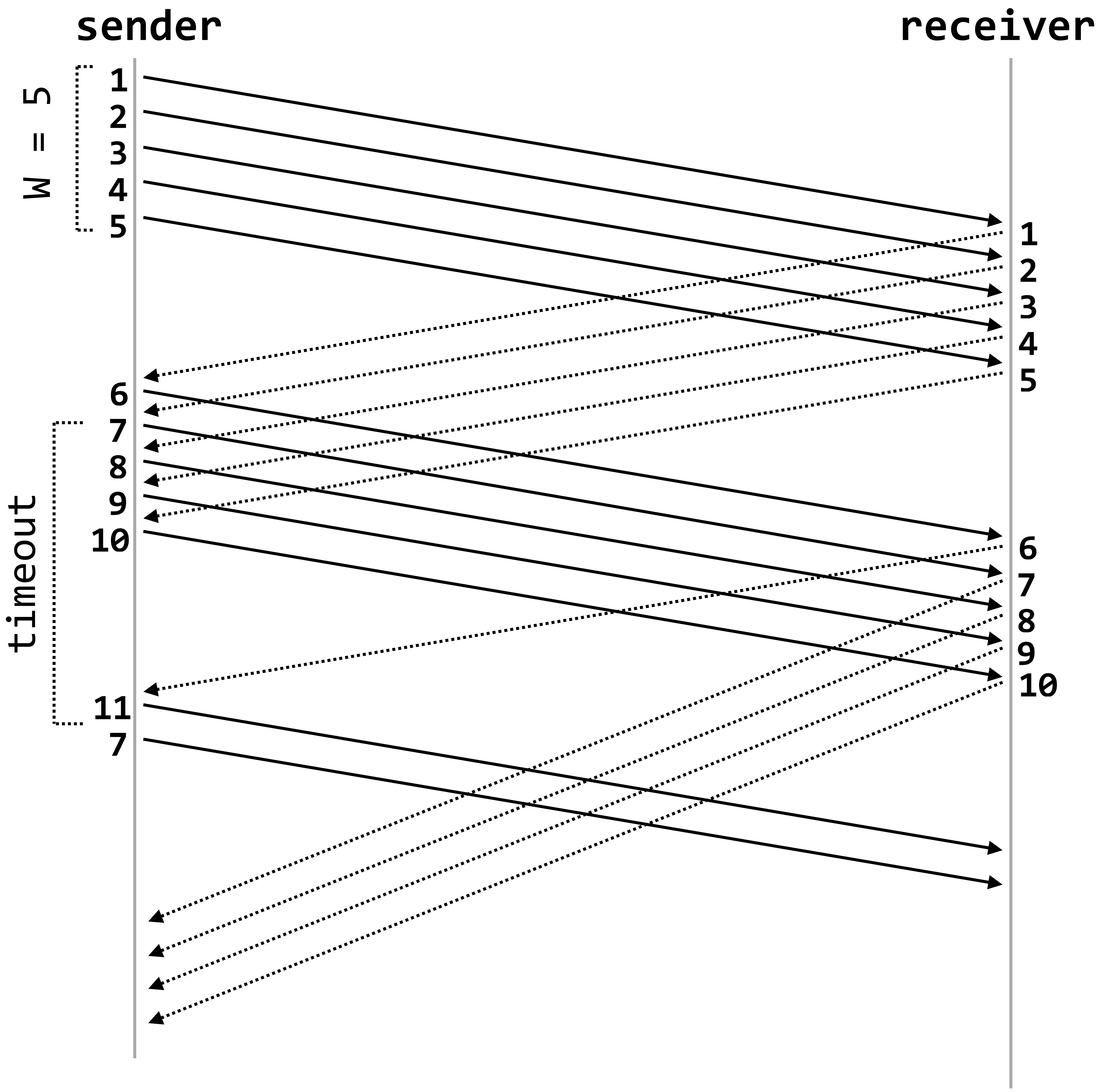
an ACK with sequence number k indicates that the receiver has received **all packets up to and including k**

timeouts: used to retransmit packets

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application

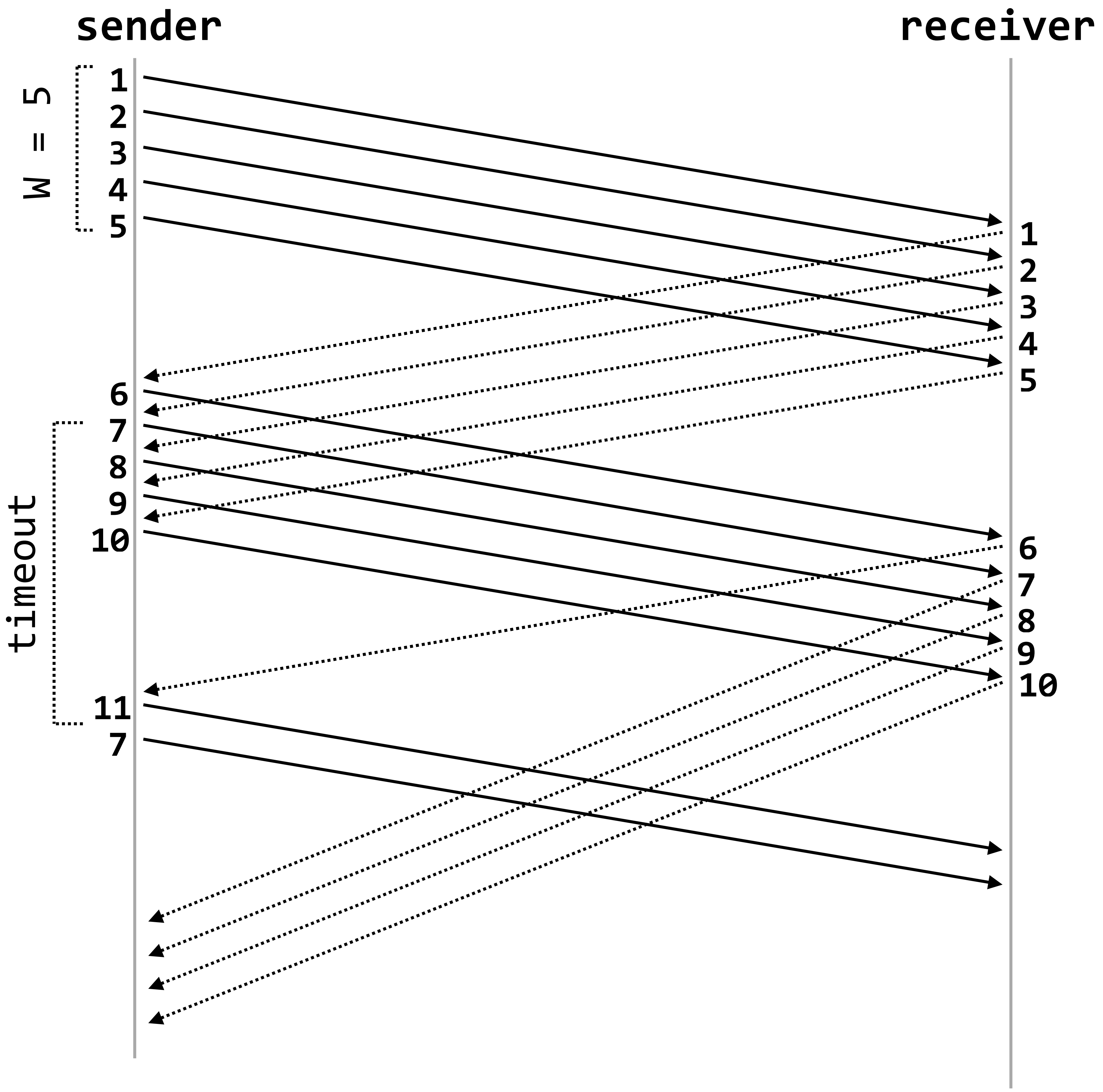


reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



question: what should `W` be?

reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application



question: what should `W` be?

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?

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

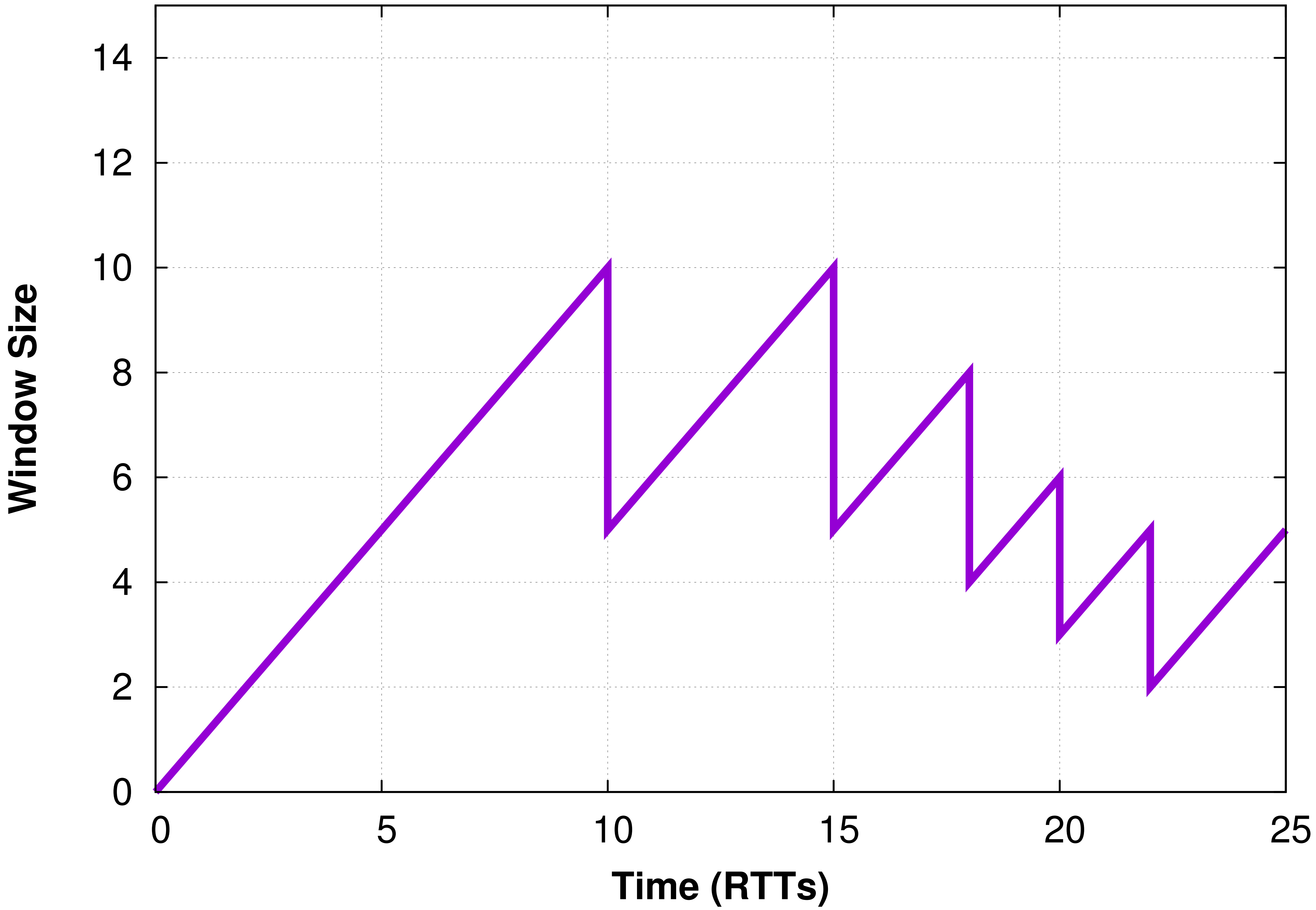
congestion control: controlling the source rates to achieve **efficiency** and **fairness**

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

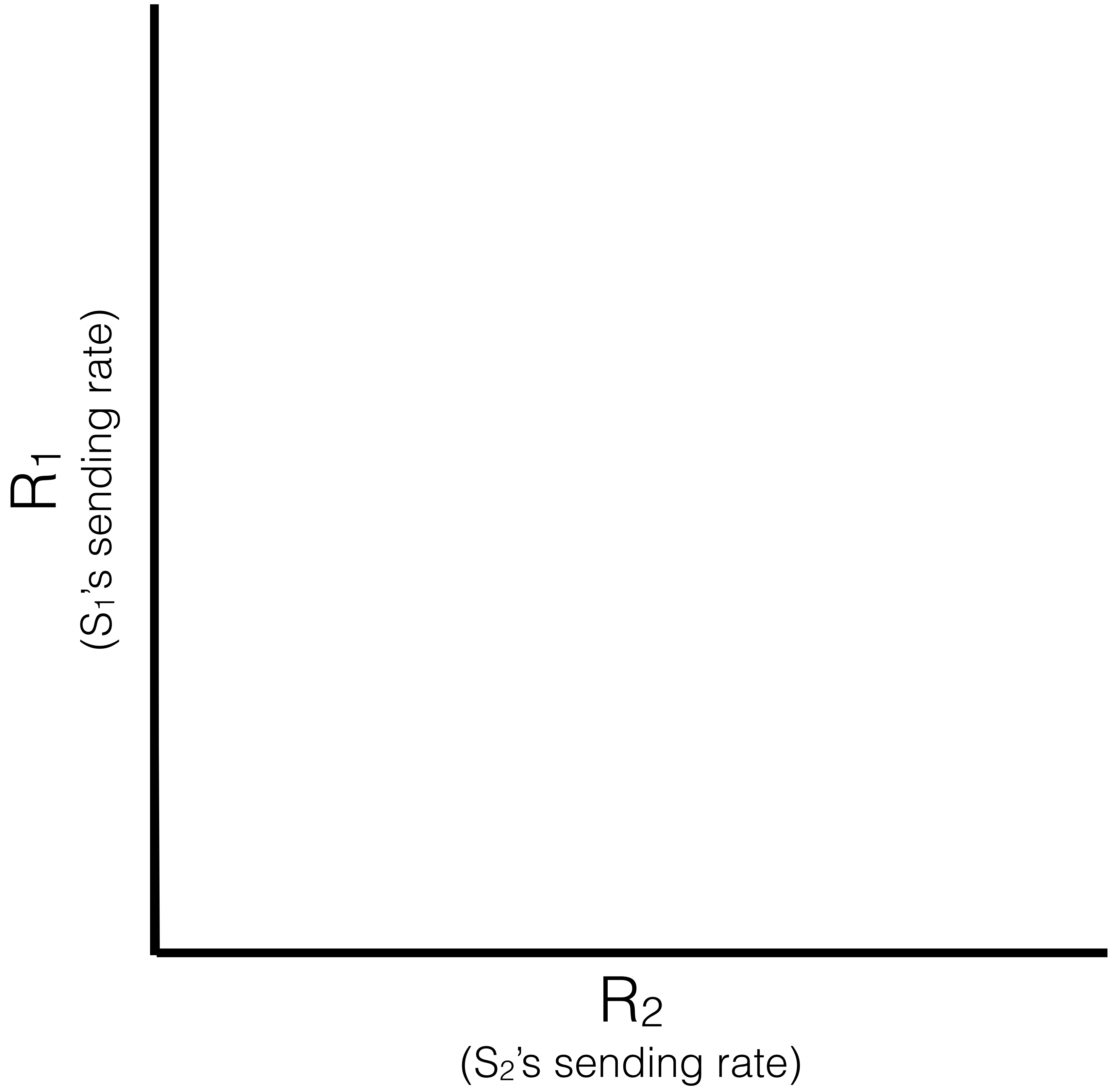


efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

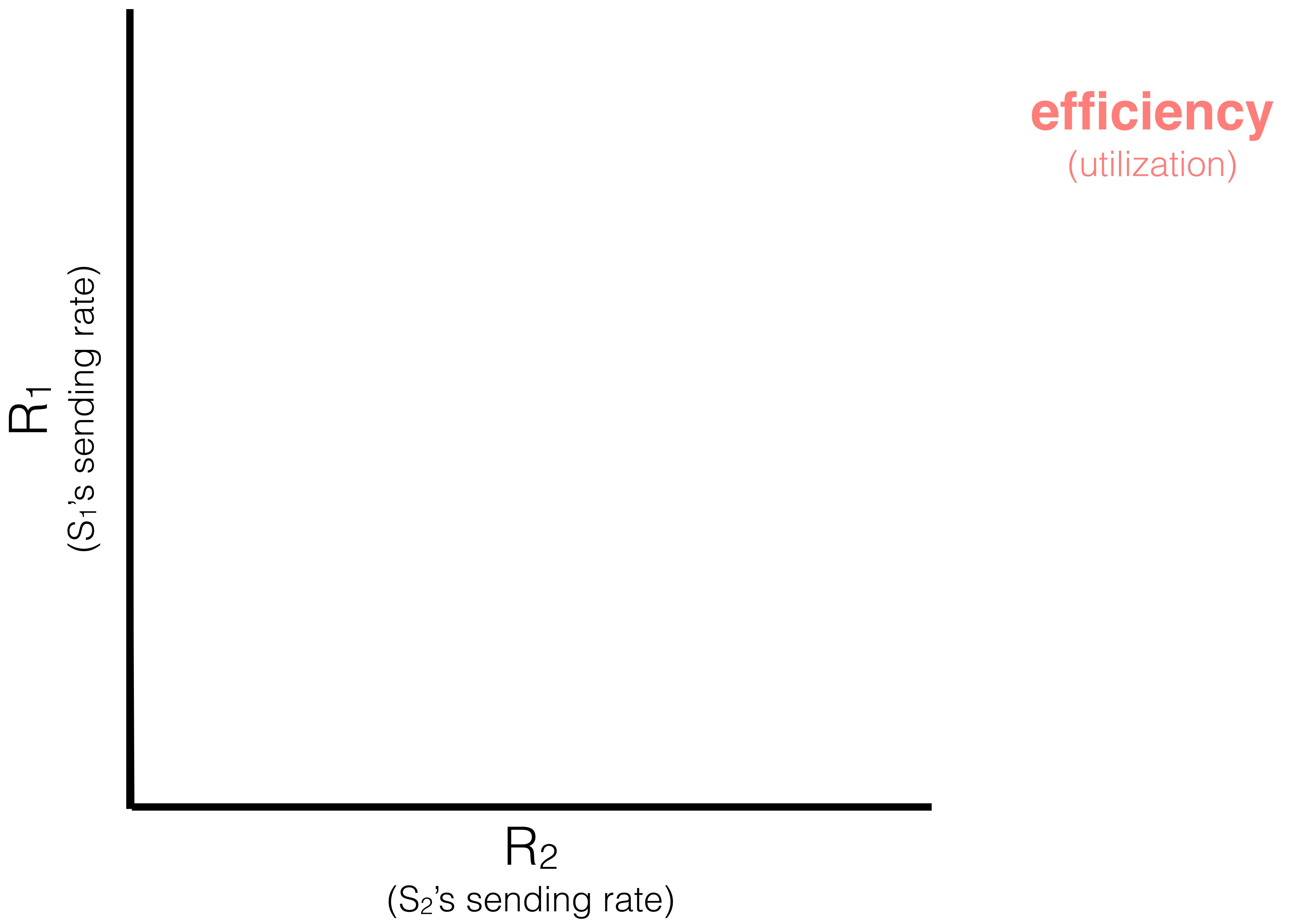


efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

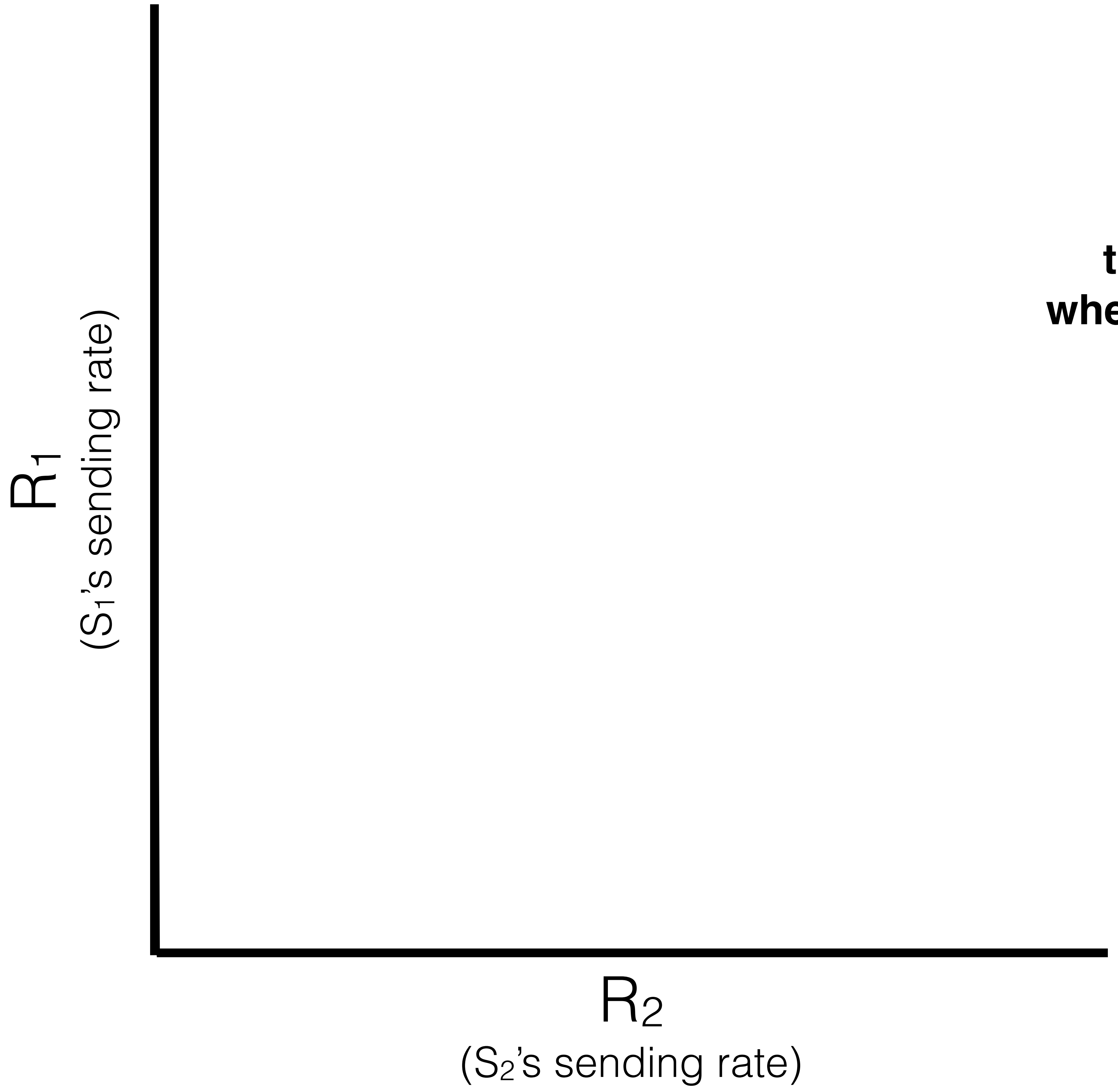


efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

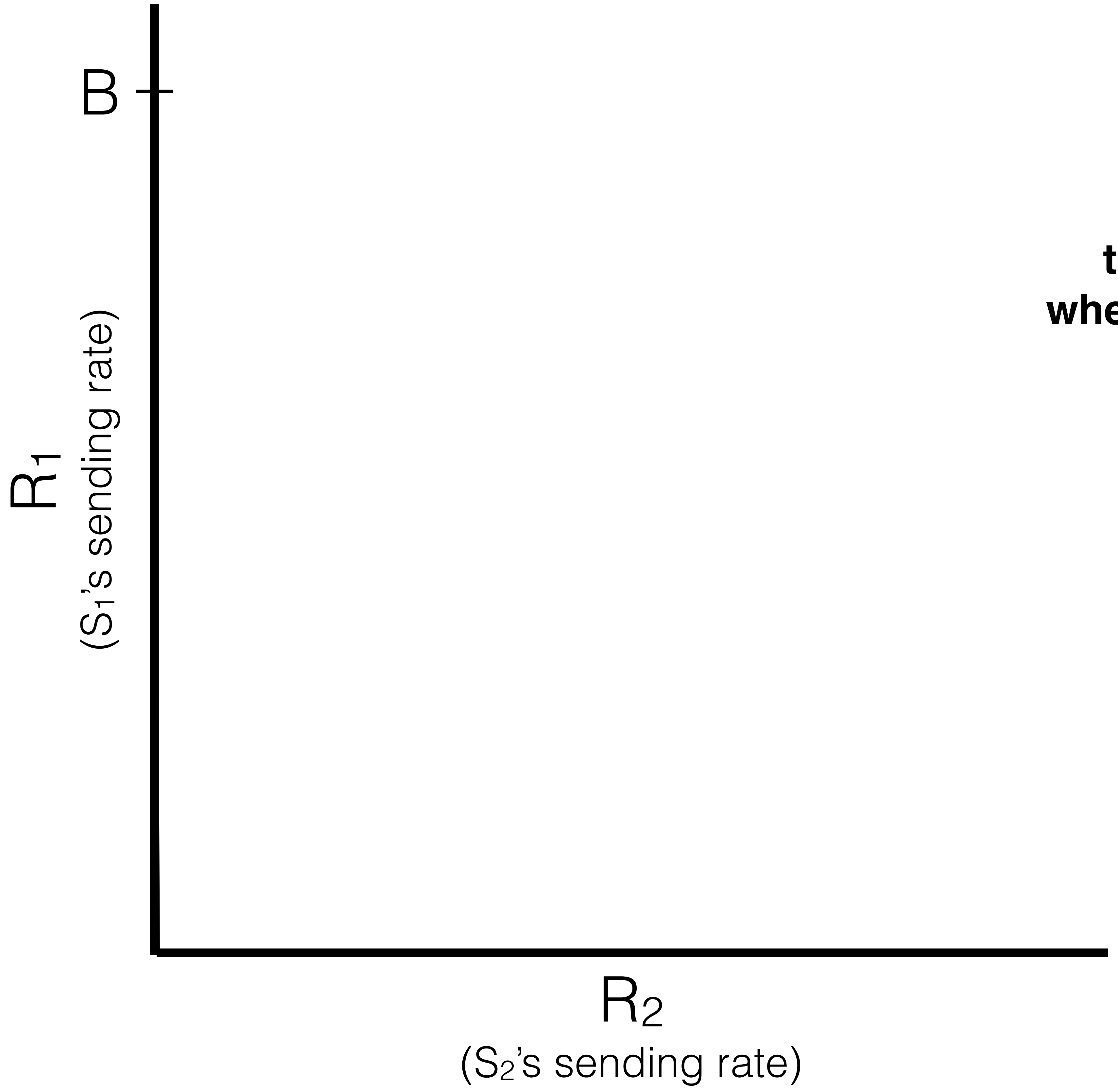
the network is fully utilized when the bottleneck link is "full"

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

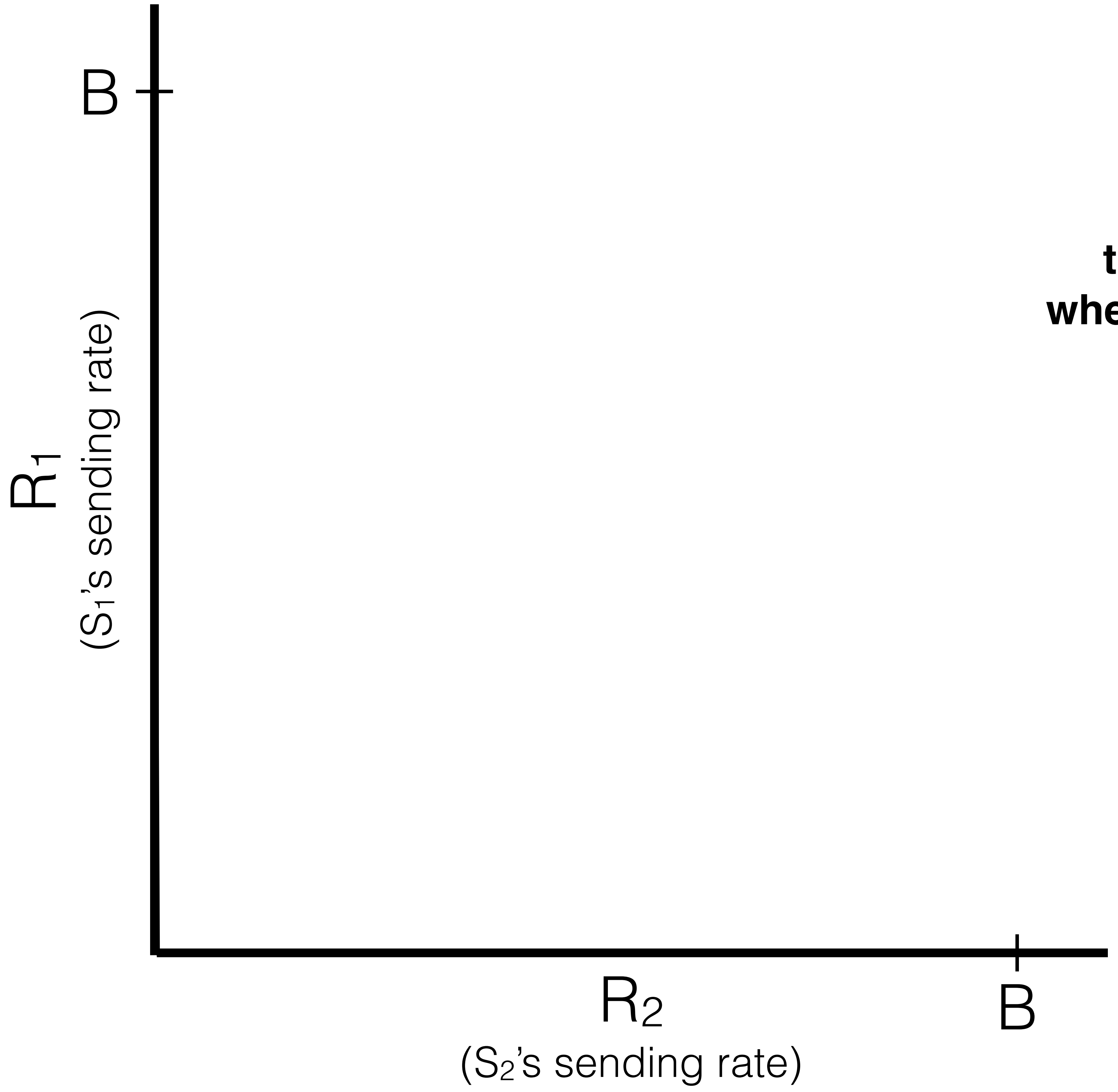
the network is fully utilized when the bottleneck link is "full"

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency

(utilization)

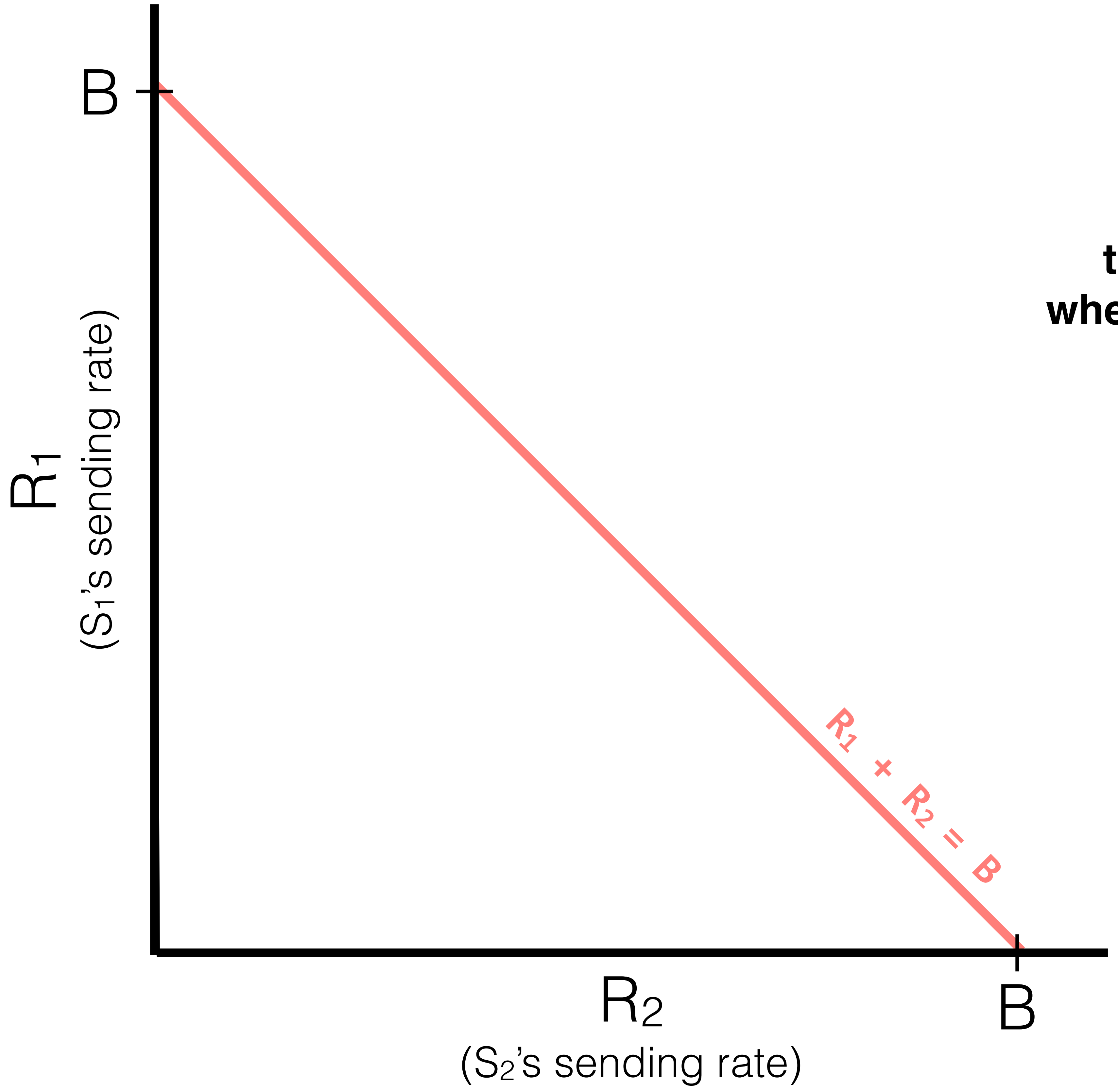
the network is fully utilized when the bottleneck link is "full"

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency

(utilization)

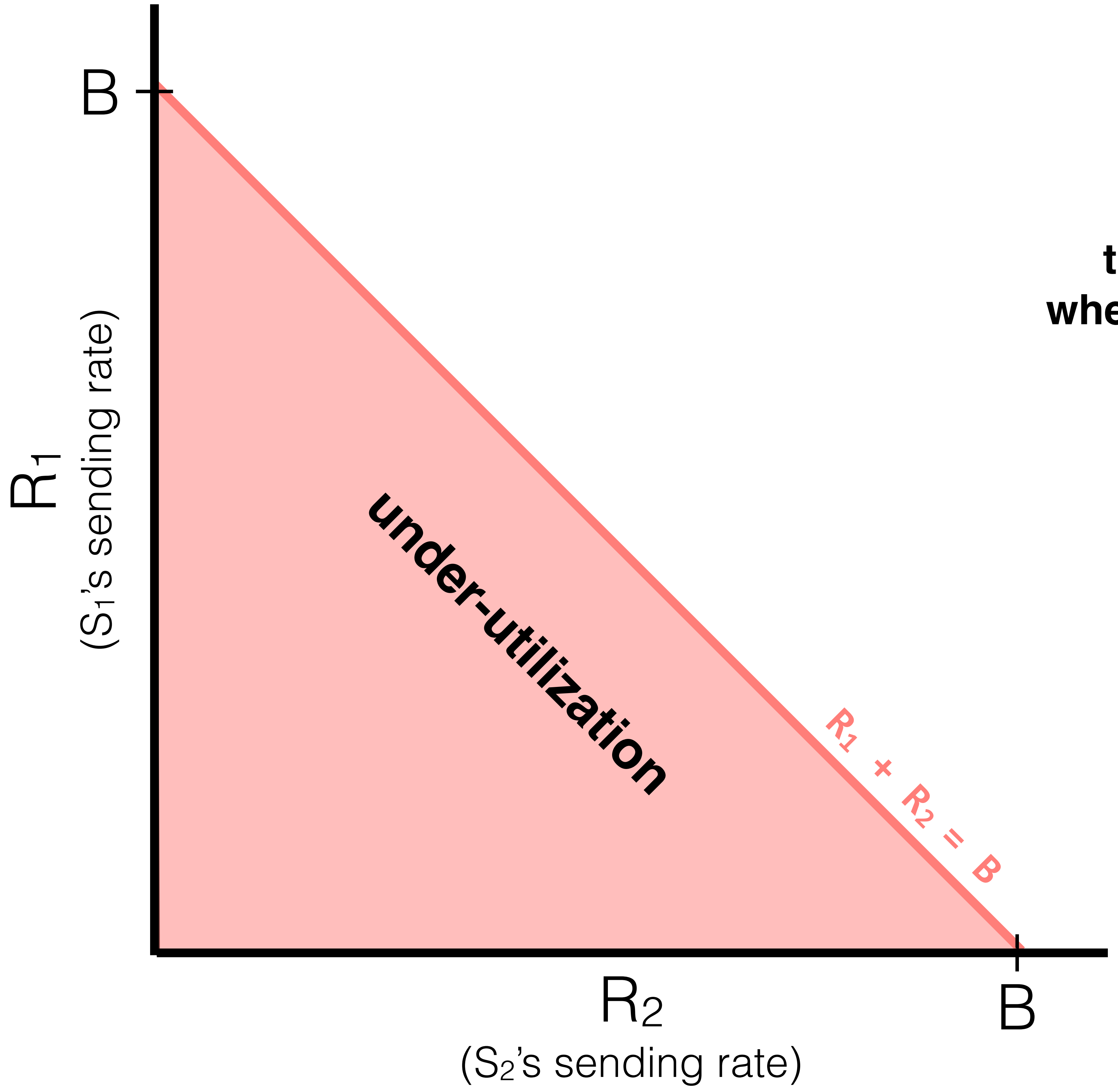
the network is fully utilized when the bottleneck link is "full"

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency

(utilization)

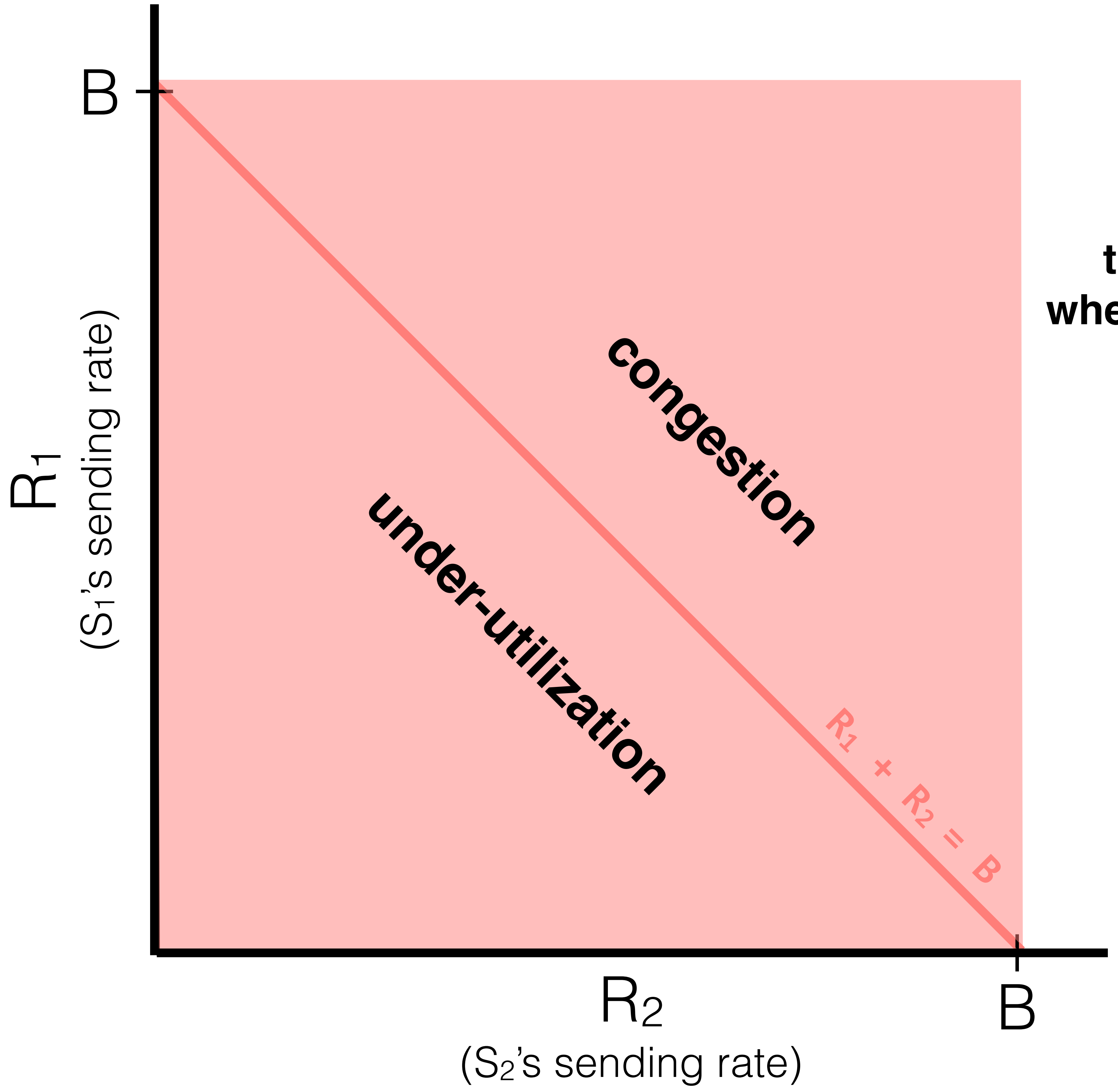
the network is fully utilized when the bottleneck link is "full"

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency

(utilization)

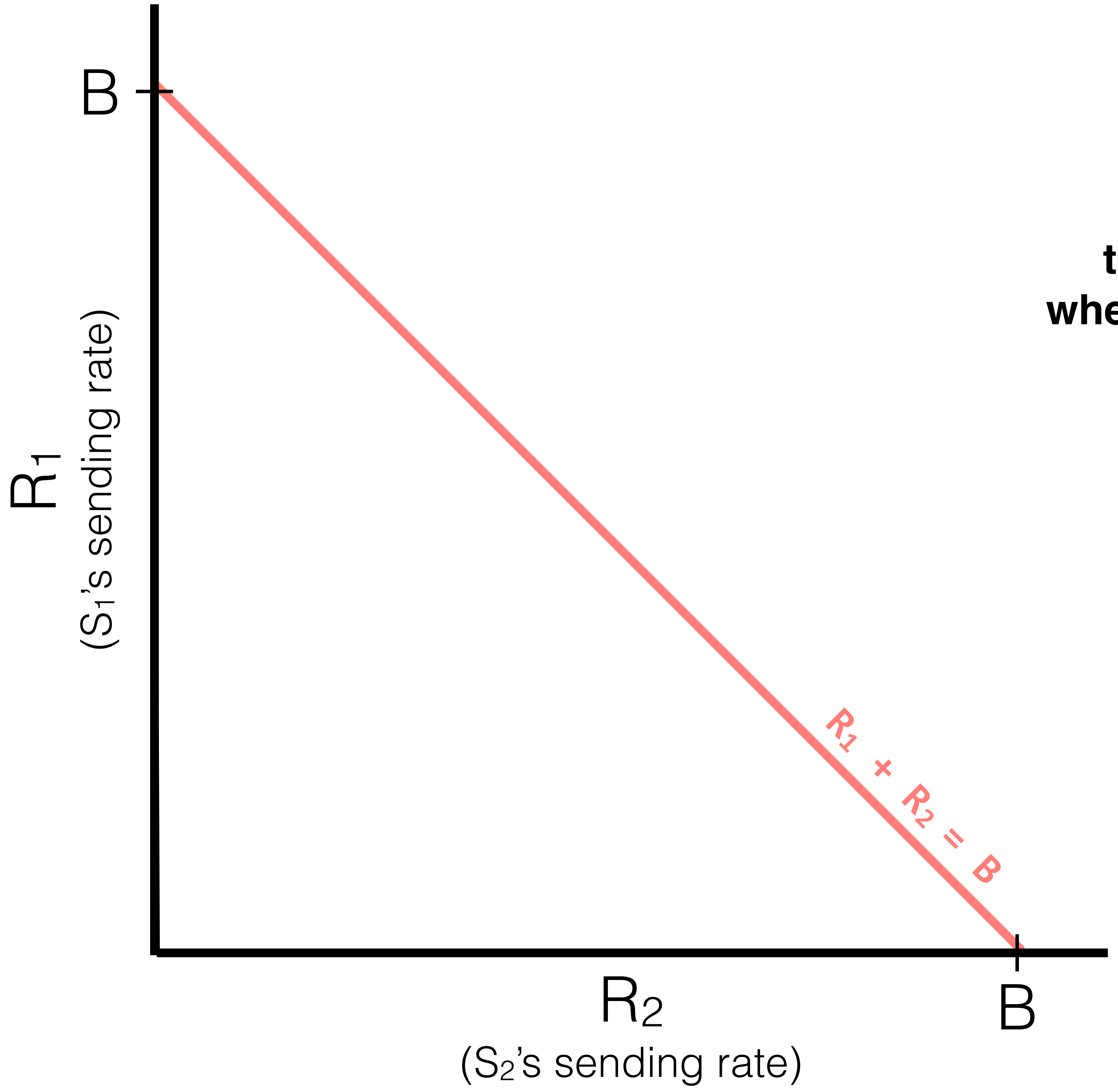
the network is fully utilized when the bottleneck link is "full"

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

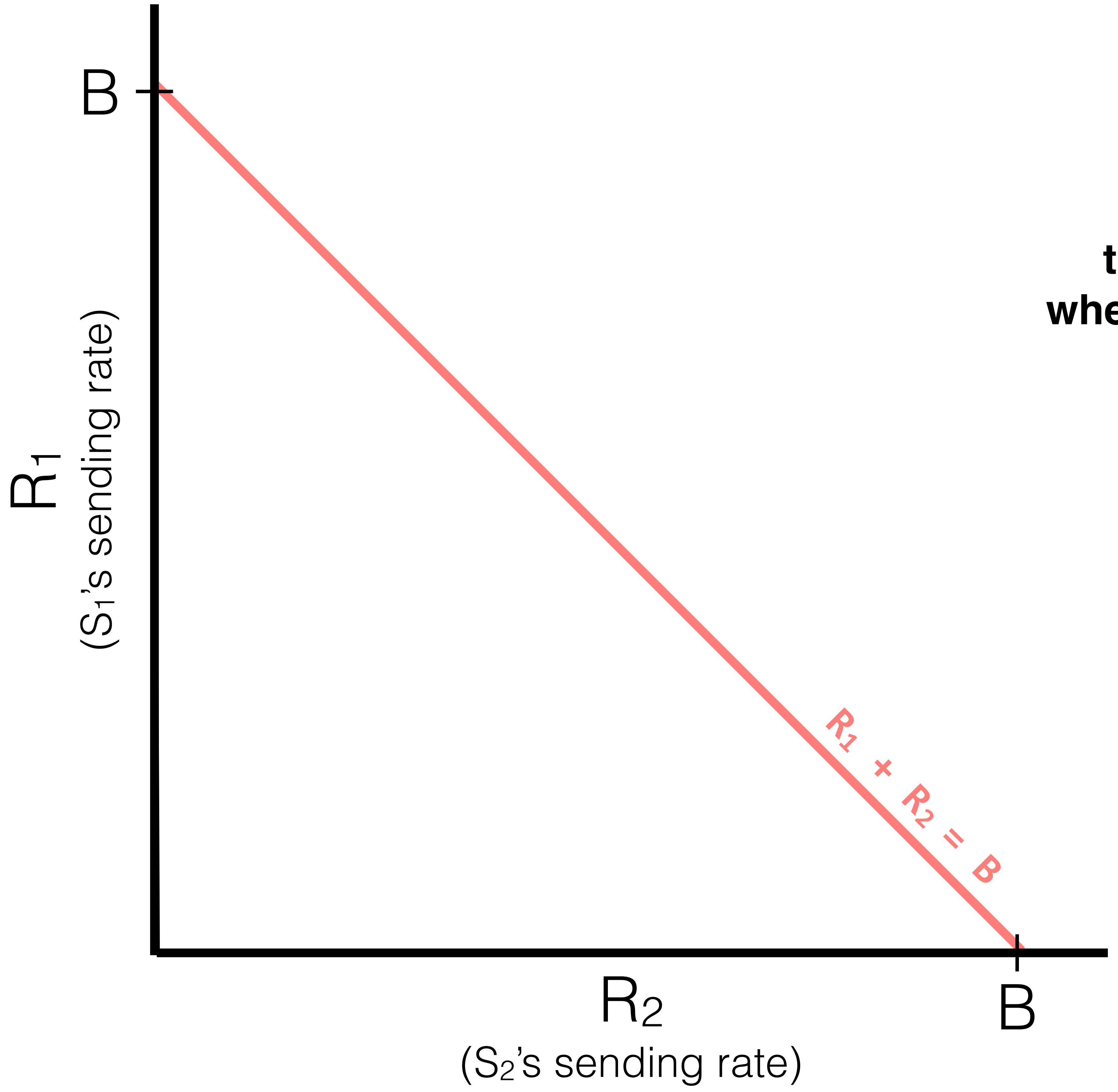
the network is fully utilized when the bottleneck link is "full"

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

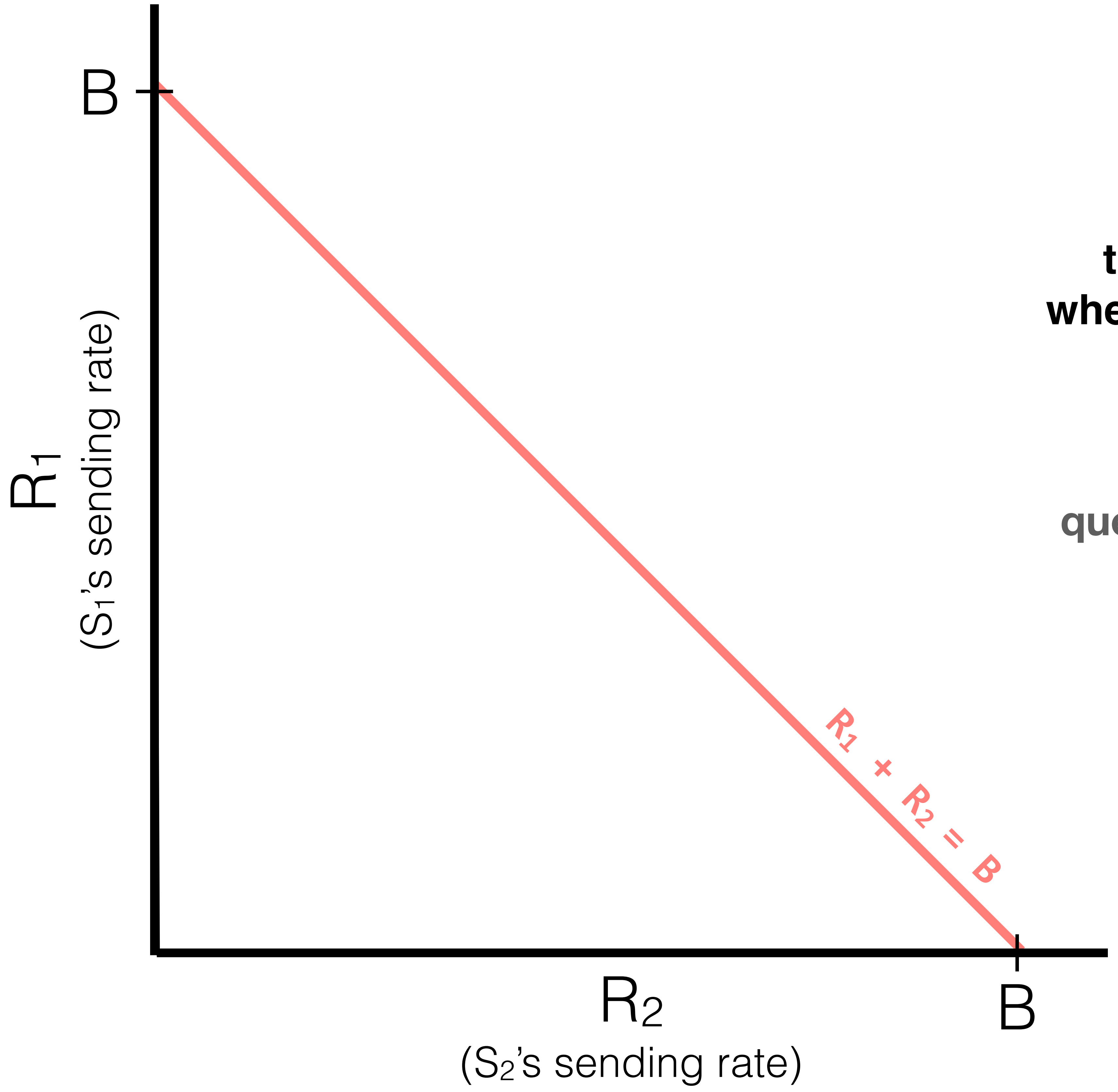
fairness

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency

(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

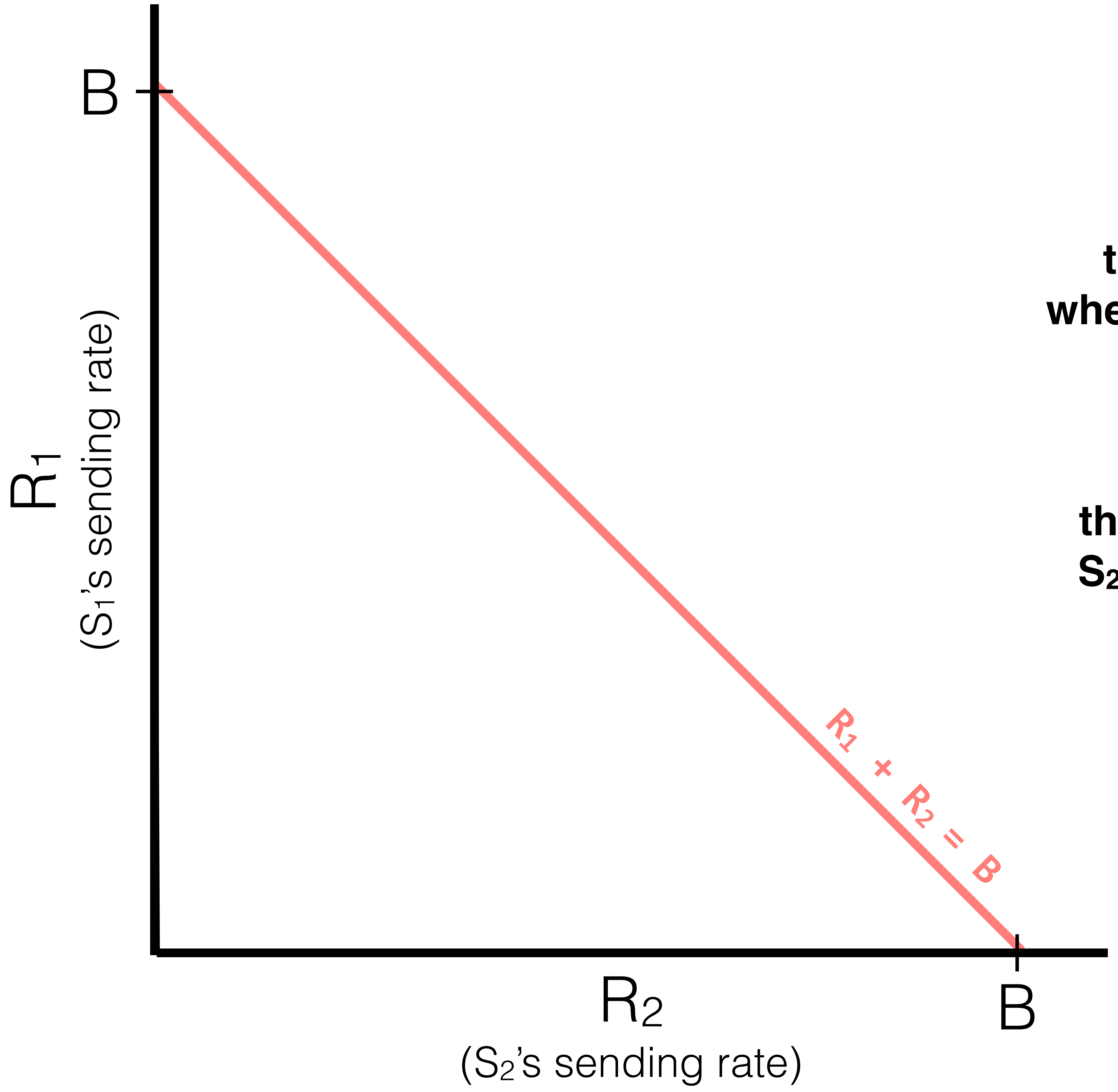
question: what line on this graph would represent fairness?

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

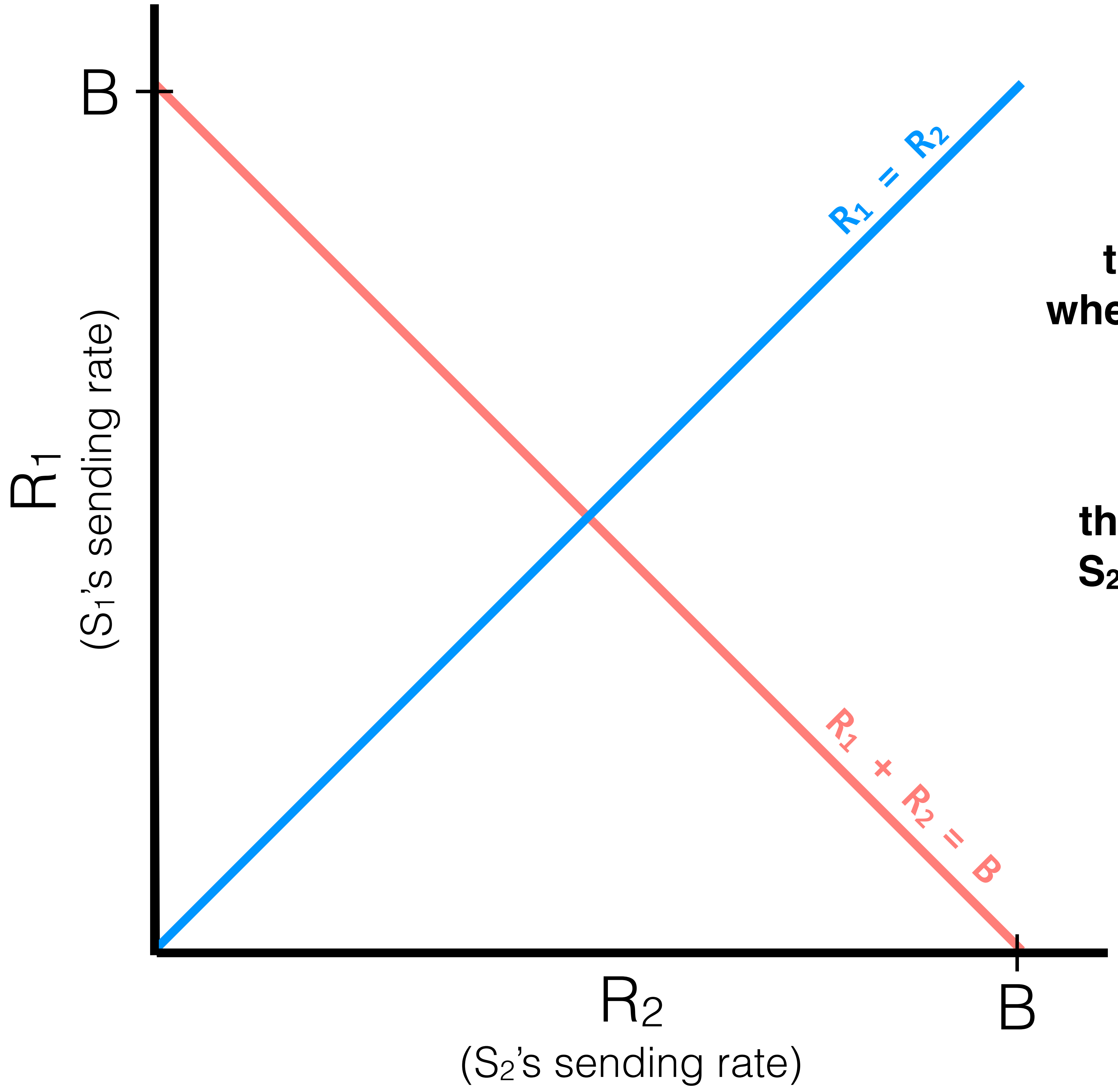
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

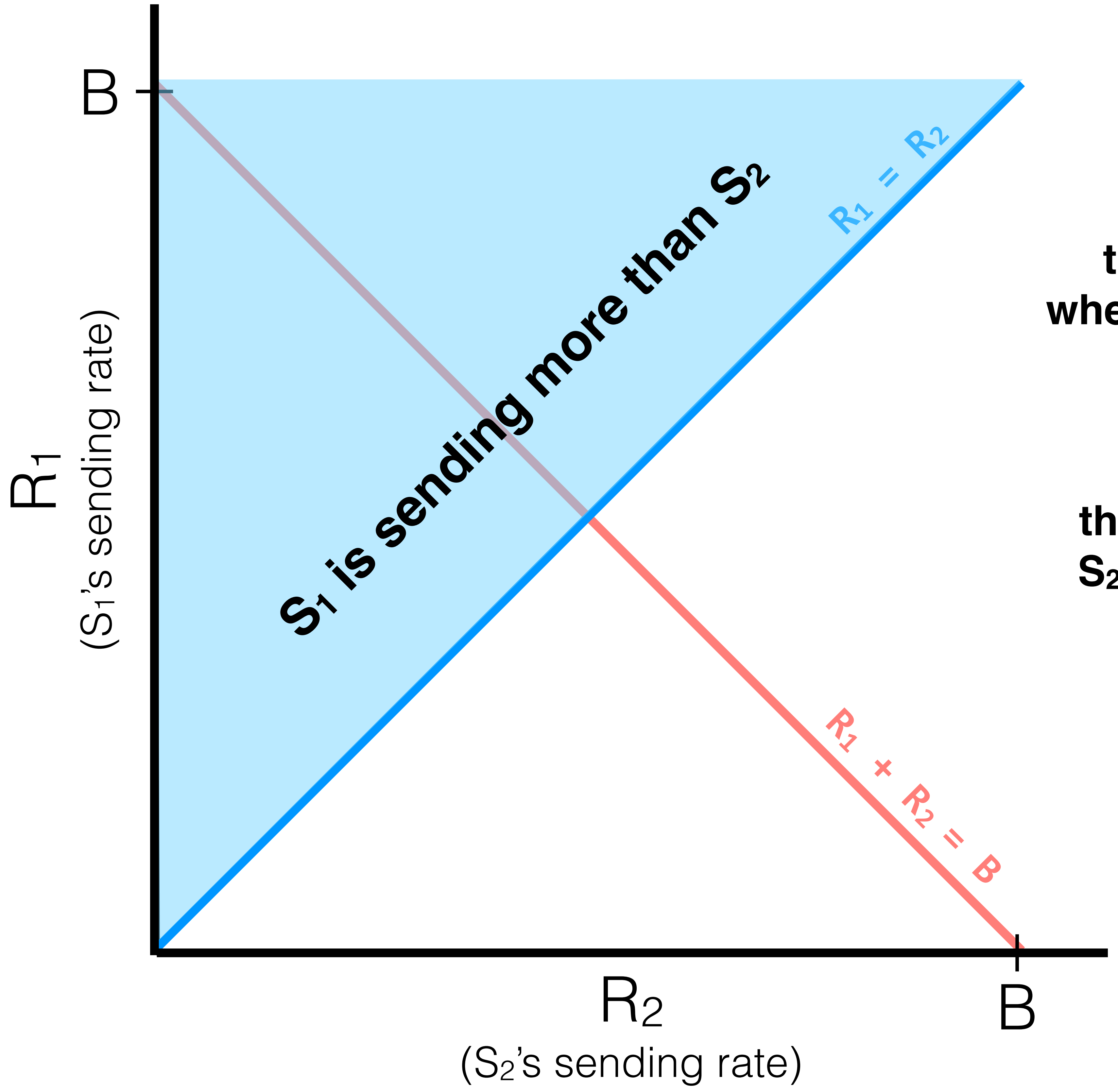
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

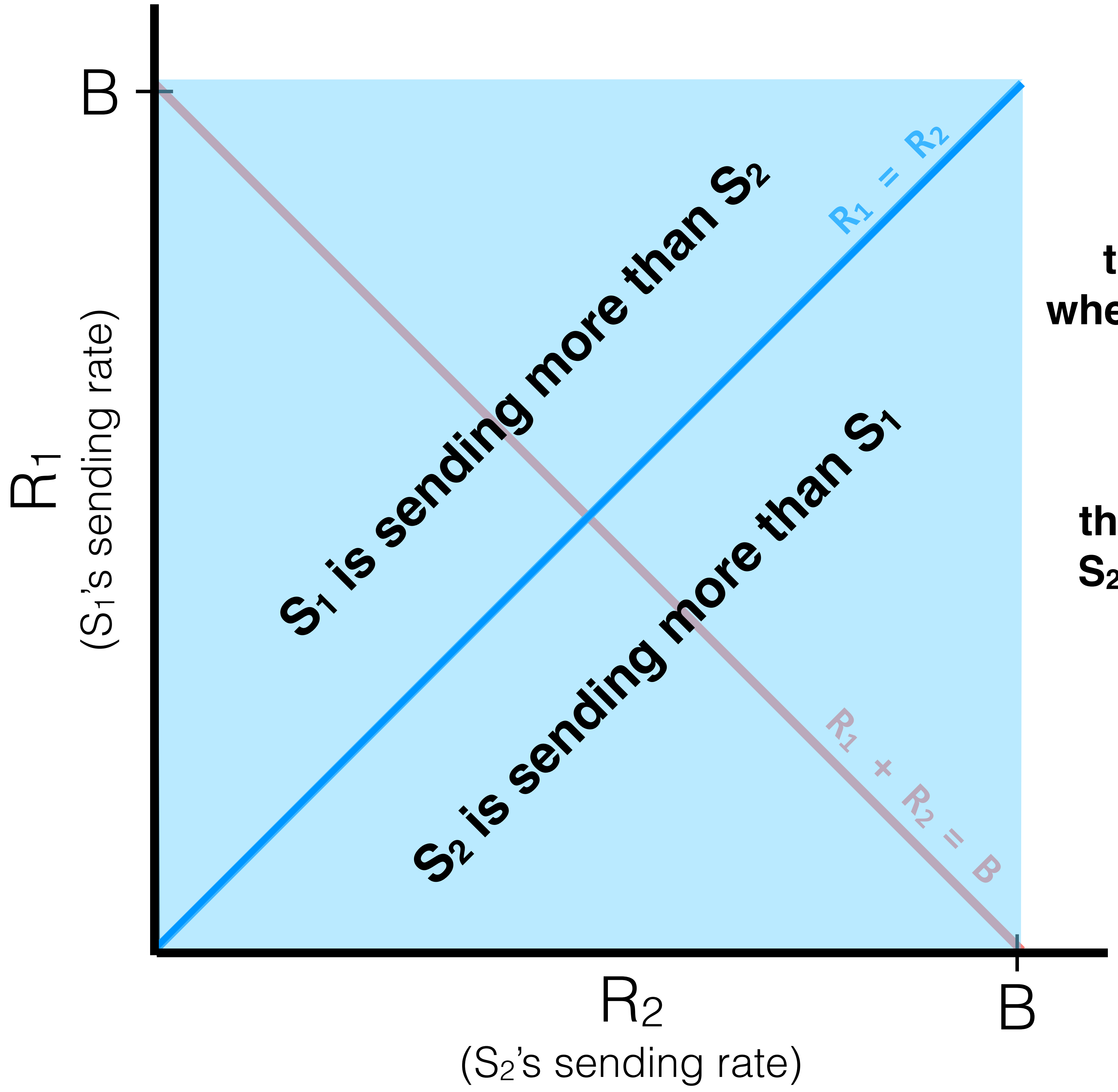
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

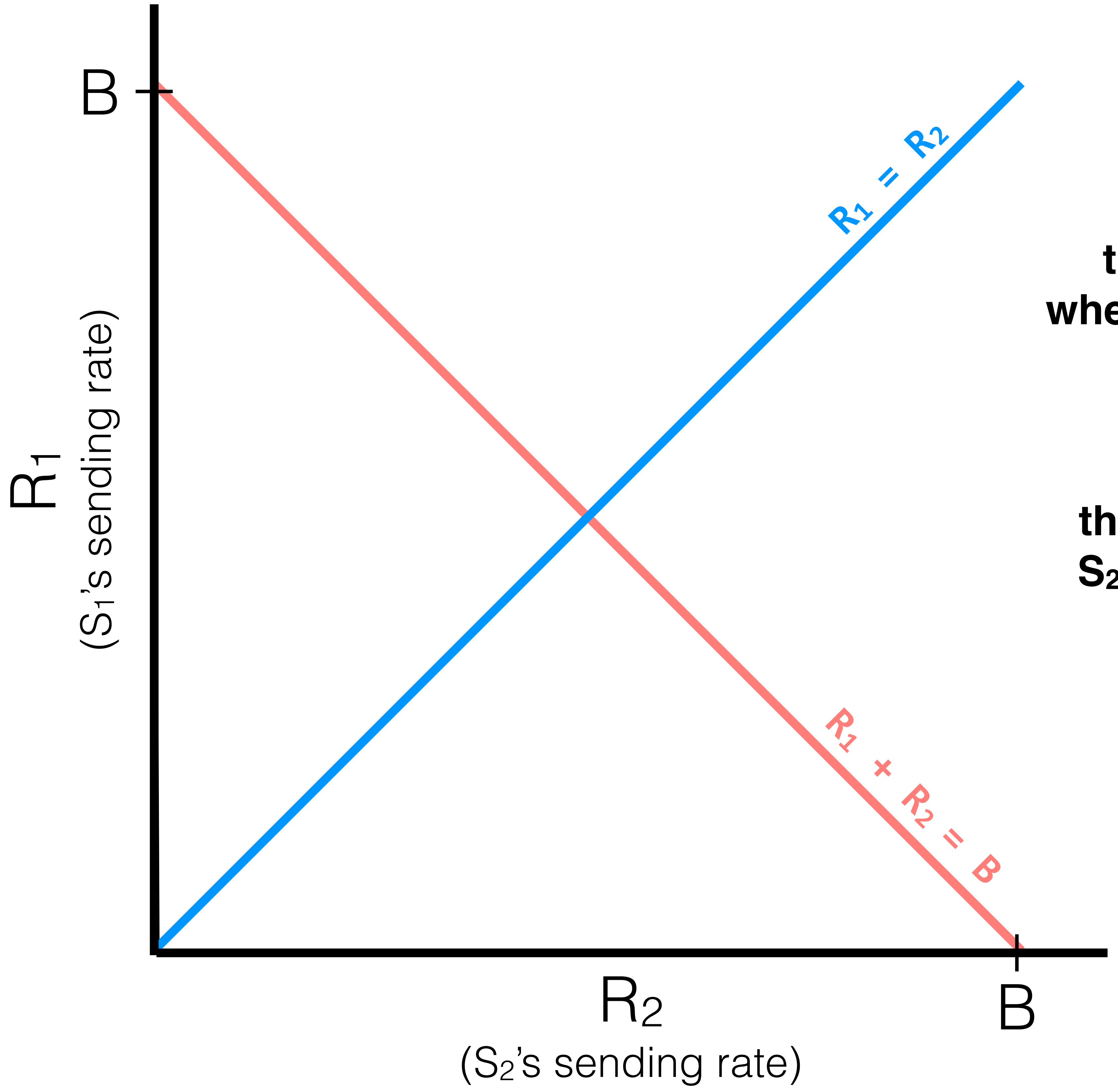
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

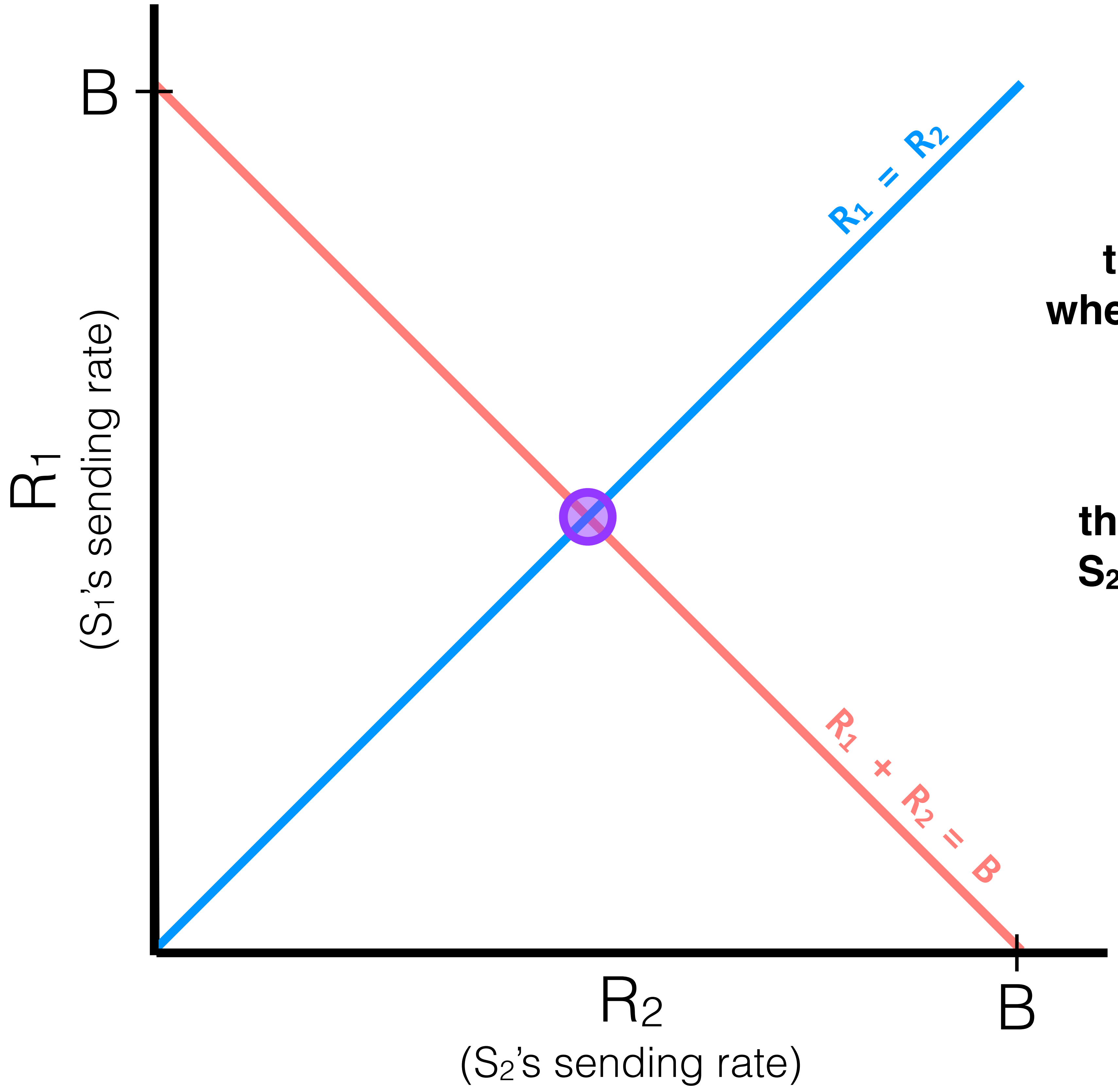
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

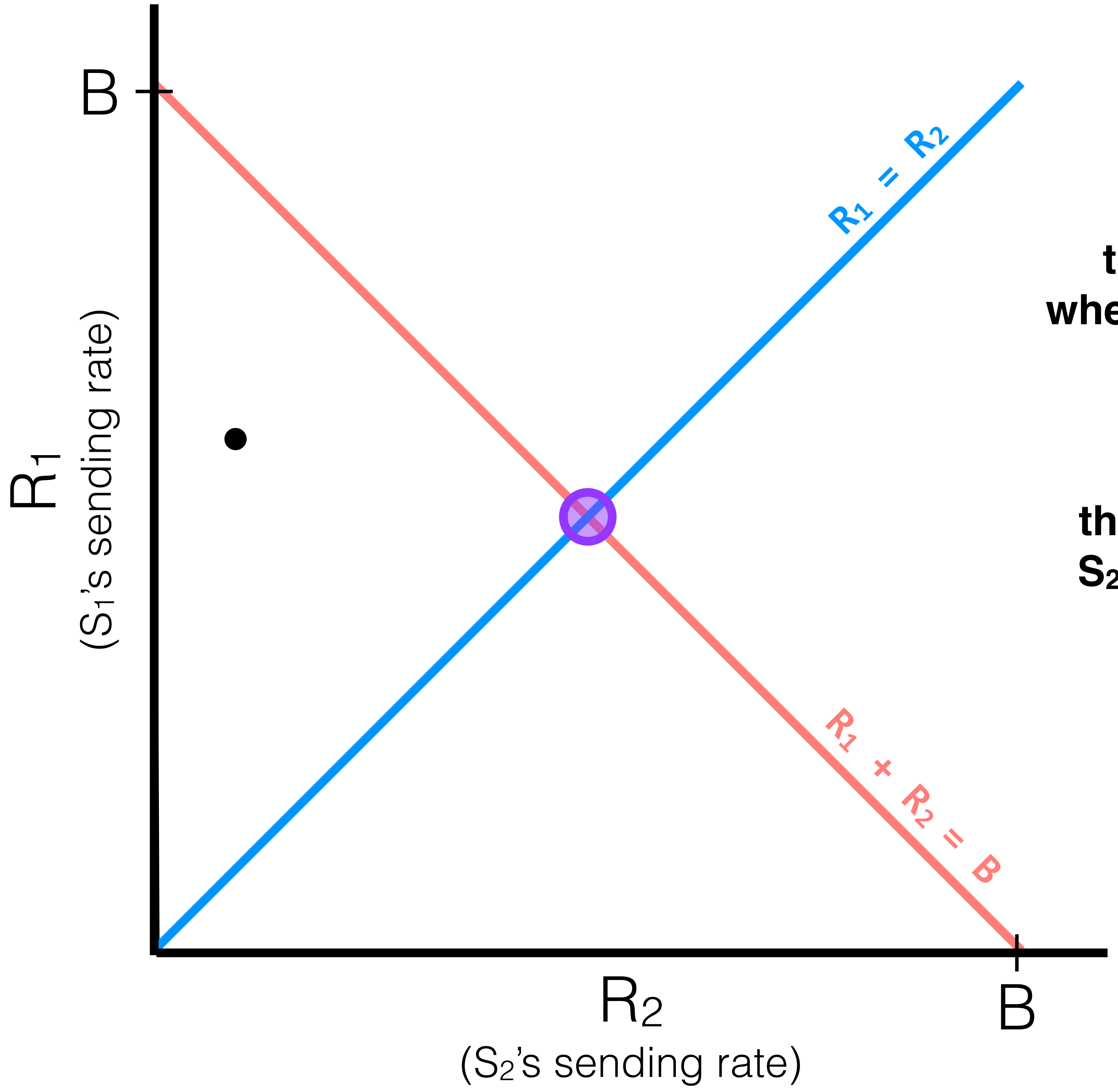
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

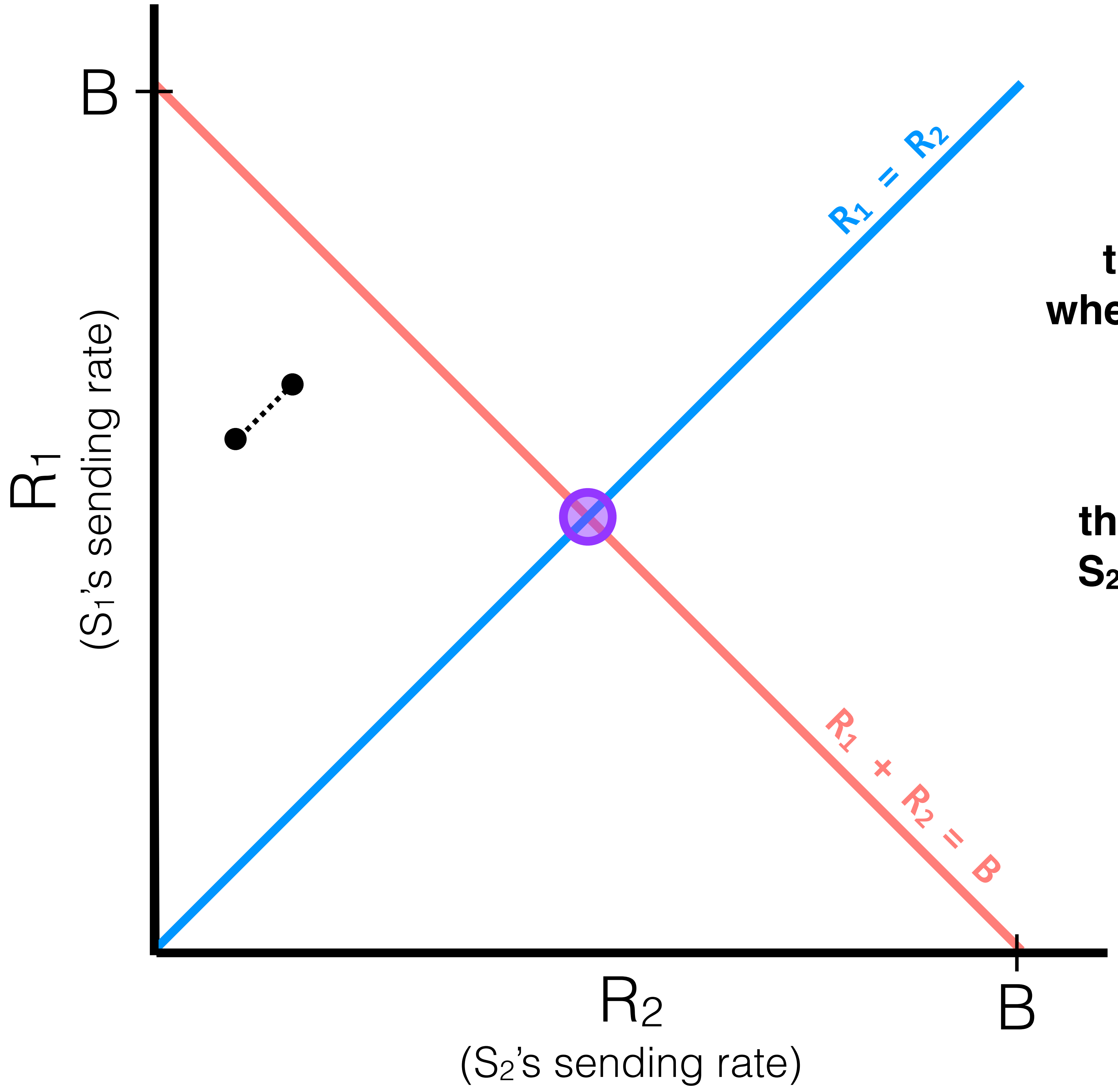
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

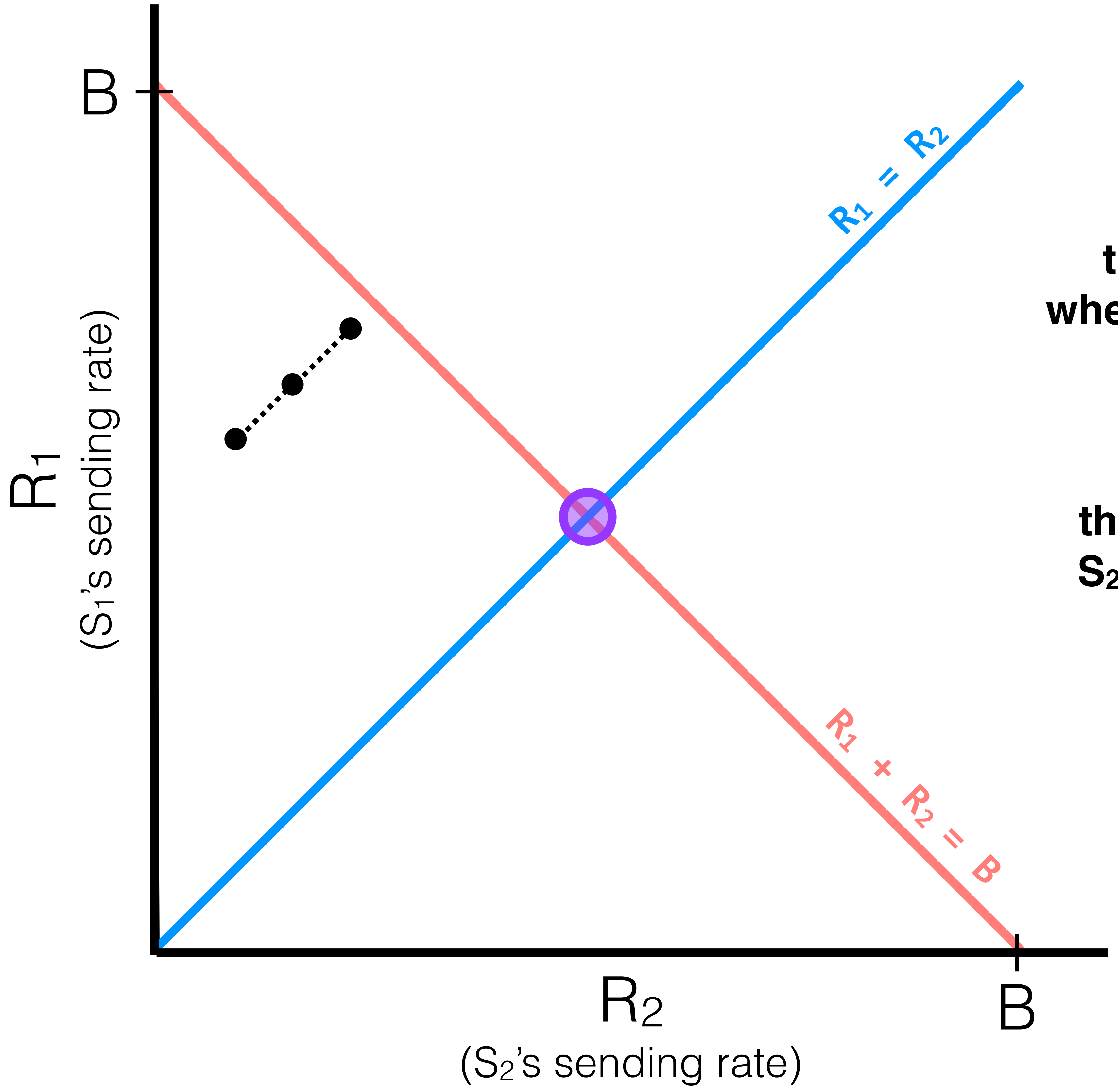
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

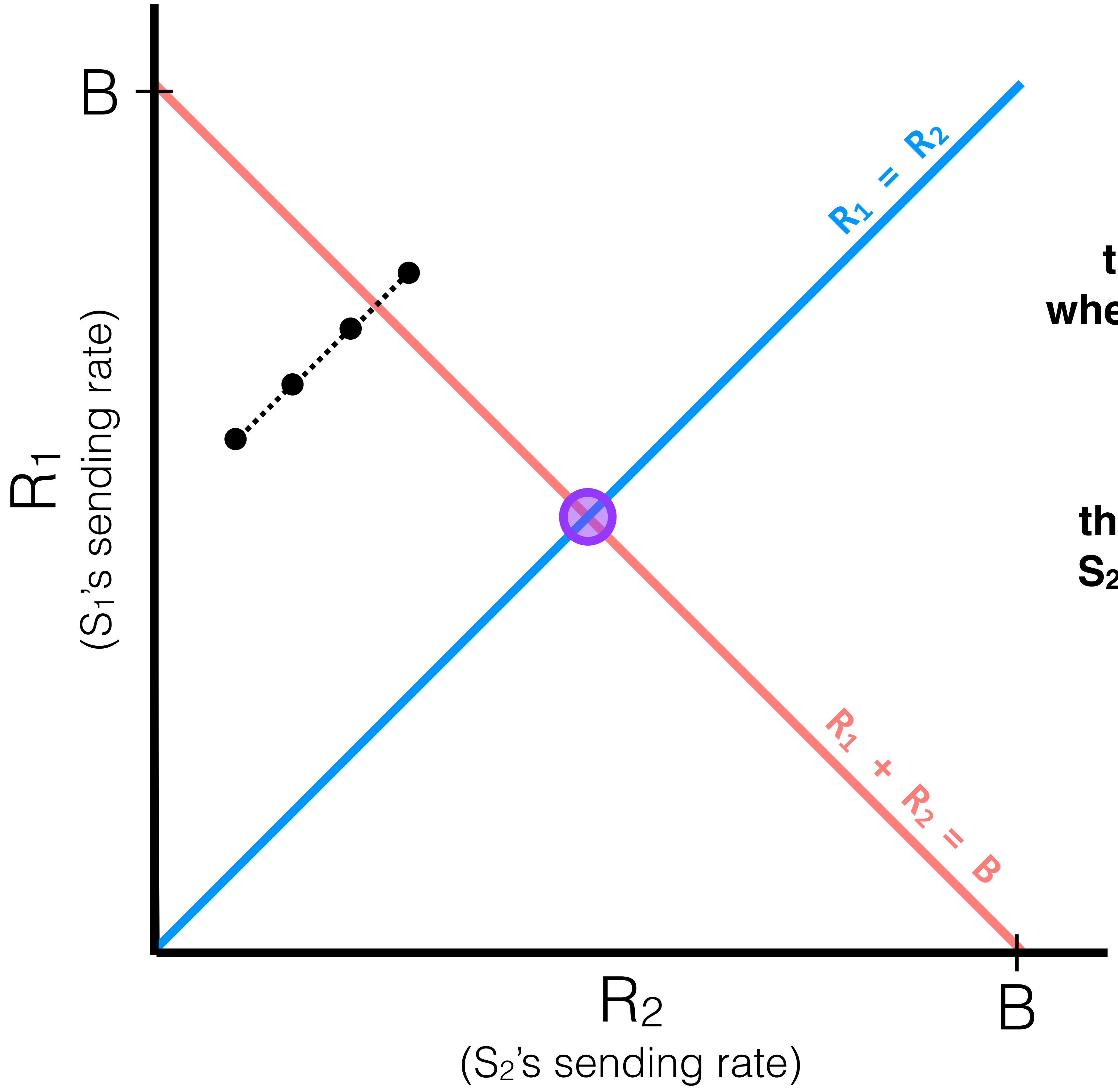
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

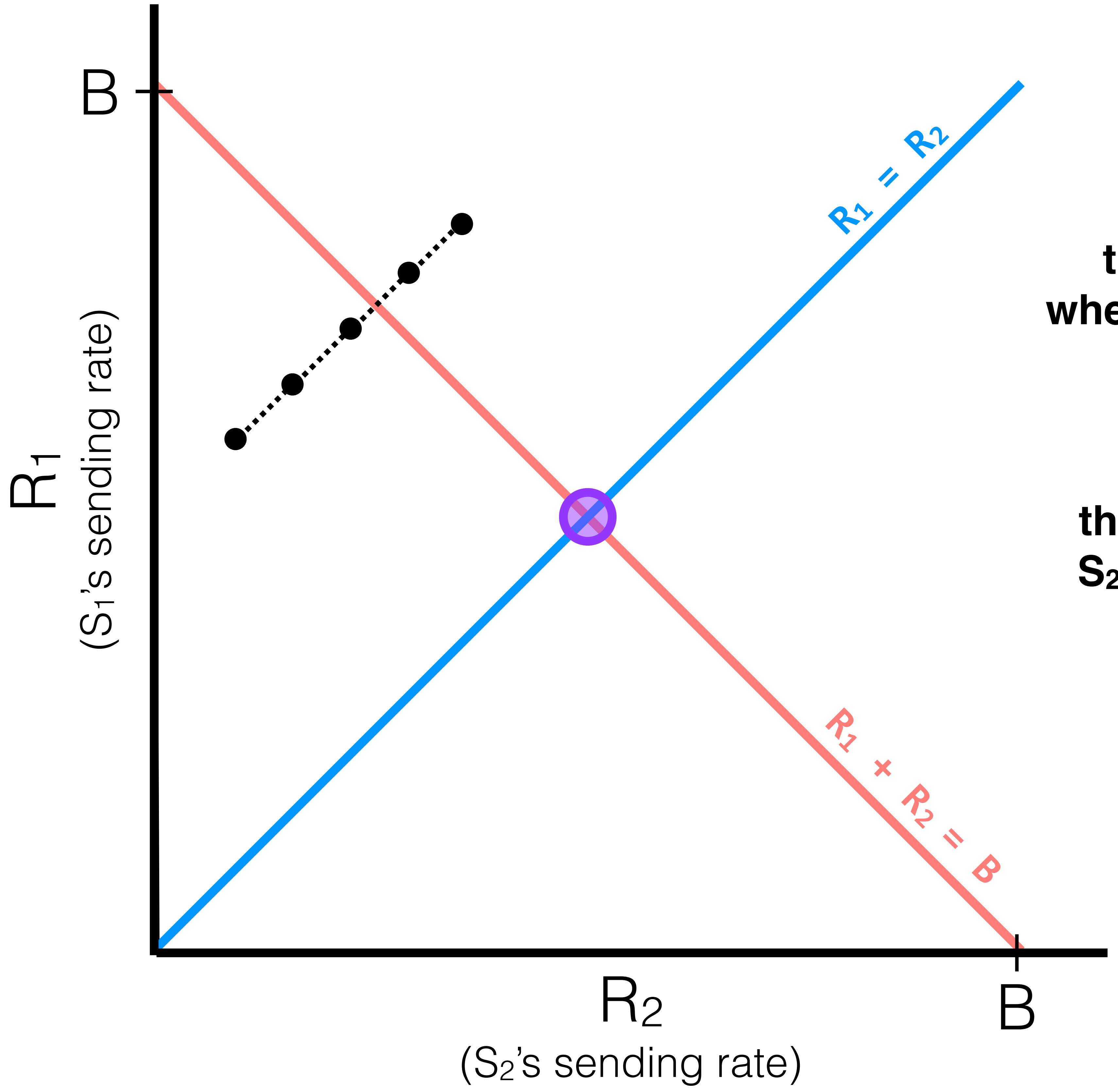
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

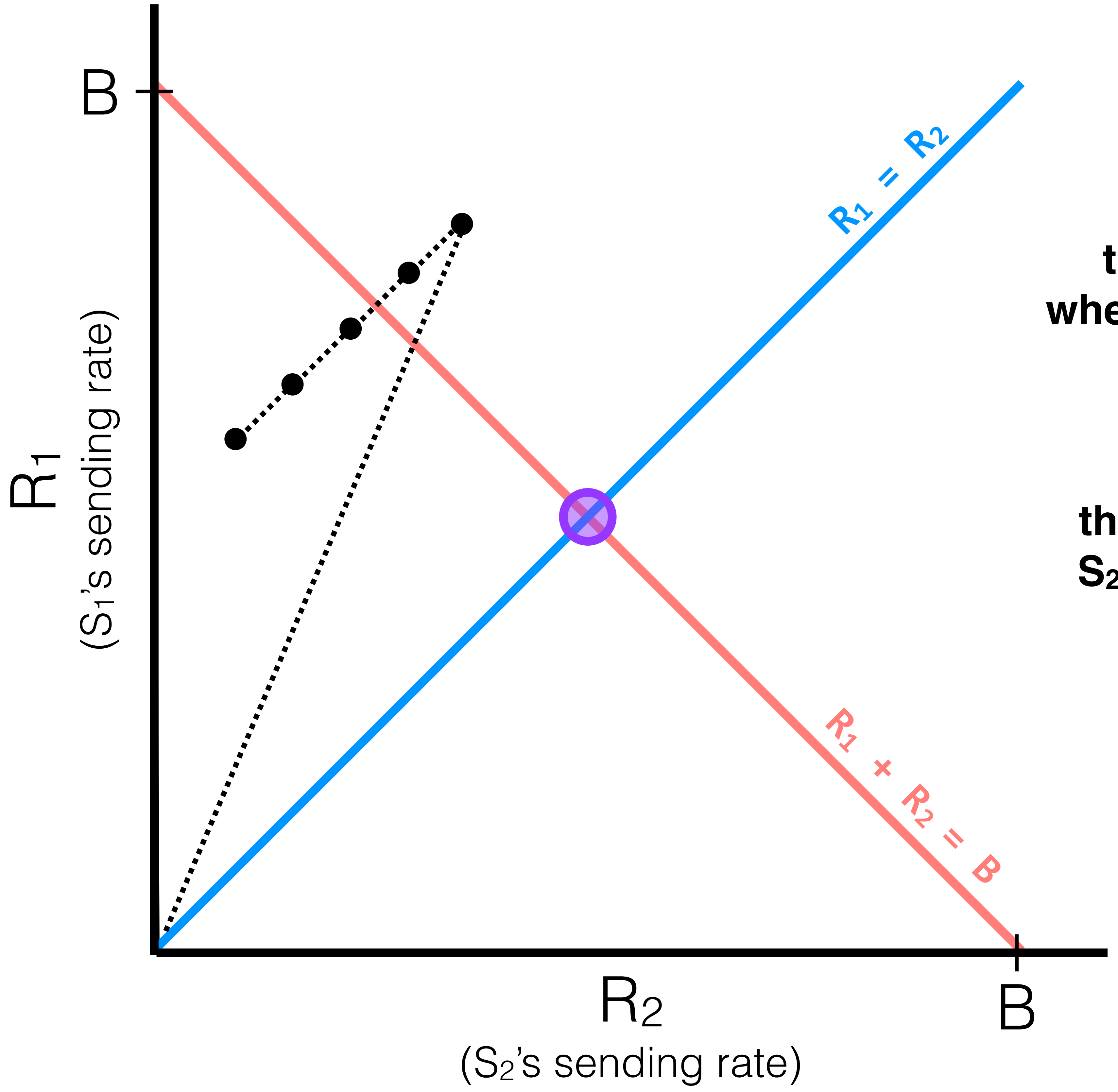
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

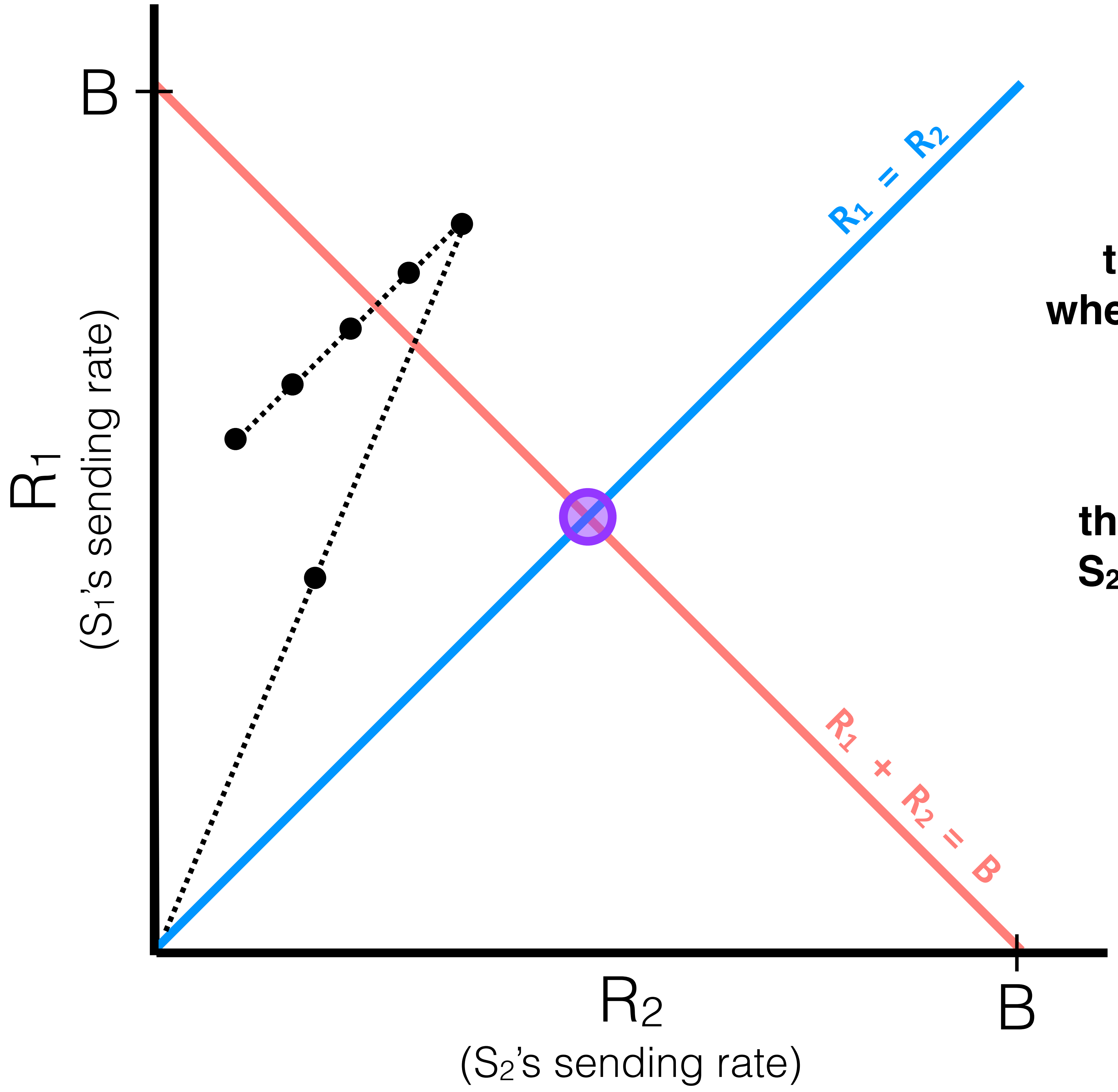
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

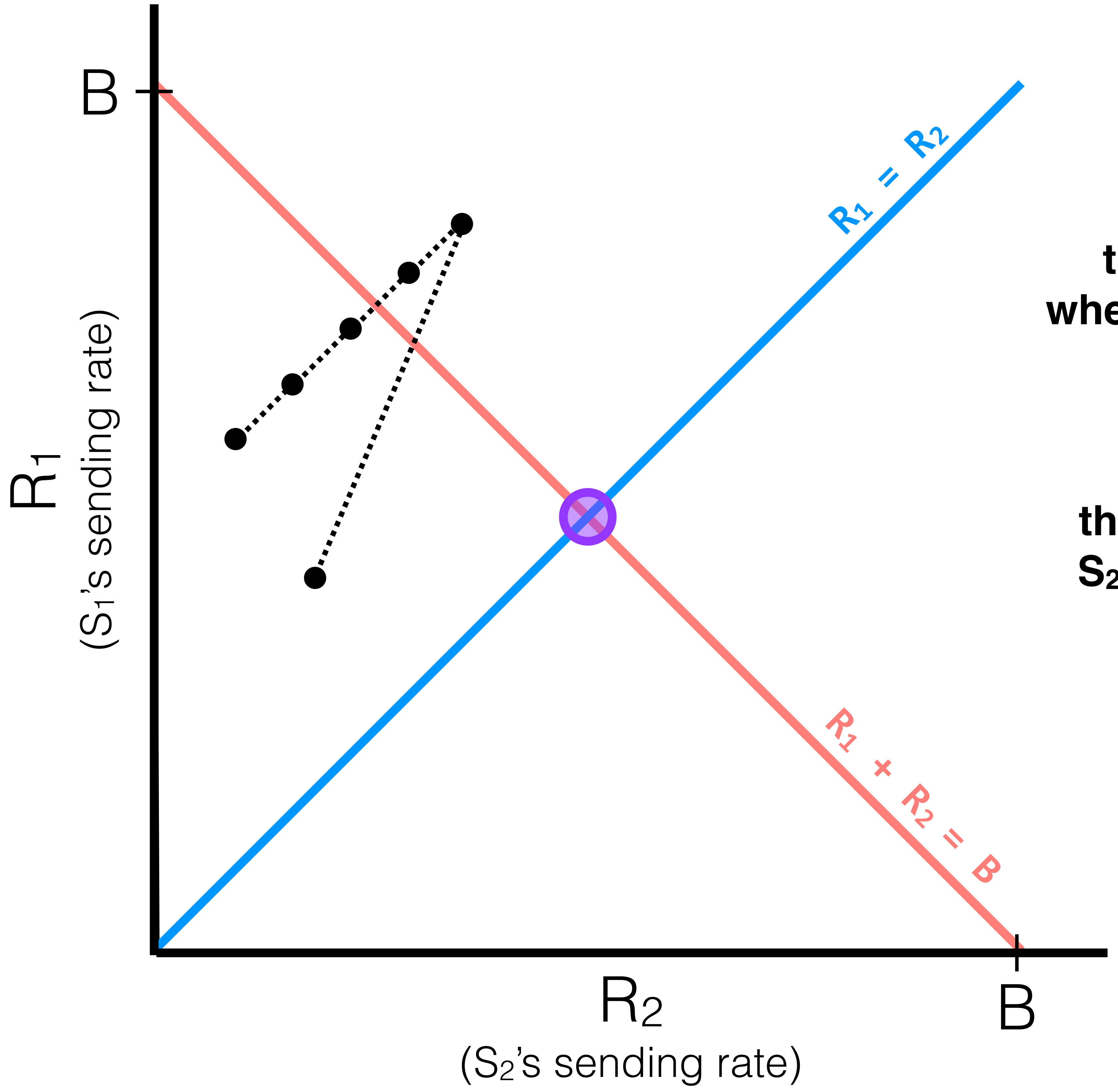
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

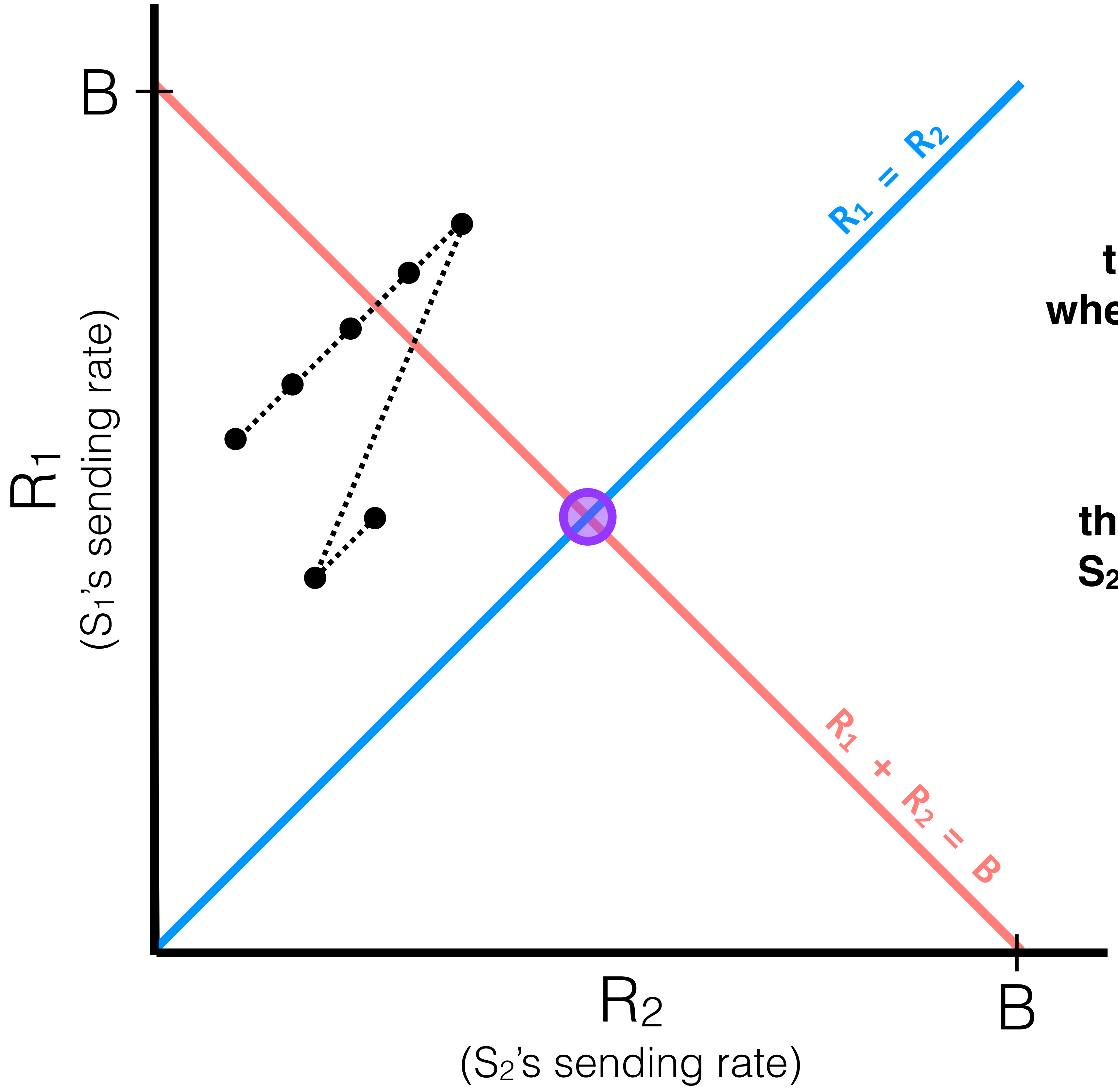
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

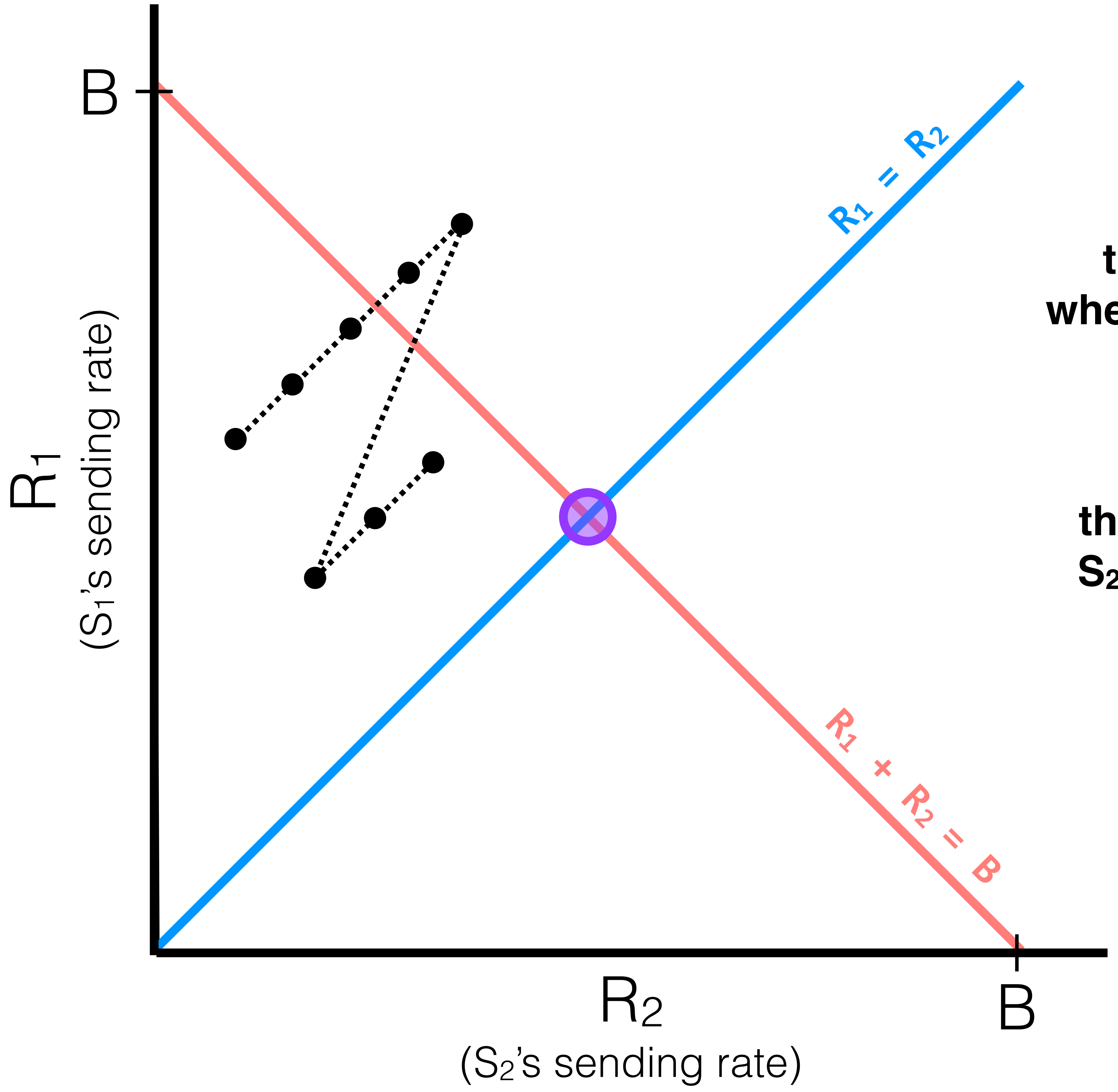
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

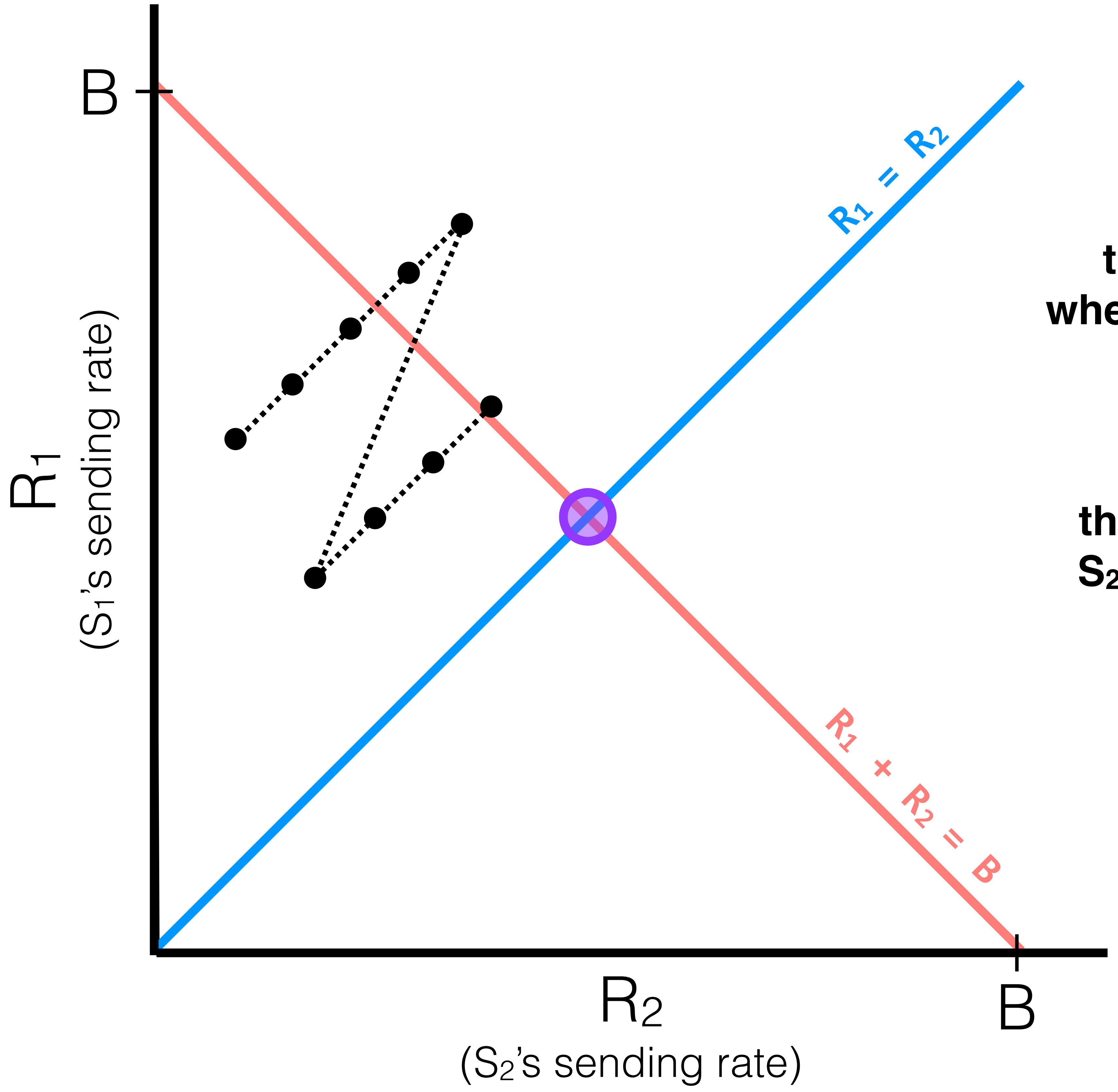
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

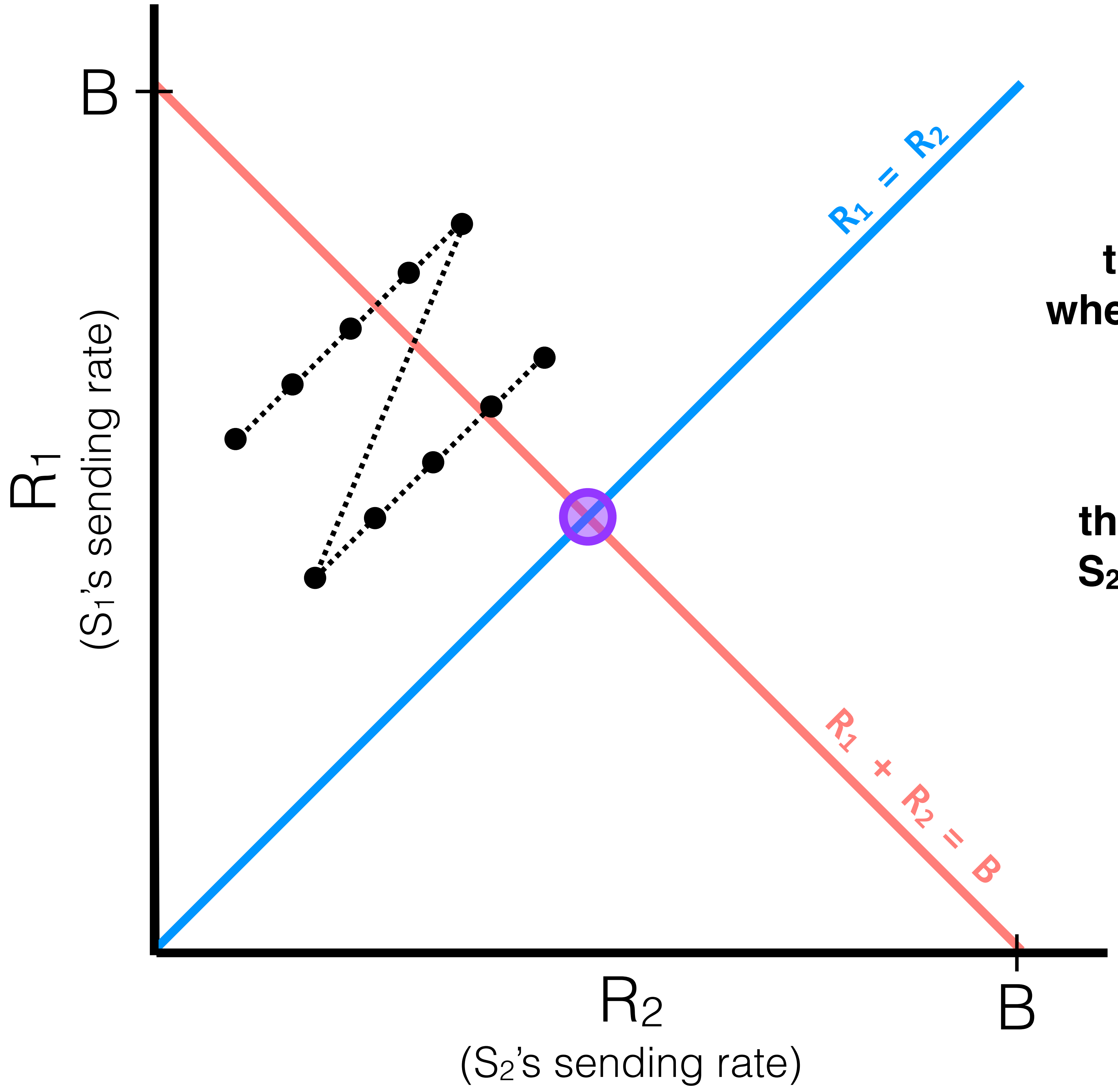
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

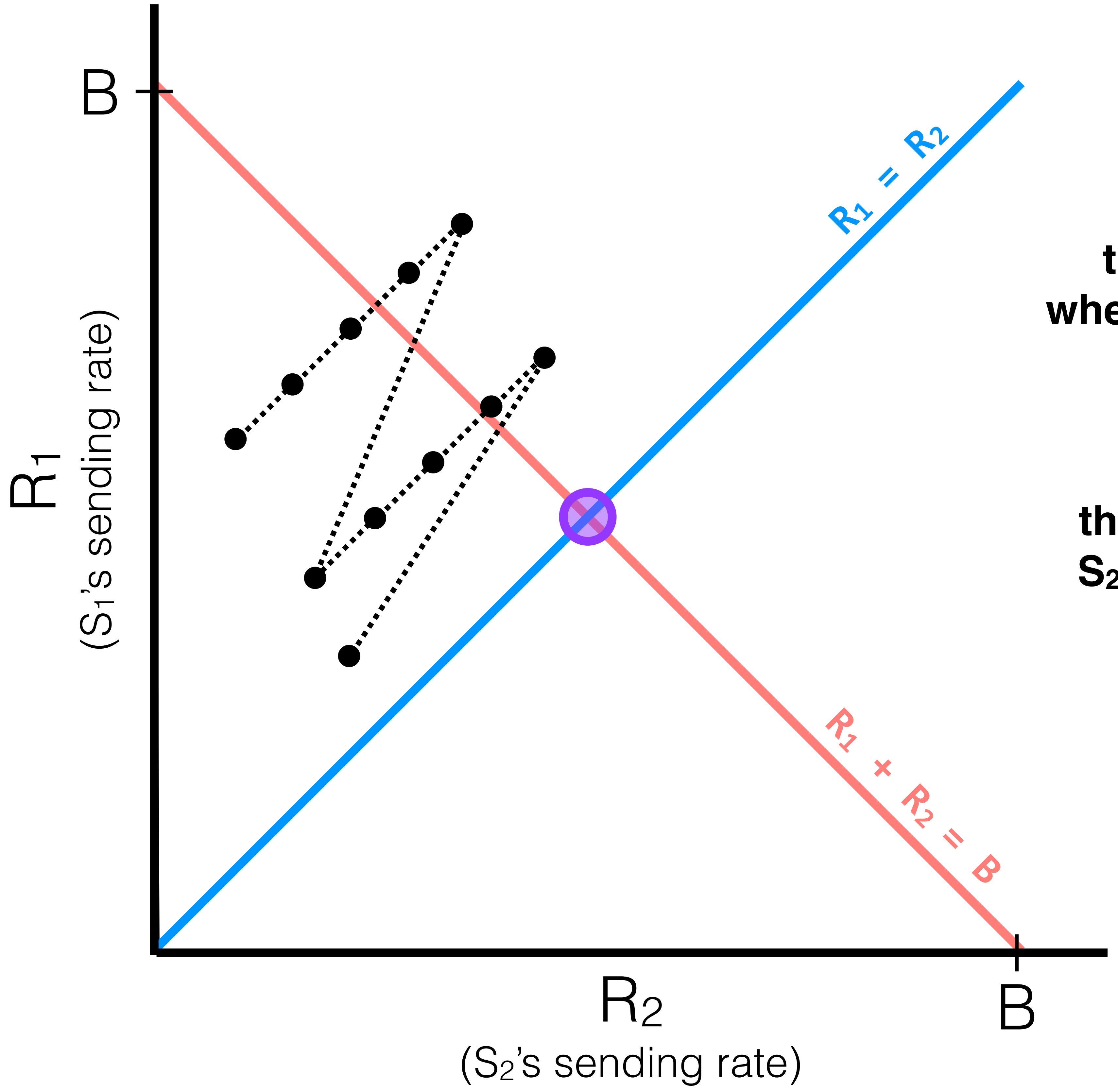
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

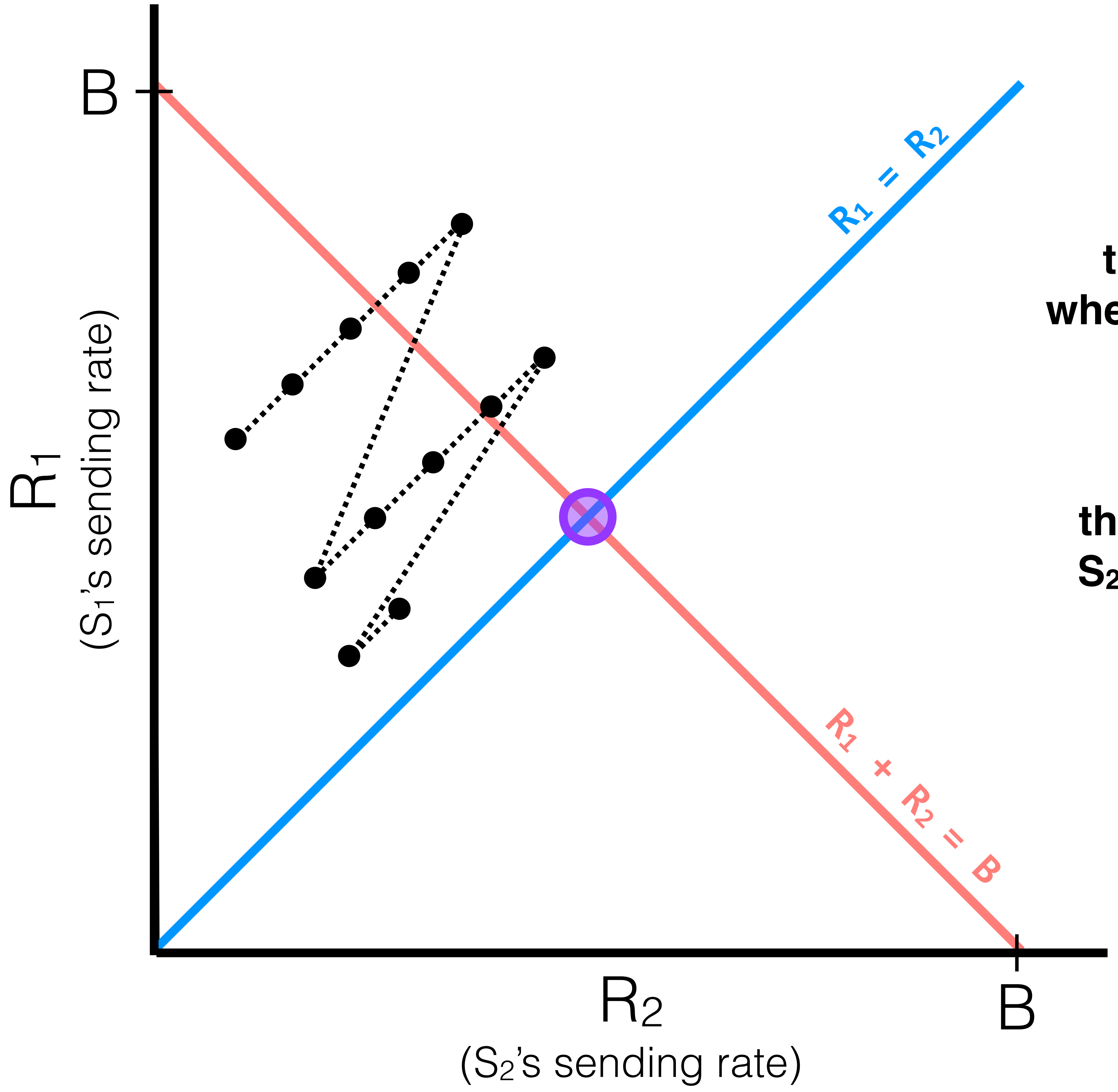
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

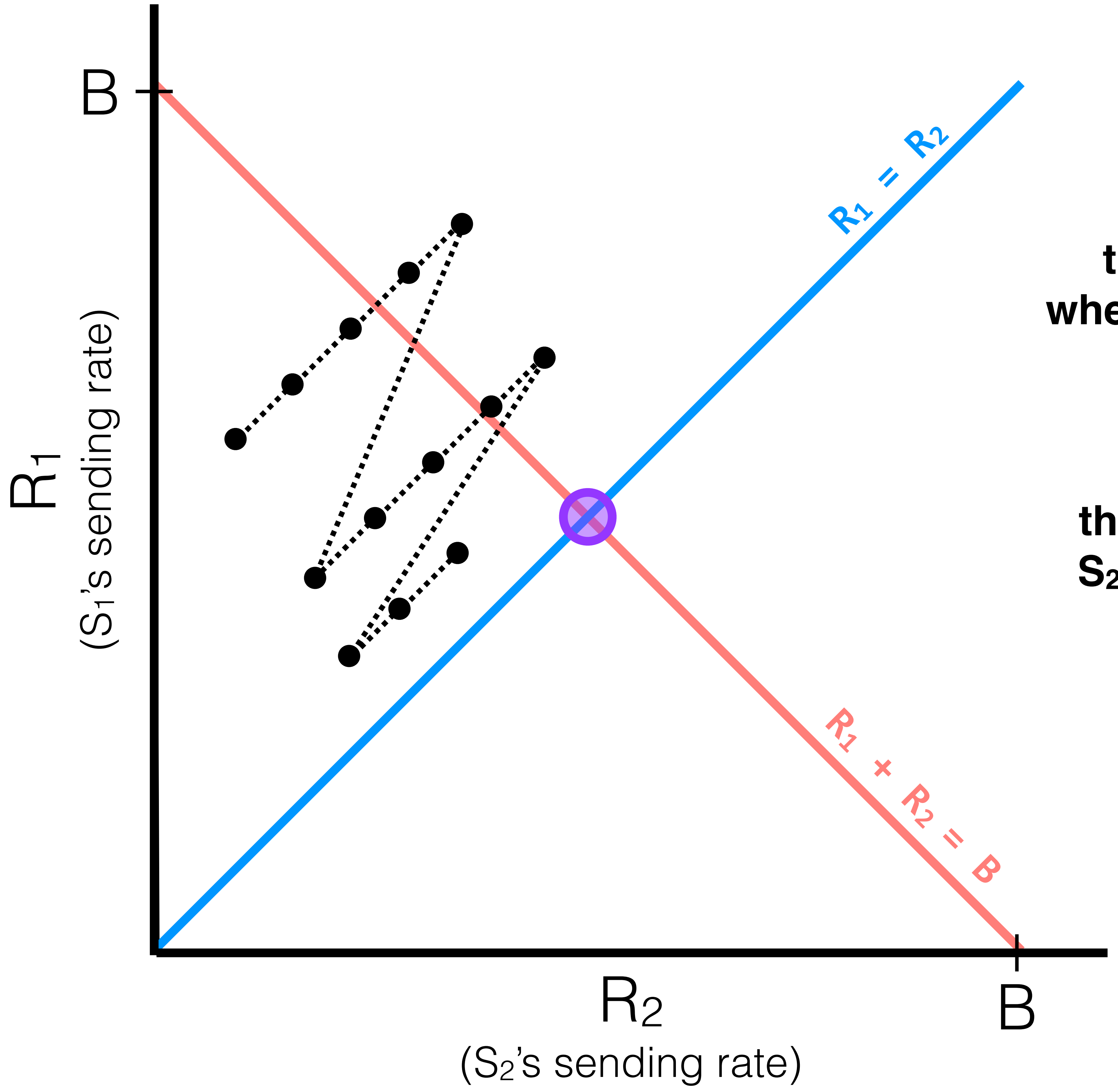
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

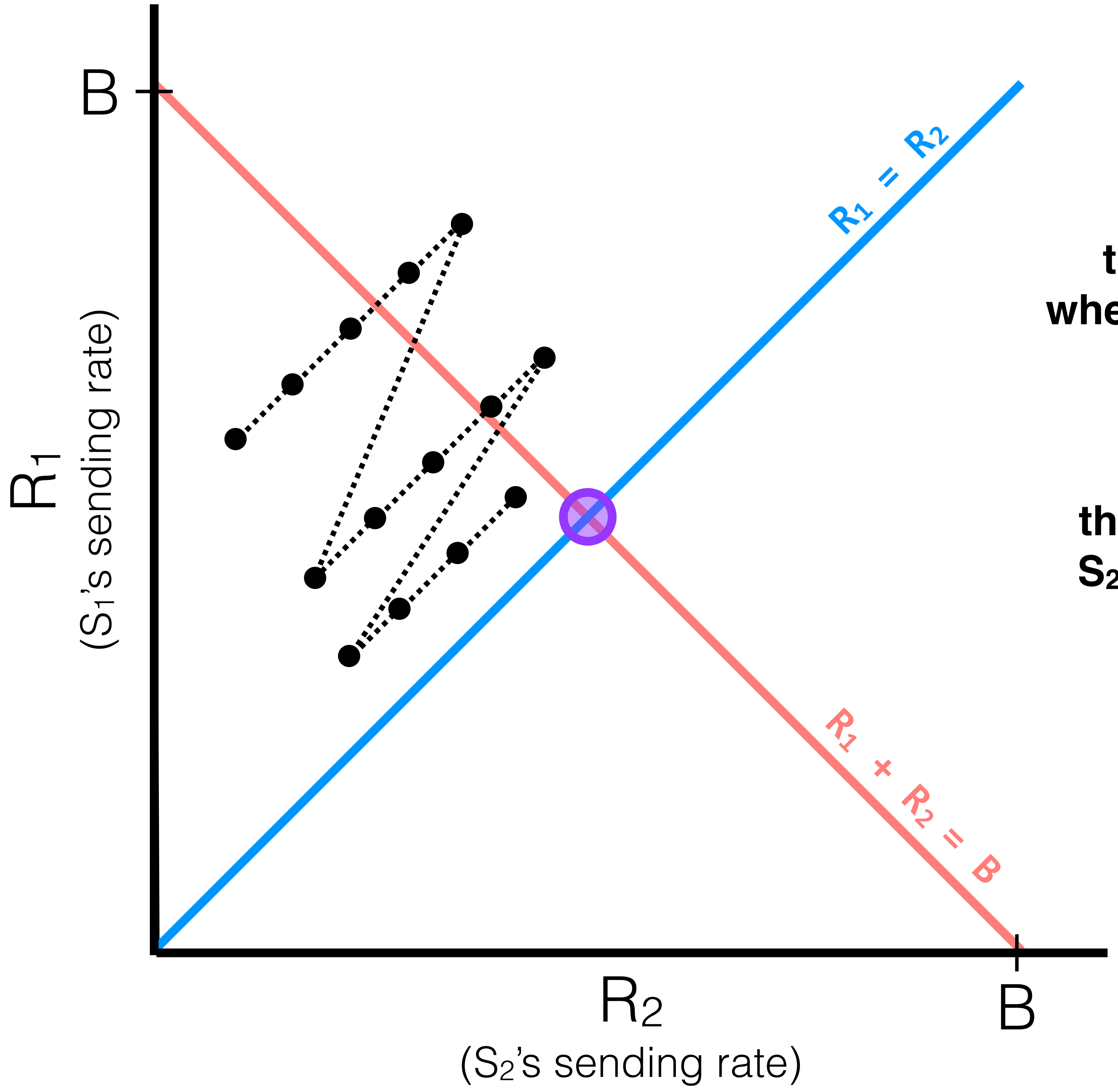
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

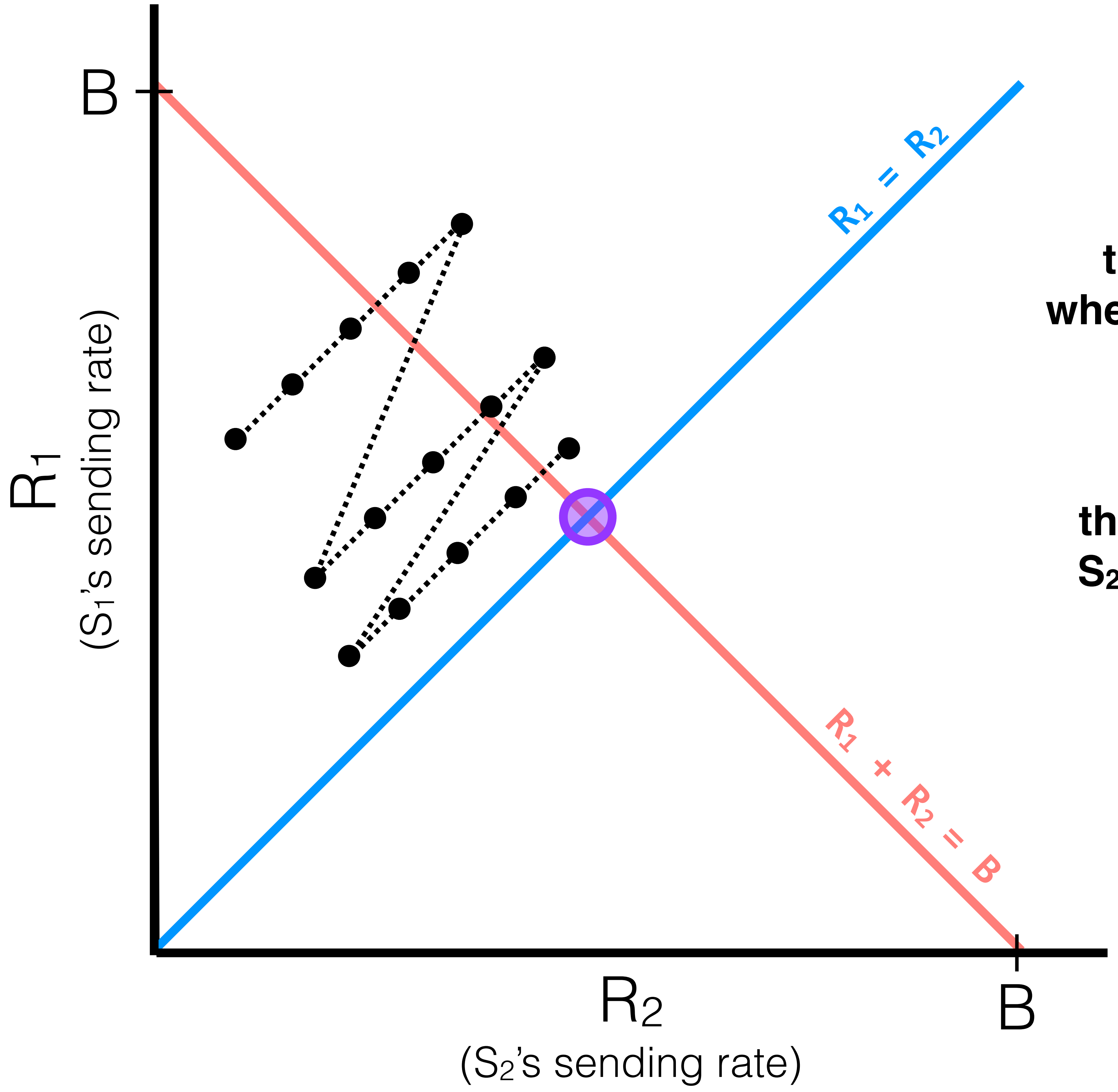
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

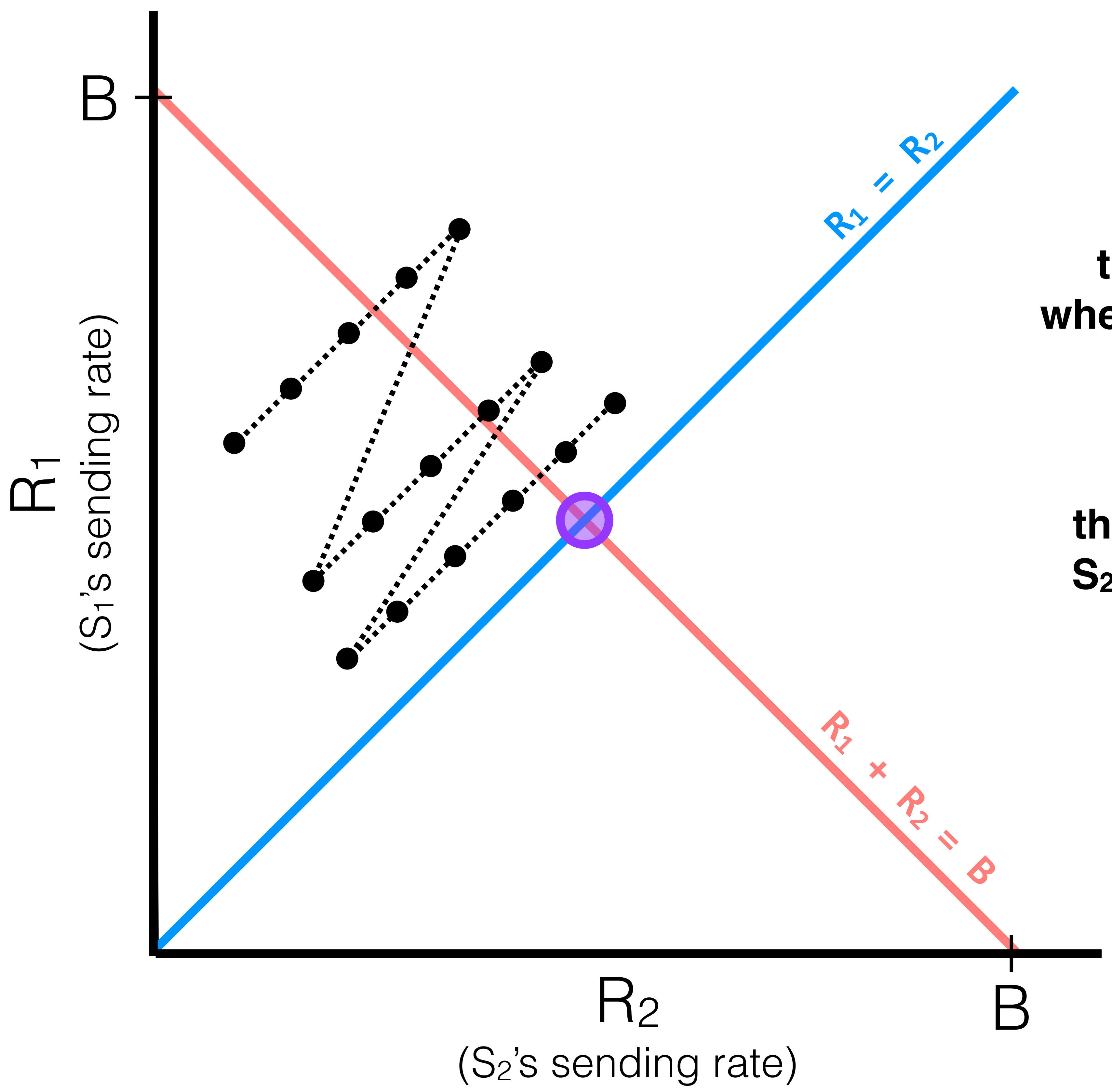
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

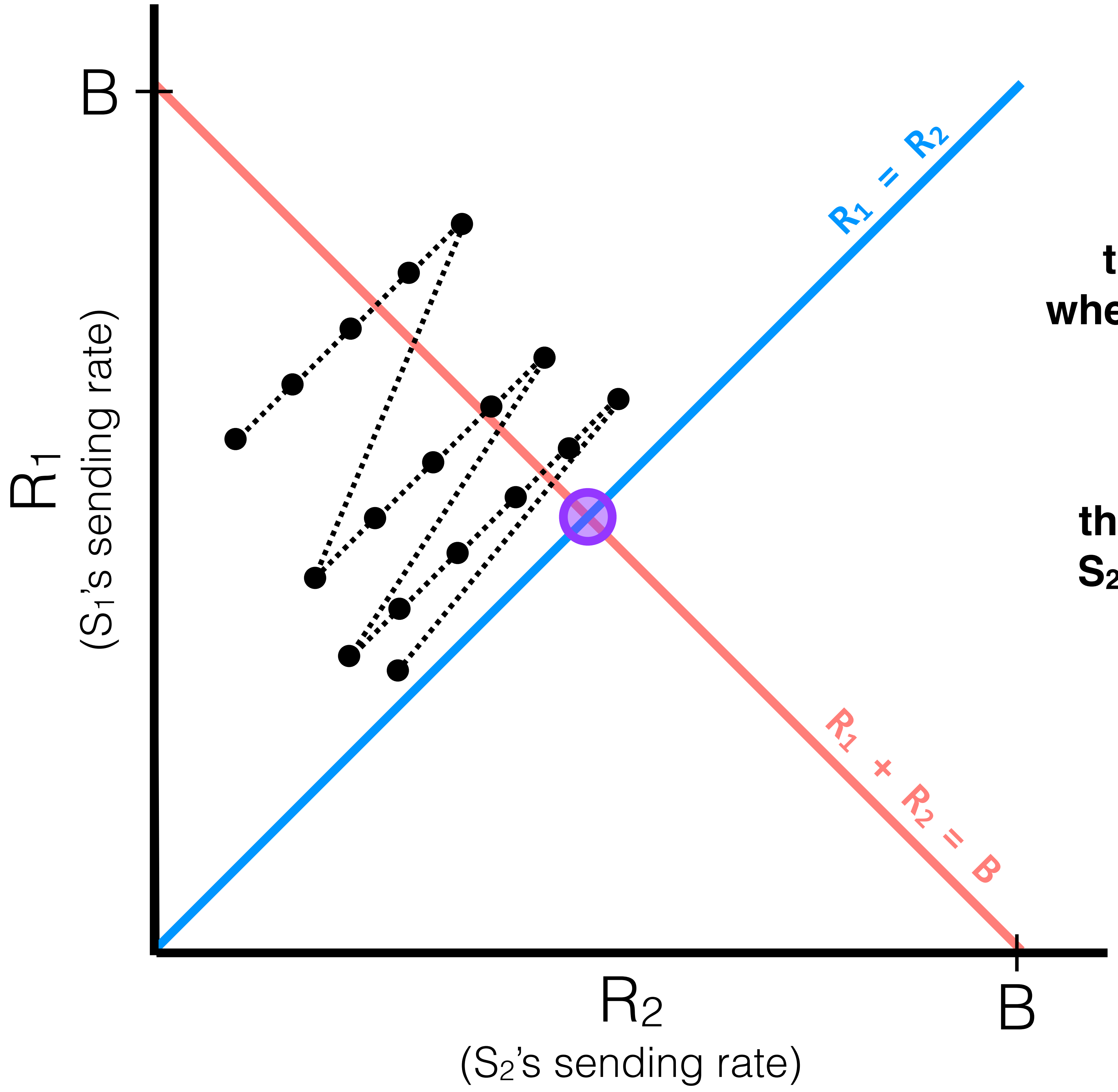
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

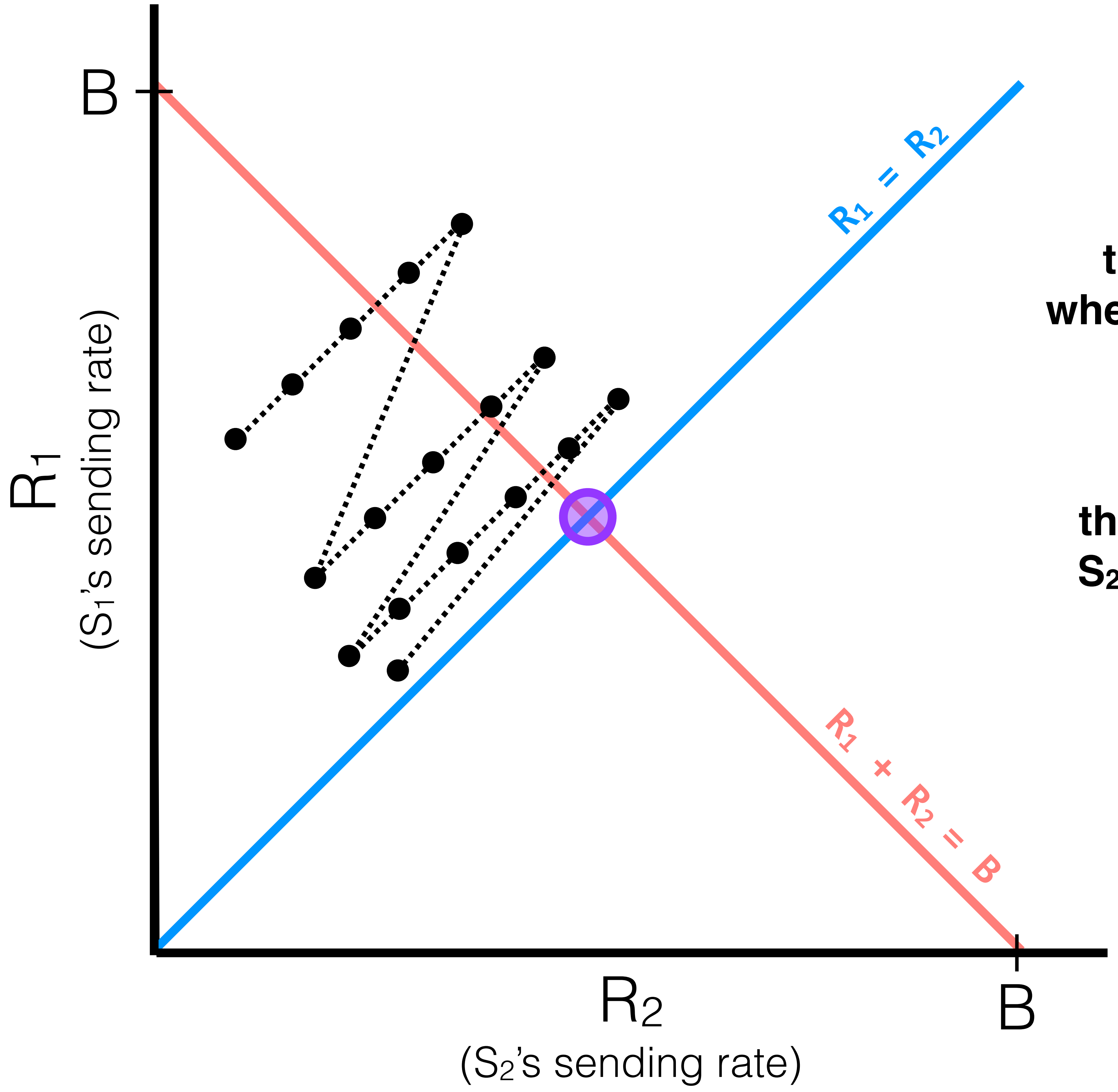
the network is fair when S_1 and S_2 are sending at the same rate

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency
(utilization)

the network is fully utilized when the bottleneck link is "full"

fairness

the network is fair when S_1 and S_2 are sending at the same rate

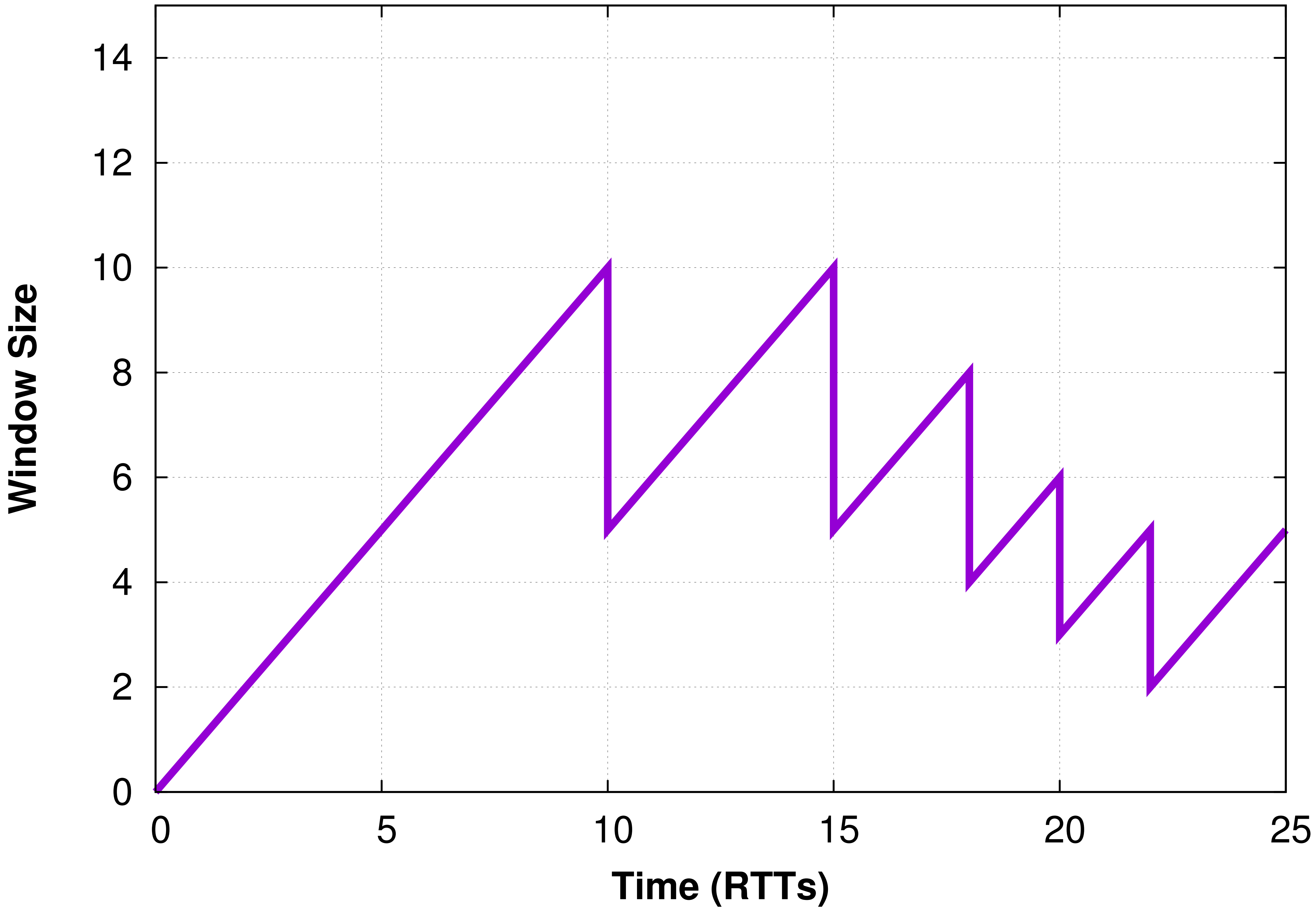
eventually, R1 and R2 will come to oscillate around the fixed point

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

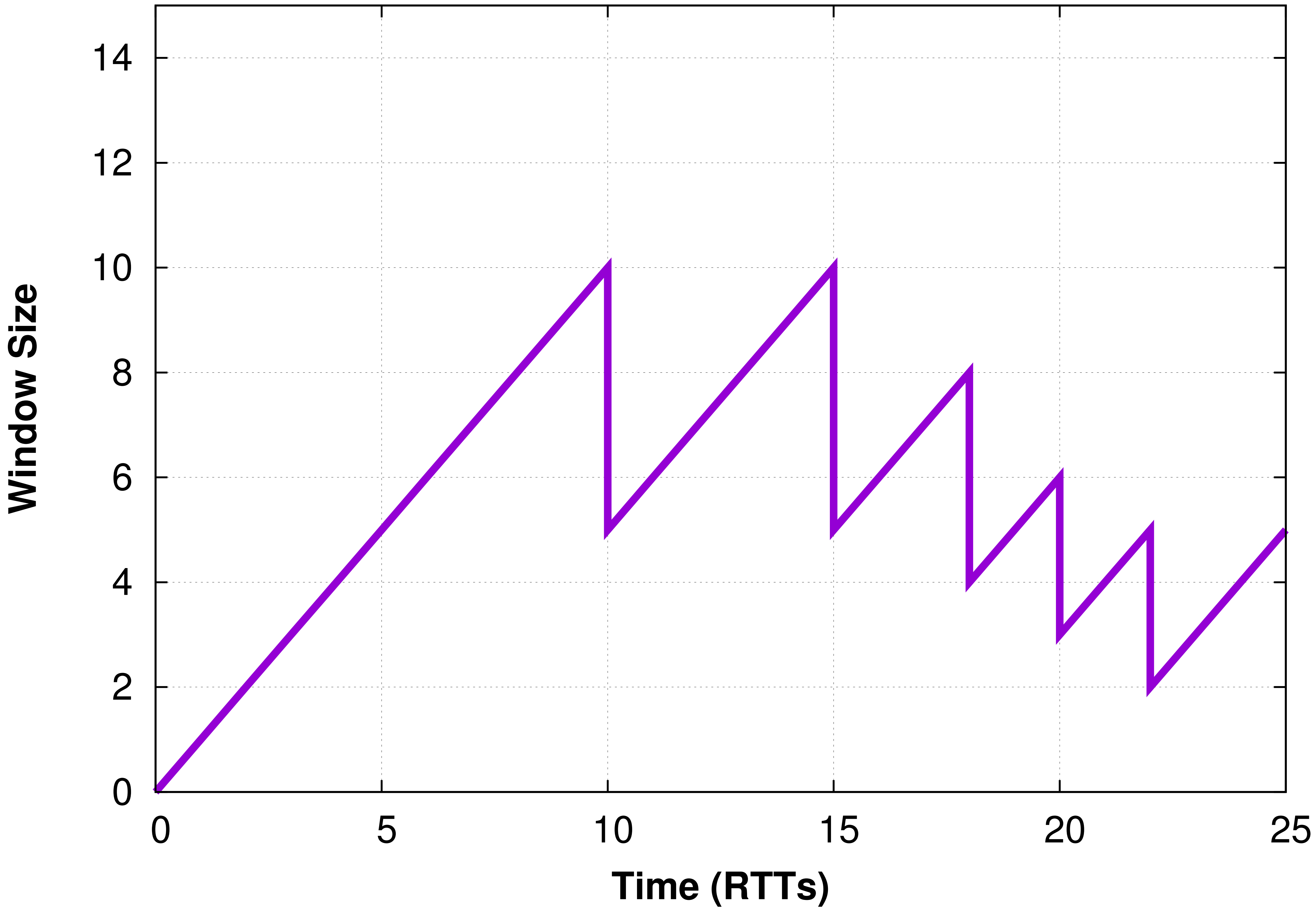


efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



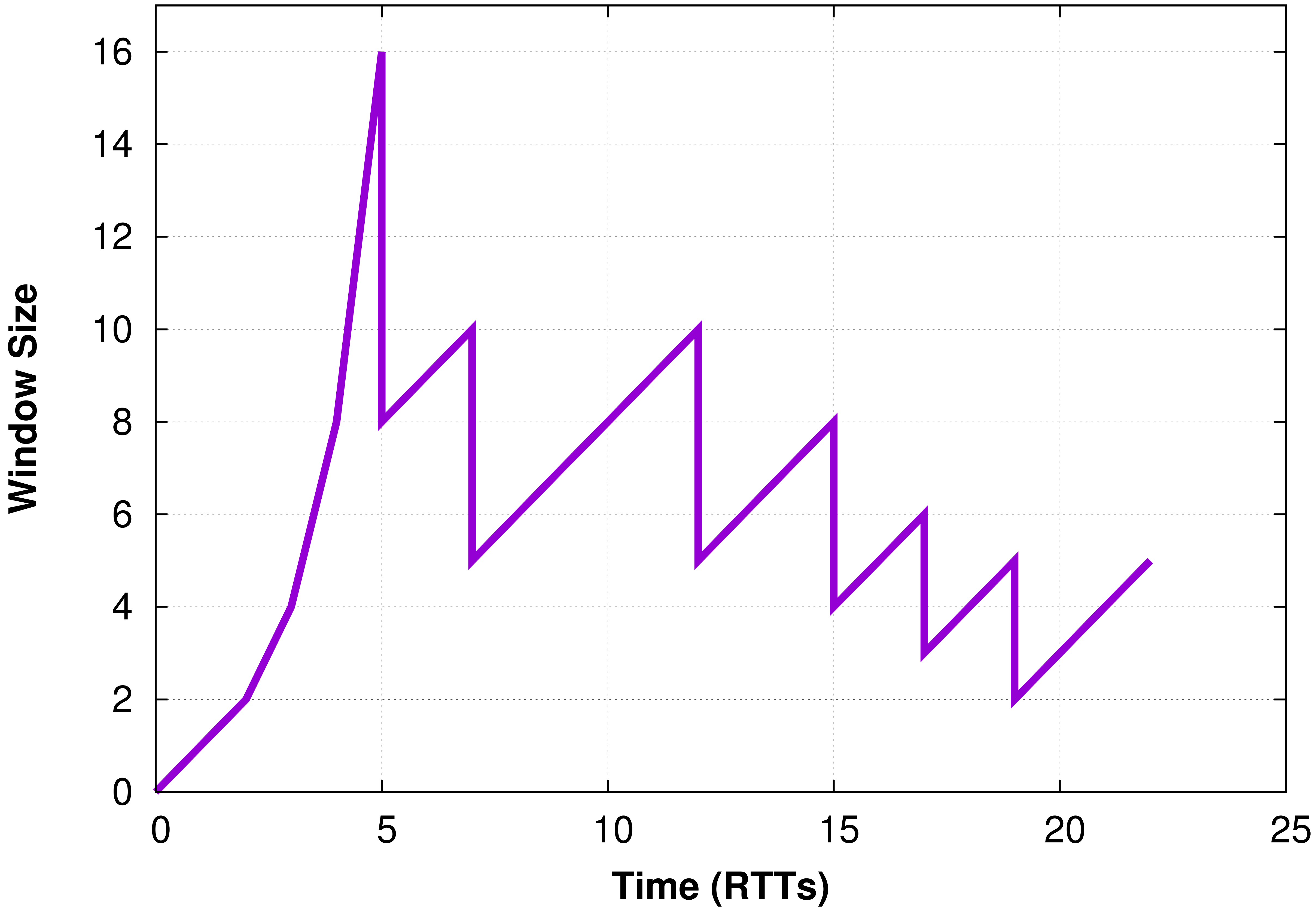
efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



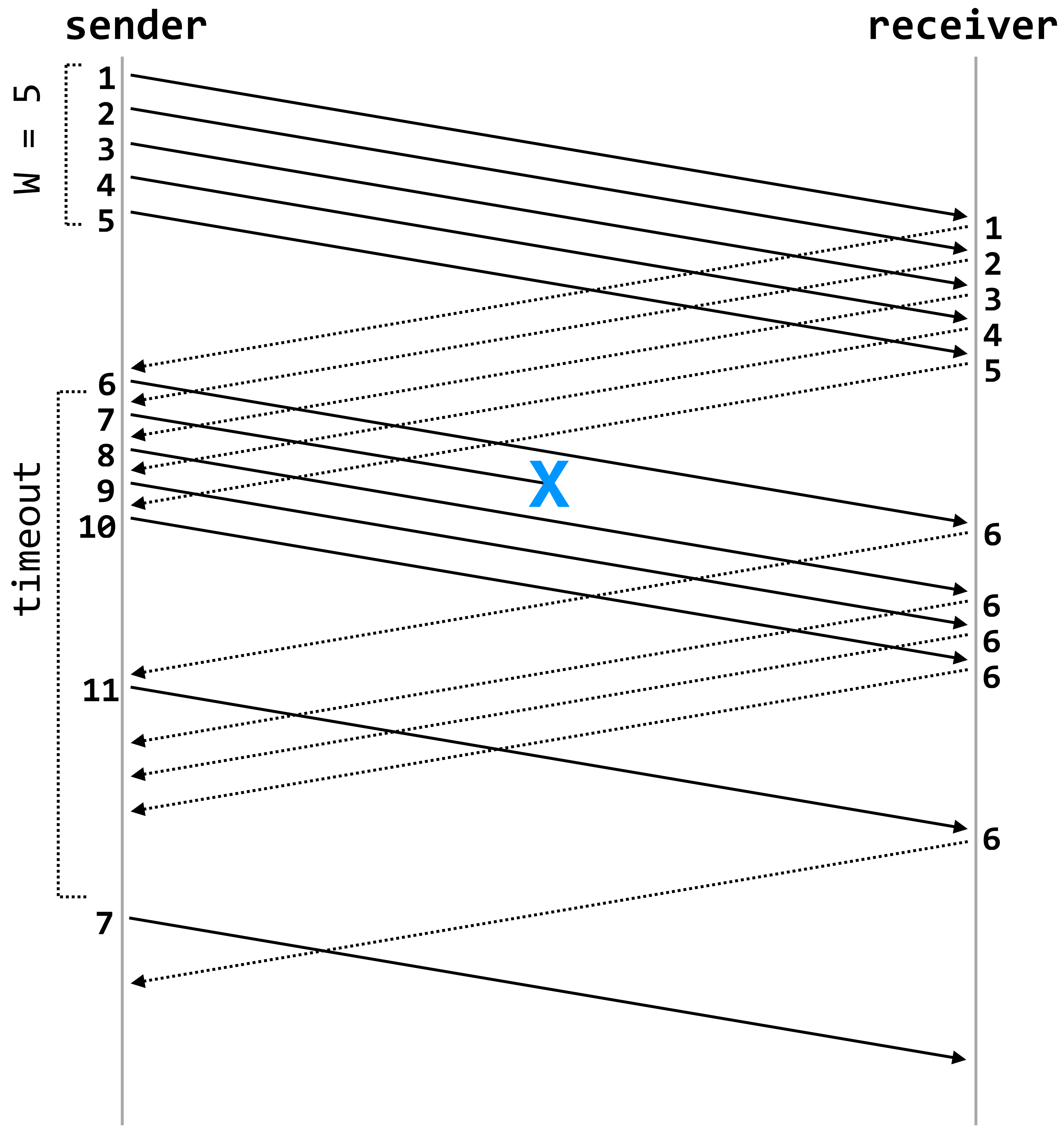
efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



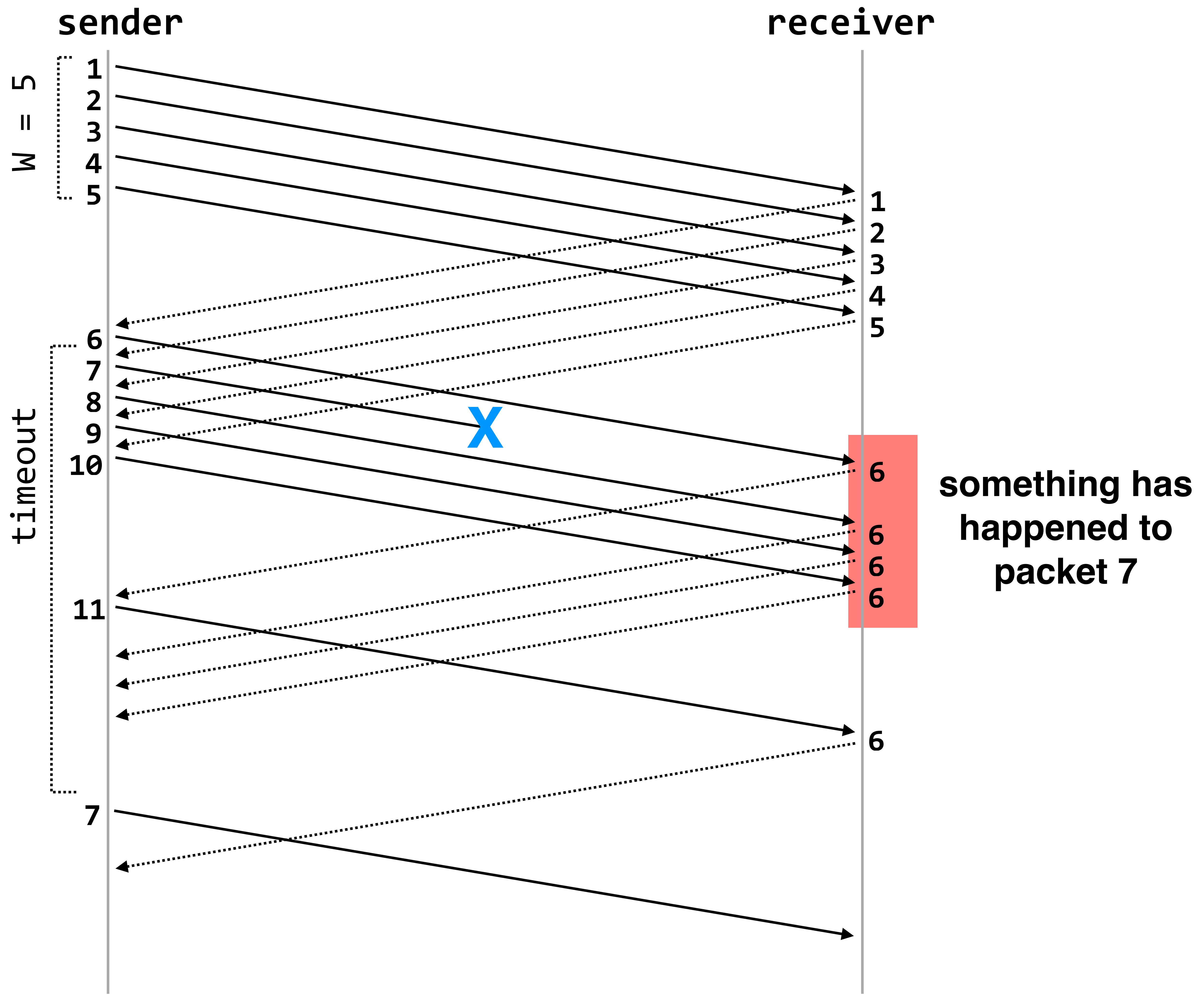
efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



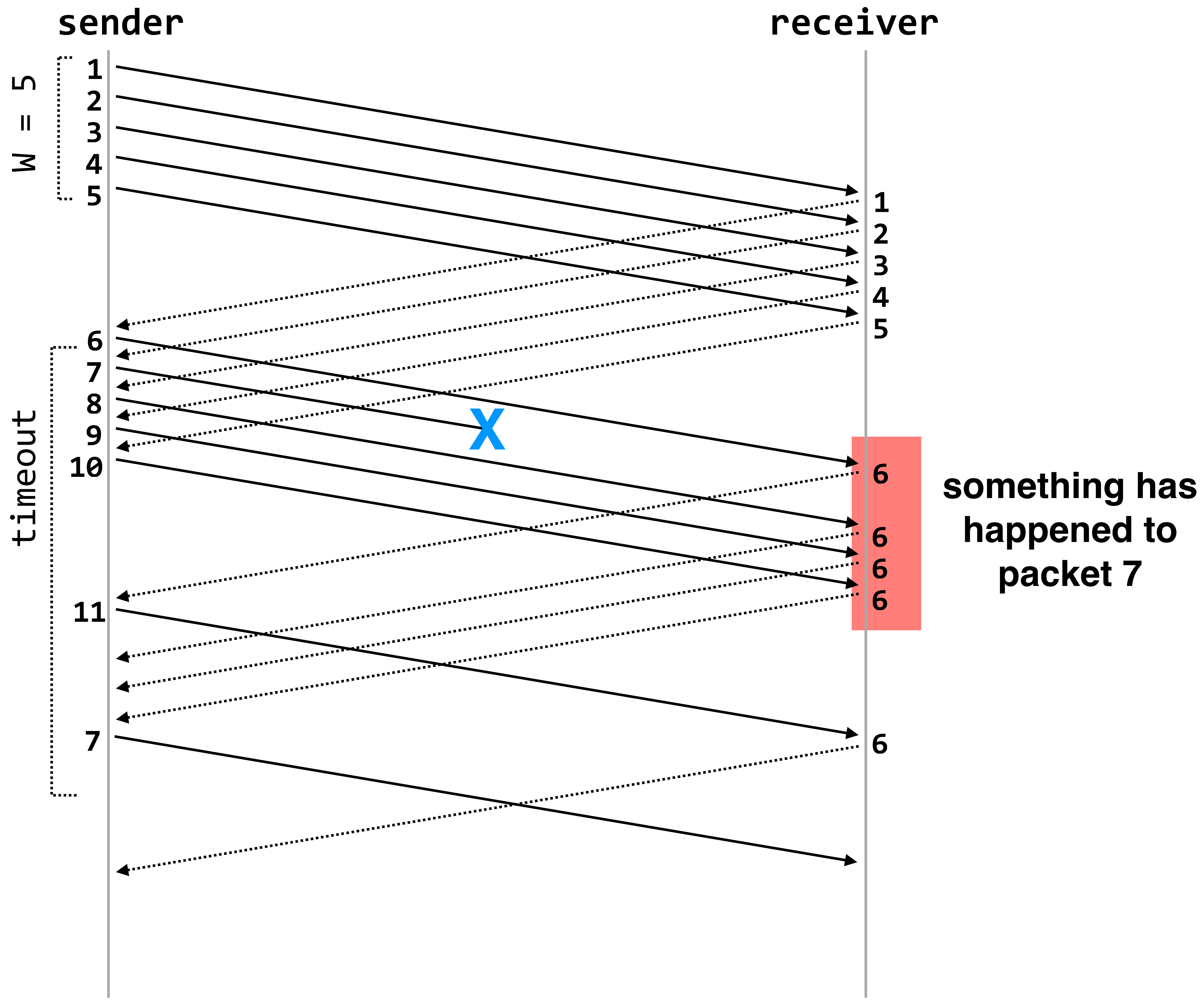
efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

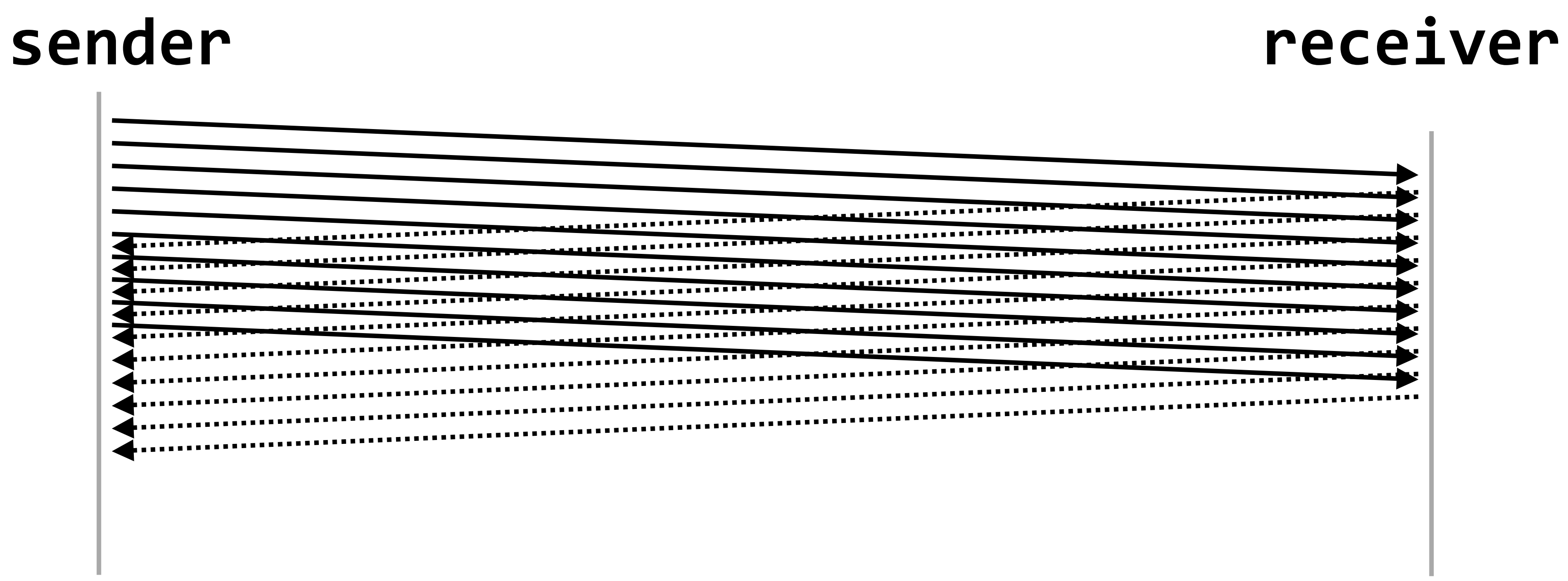
AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 "dup" ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



in practice, if a single packet is lost, the three "dup" ACKs will be received before the timeout for that packet expires

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

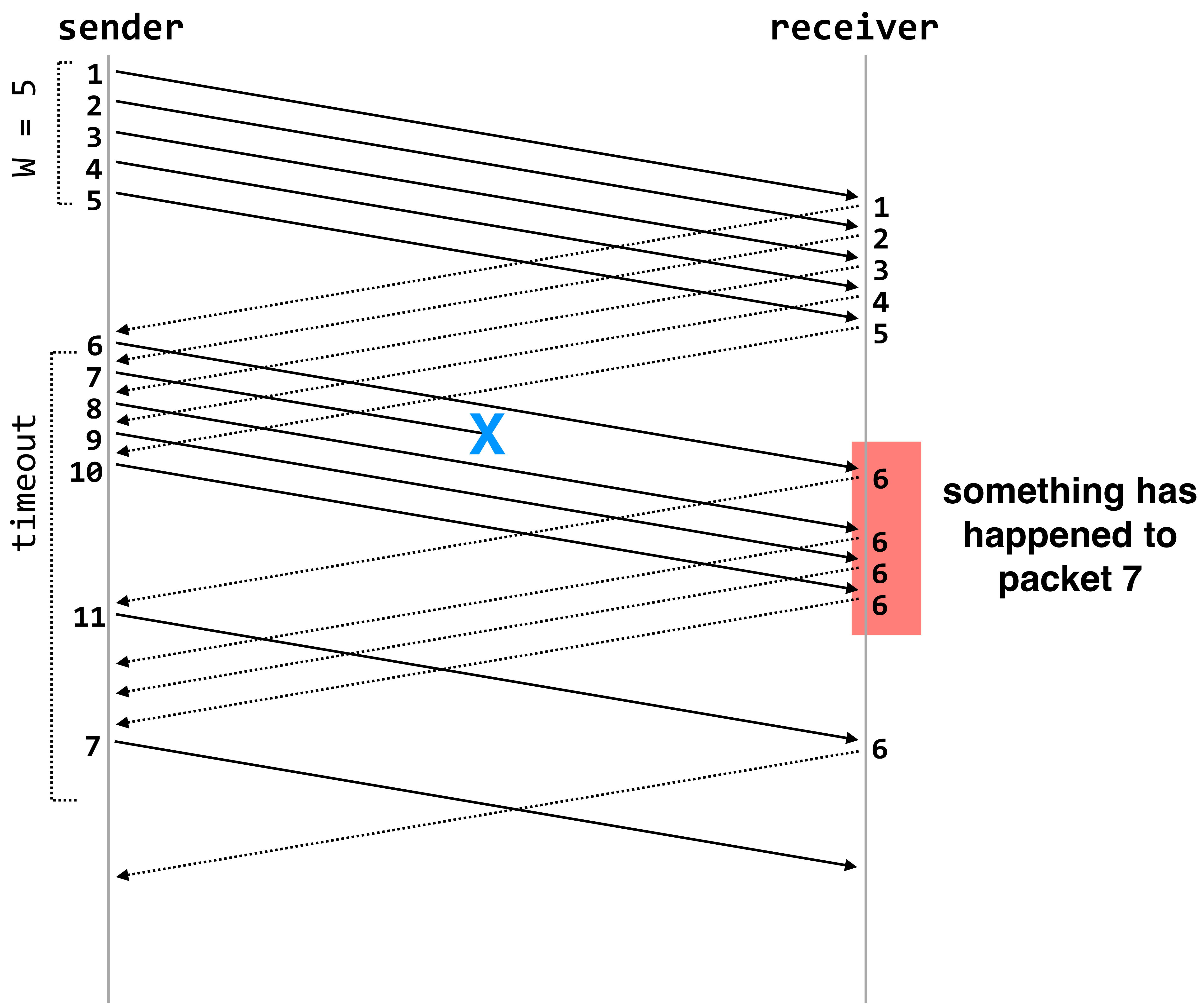
AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet **k+1** as soon as four ACKs with sequence number **k** are received

(four = original ACK + 3 "dup" ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

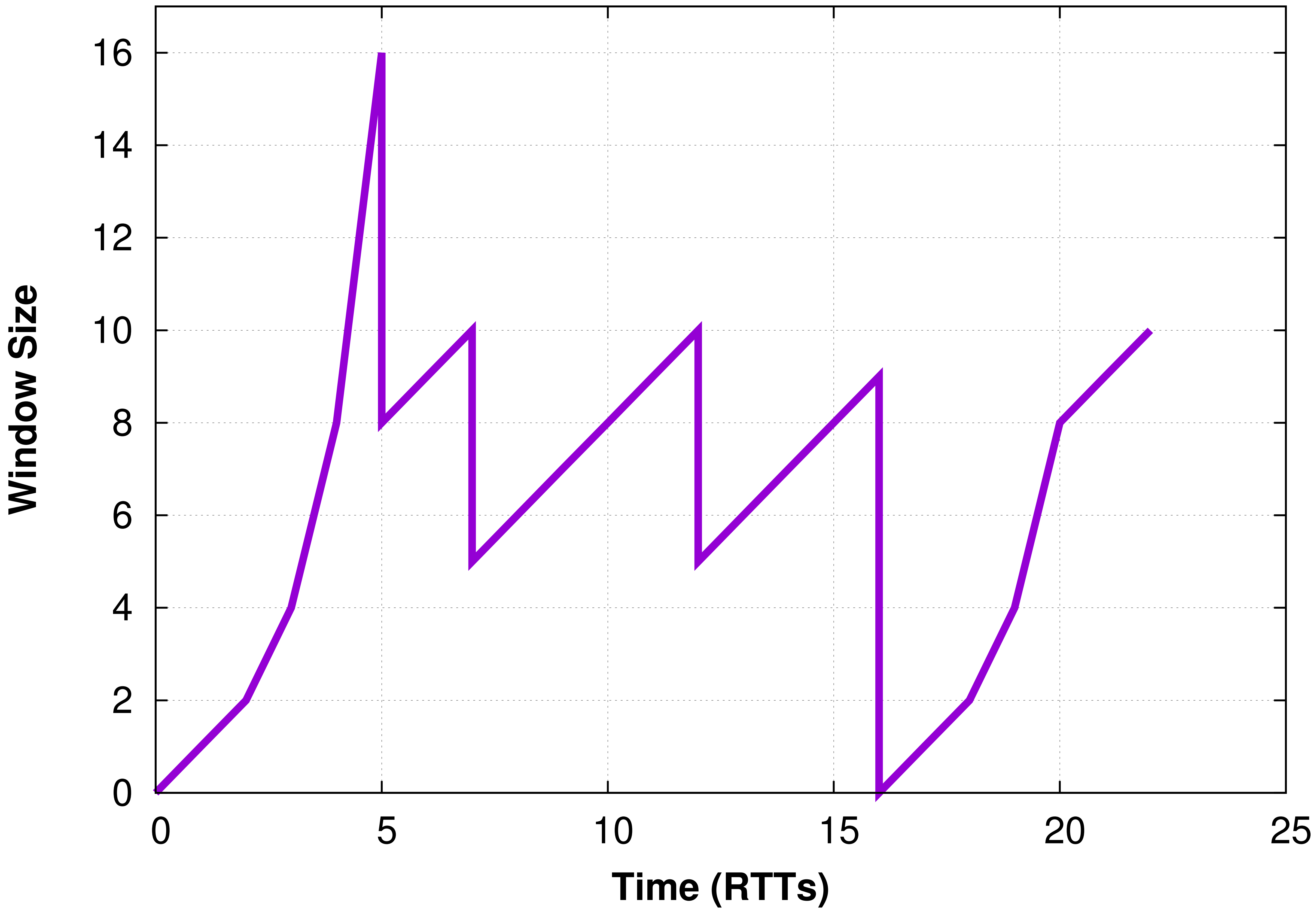
AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 "dup" ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



in practice, a retransmission due to a timeout happens when there is *significant* loss. senders are even more conservative, dropping their window back down to 1

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

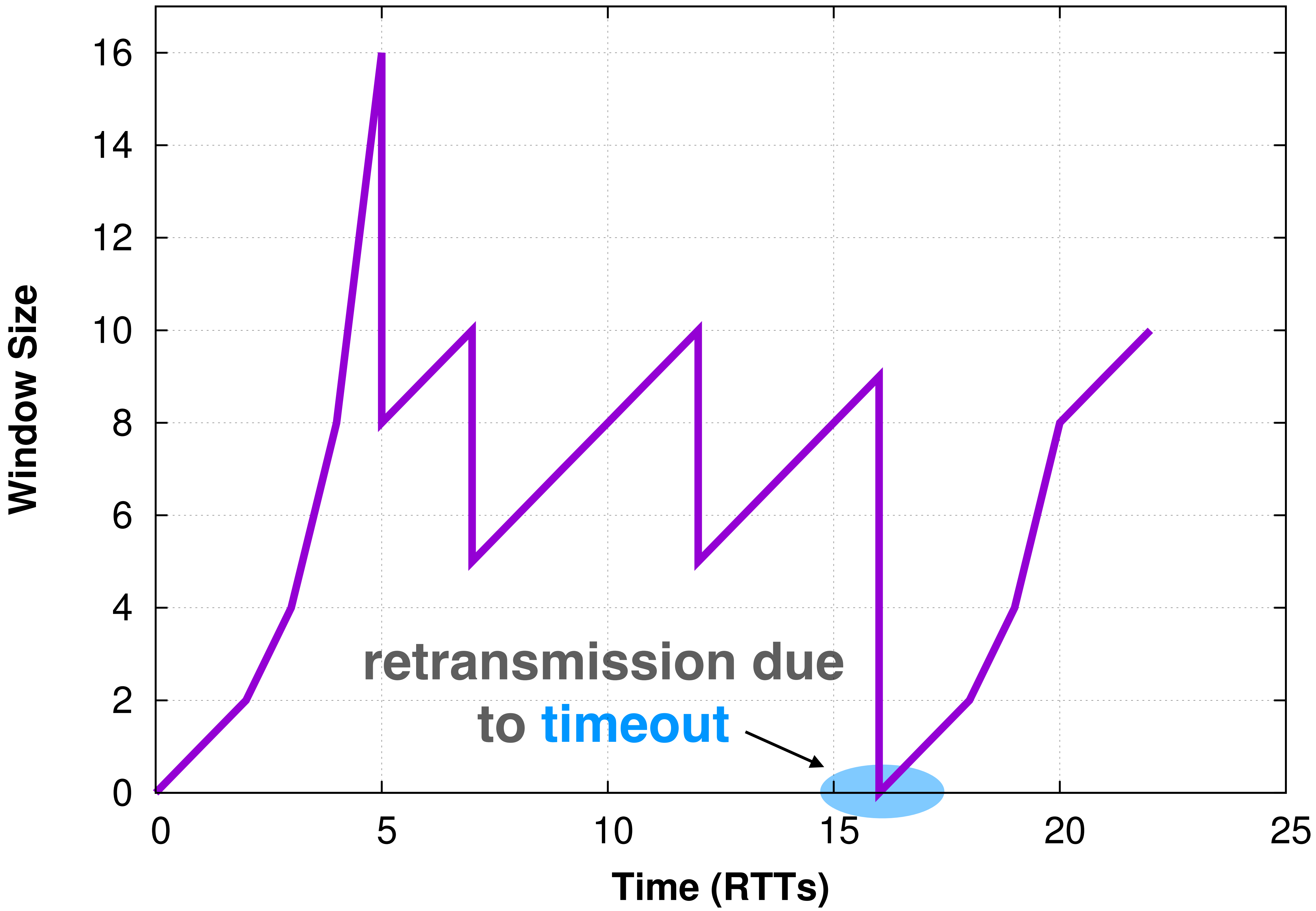
AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 "dup" ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



in practice, a retransmission due to a timeout happens when there is *significant* loss. senders are even more conservative, dropping their window back down to 1

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

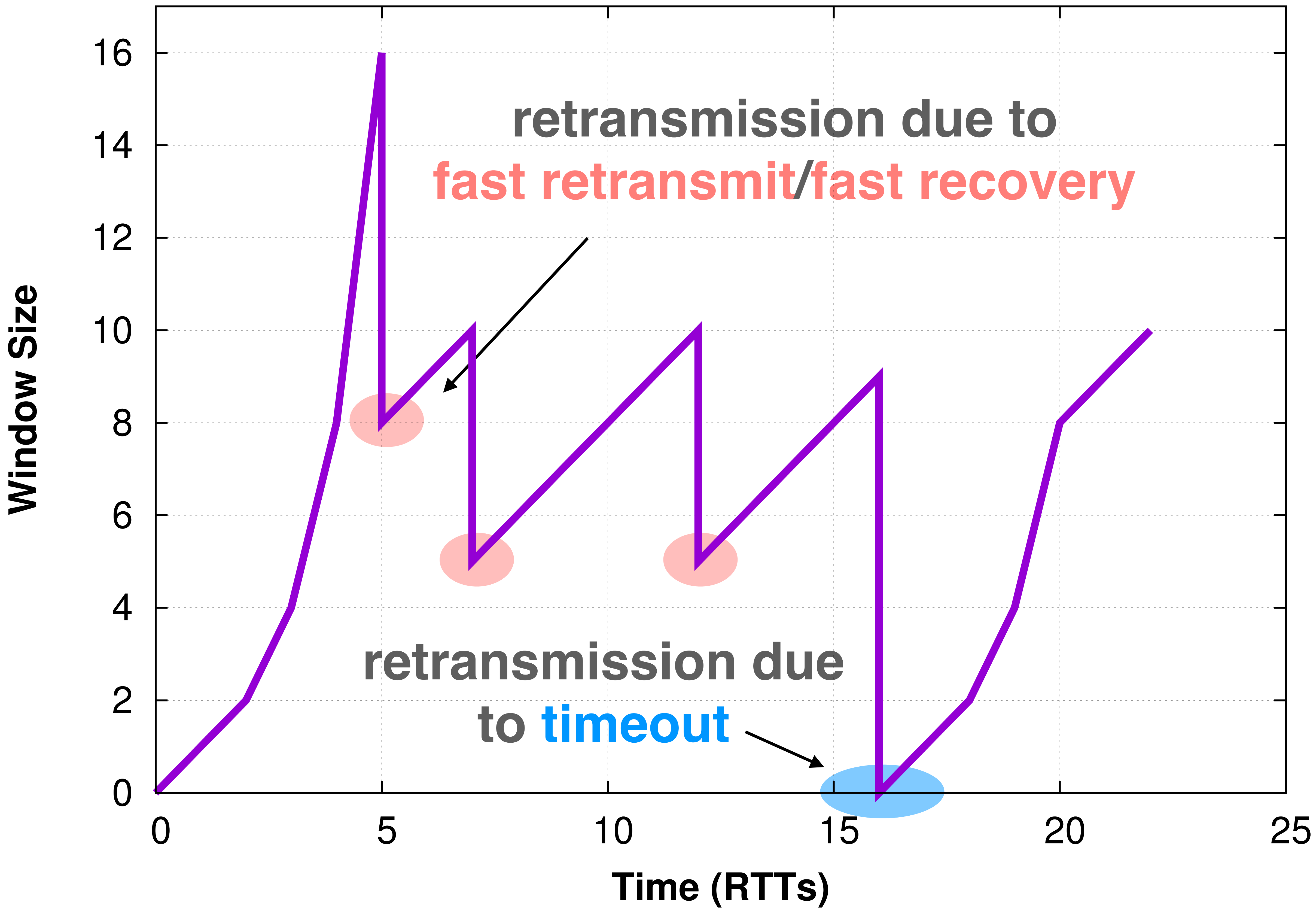
AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 “dup” ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**



in practice, a retransmission due to a timeout happens when there is *significant* loss. senders are even more conservative, dropping their window back down to 1

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 "dup" ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 "dup" ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

in certain types of networks, this style of congestion control can make these problems *worse*

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 “dup” ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

in certain types of networks, this style of congestion control can make these problems *worse*

in practice, fairness is tough to define and assess

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 “dup” ACKs)

congestion control: controlling the source rates to achieve **efficiency** and **fairness**

in certain types of networks, this style of congestion control can make these problems *worse*

in practice, fairness is tough to define and assess

AIMD is not the final word in congestion avoidance; modern versions (e.g. CUBIC TCP) use different rules to set the window size

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, `W = W + 1`; else, `W = W/2`

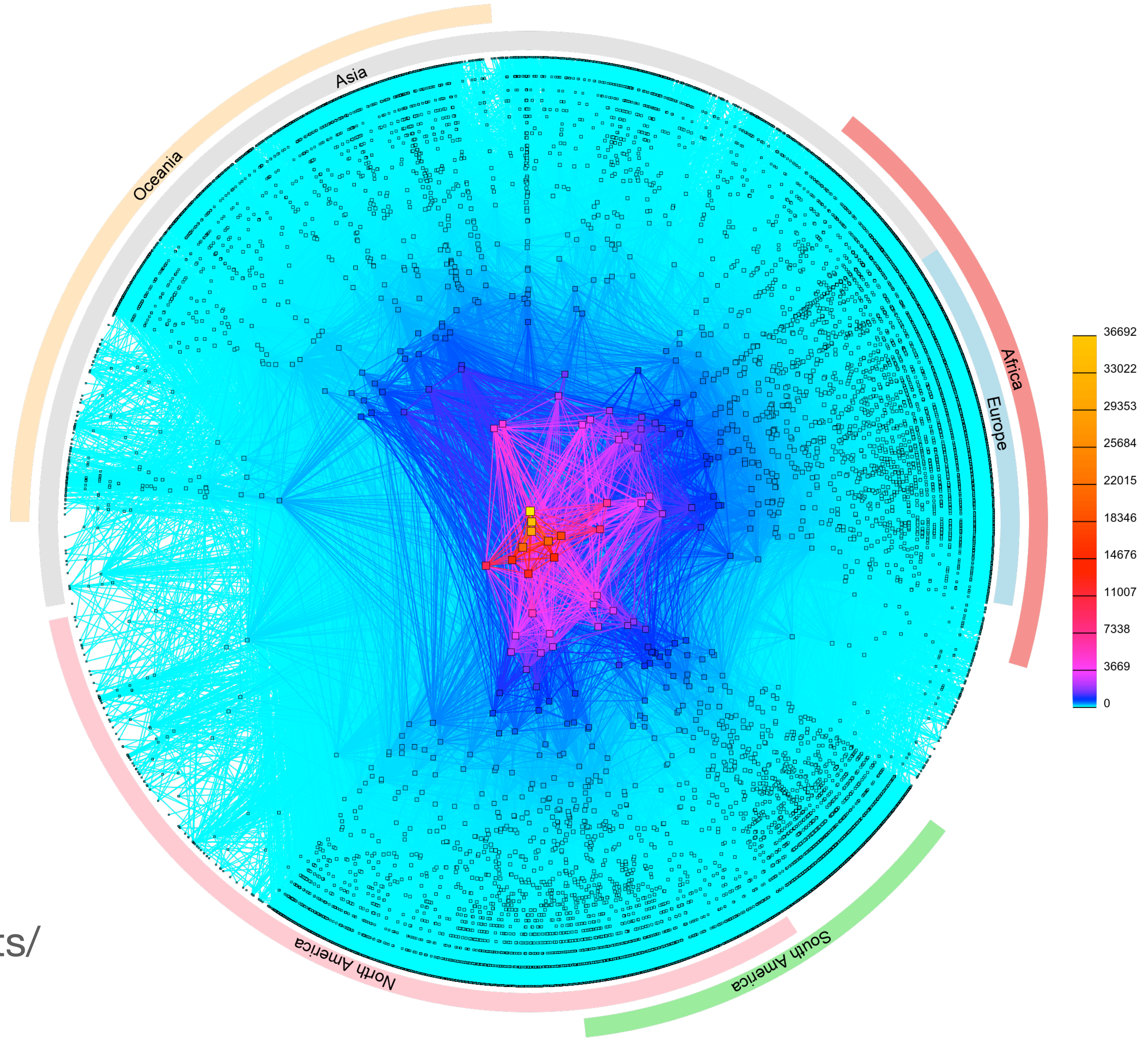
slow-start: at the start of the connection, double `W` every RTT

fast retransmit/fast recovery: retransmit packet `k+1` as soon as four ACKs with sequence number `k` are received

(four = original ACK + 3 “dup” ACKs)

1970s: ARPAnet — 1978: flexibility and layering — early 80s: growth → change — late 80s: growth → problems — 1993: commercialization

hosts.txt | distance-vector | **TCP**, UDP | OSPF, EGP, DNS | congestion collapse (which led to **congestion control**) | policy routing | CIDR



CAIDA's IPv4 AS Core, January 2020 (https://www.caida.org/projects/cartography/as-core/2020/)

application	the things that actually generate traffic
transport	sharing the network, reliability (or not) *examples: TCP, UDP*
network	naming, addressing, routing *examples: IP*
link	communication between two directly-connected nodes *examples: ethernet, bluetooth, 802.11 (wifi)*

next time: TCP congestion control doesn't react to congestion until after it's a problem; could we get senders to react before queues are full?