Slow Start TCP Reno Demystified

How congestion avoidance modes are working

10 november 2017

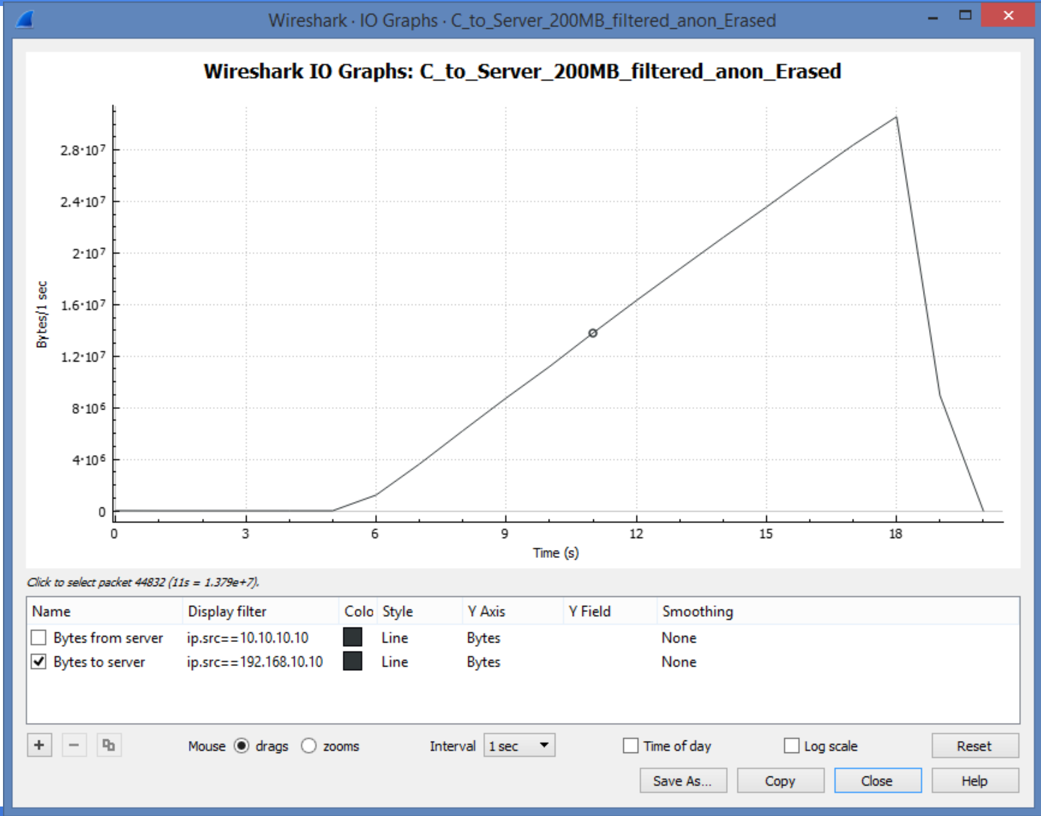
Christian Reusch

CRnetPACKETS

#SF16EU

An interesting Graph?

https://osqa-ask.wireshark.org/questions/55972/slow-writes-even-slower-reads-spanning-wan-to-netapp


Wireshark · IO Graphs · C_to_Server_200MB_filtered_anon_Erased
Wireshark IO Graphs: C_to_Server_200MB_filtered_anon_Erased
Bytes/1 sec
2.8·10⁷
2.4·10⁷
2·10⁷
1.6·10⁷
1.2·10⁷
8·10⁶
4·10⁶
0
0
3
6
9
12
15
18
Time (s)
+
−
Reset
Save As…
Copy
Close
Help

Click to select packet 44832 (11s = 1.379e+7).

Name	Display filter	Colo	Style	Y Axis	Y Field	Smoothing
☐ Bytes from server	ip.src==10.10.10.10	■	Line	Bytes		None
☑ Bytes to server	ip.src==192.168.10.10	■	Line	Bytes		None

Mouse ● drags ○ zooms

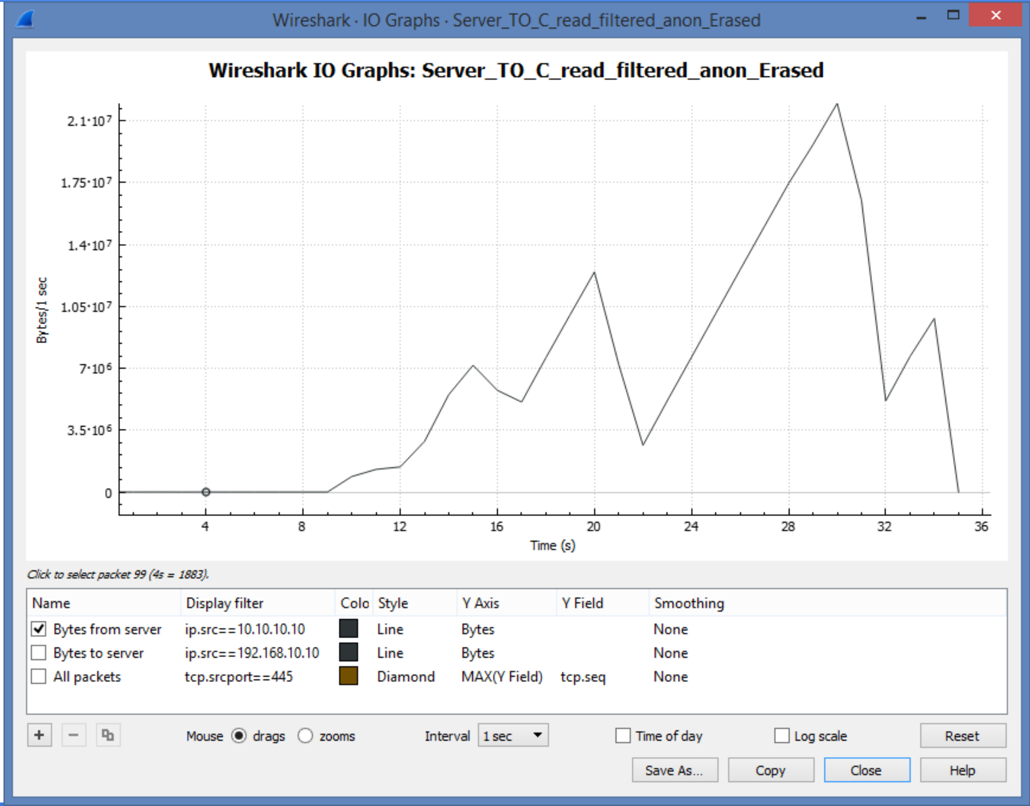
Interval 1 sec

☐ Time of day

☐ Log scale

Another interesting Graph?

https://osqa-ask.wireshark.org/questions/55972/slow-writes-even-slower-reads-spanning-wan-to-netapp



Click to select packet 99 (4s = 1883).

Name	Display filter	Colo	Style	Y Axis	Y Field	Smoothing
☑ Bytes from server	ip.src==10.10.10.10	■	Line	Bytes		None
☐ Bytes to server	ip.src==192.168.10.10	■	Line	Bytes		None
☐ All packets	tcp.srcport==445	■	Diamond	MAX(Y Field)	tcp.seq	None

Mouse ● drags ○ zooms

Interval 1 sec

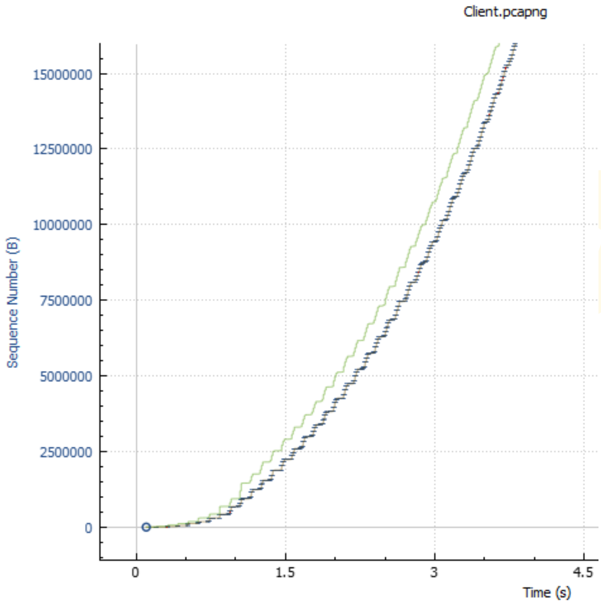
☐ Time of day

☐ Log scale

Have you ever heard this sentence?

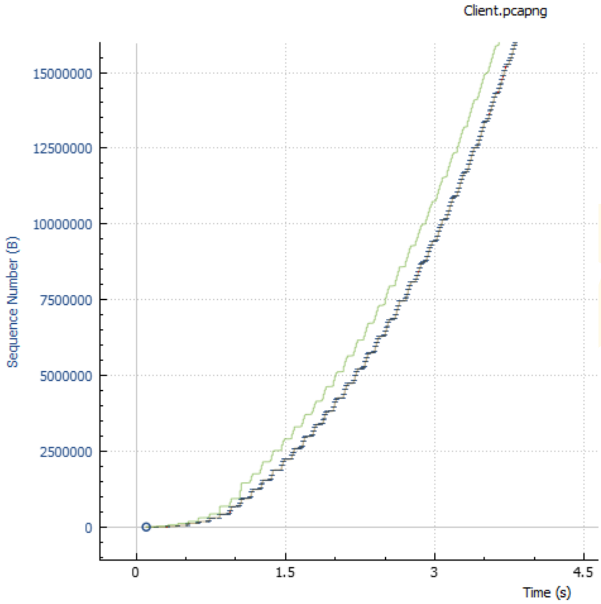
- **<u>This is due to Slow Start!</u>**

- But how does Slow Start work?
- Why is it called Slow Start?
- And why do we need Slow Start?



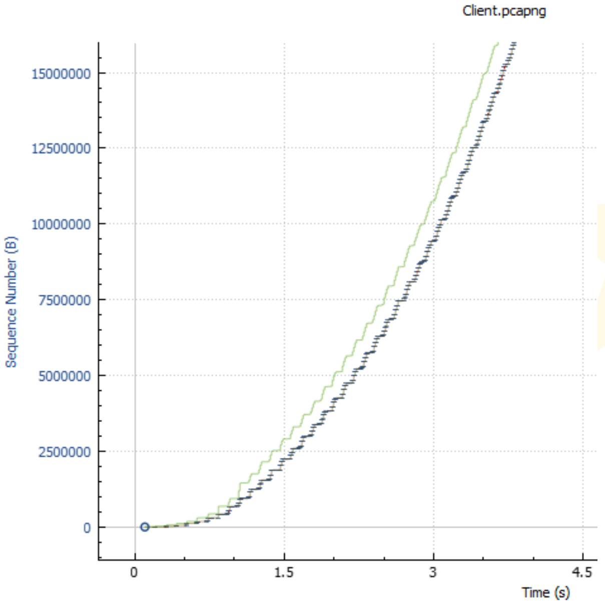
Slow Start RFC History

- Slow Start is defined in the following RFCs
  - **RFC5681** of the year 2009 obsoletes:
    - RFC2581 of the year 1999 which obsoletes:
      - RFC2001 of the year 1997 author Richard W. Stevens



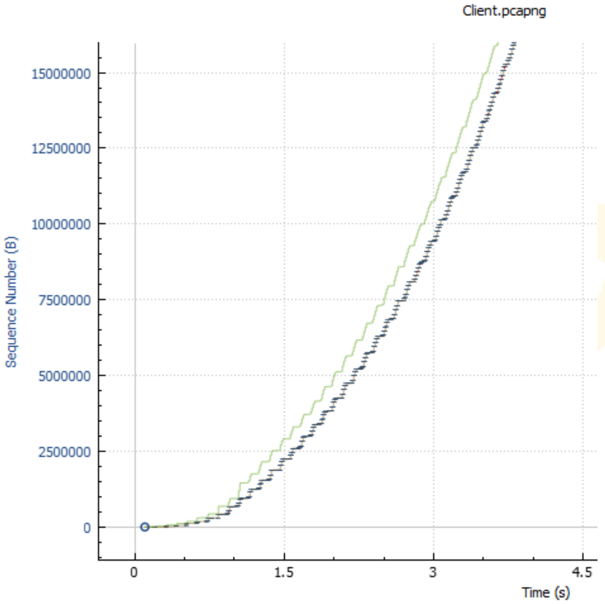
Slow Start Variables

- We need Slow Start to determine the available bandwidth
- We do this by using the following variables:
  - Receive Window Size **rwnd**
  - Congestion Window: **cwnd**
  - Slow Start Threshhold: **ssthresh**
  - Initial Congestion Window Size **iw**
  - Sender Maximum Segment Size **smss**
  - Receivers Maximum Segment Size **rmss**



rwnd = Receivers Advertised Window

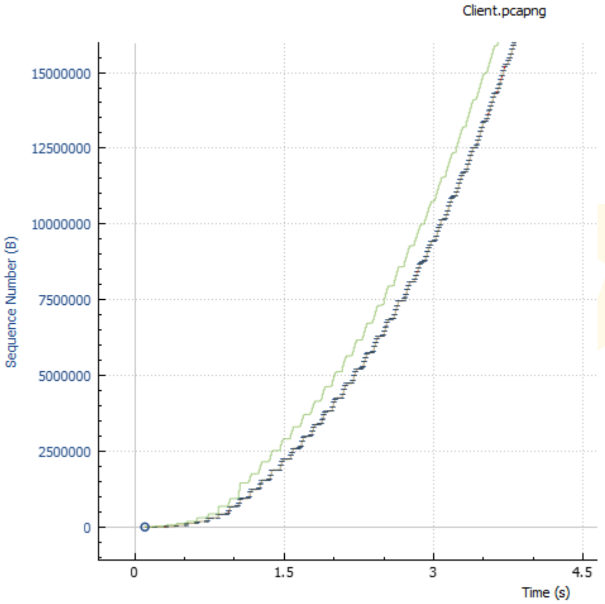
- **rwnd:**
  - The receiver's advertised window (**rwnd**) is a receiver-side limit on the amount of outstanding data.
  - The **rwnd** size can be seen in a trace.
    -> It is represented by the TCP Window Size within the TCP- header



cwnd = Congestion Window

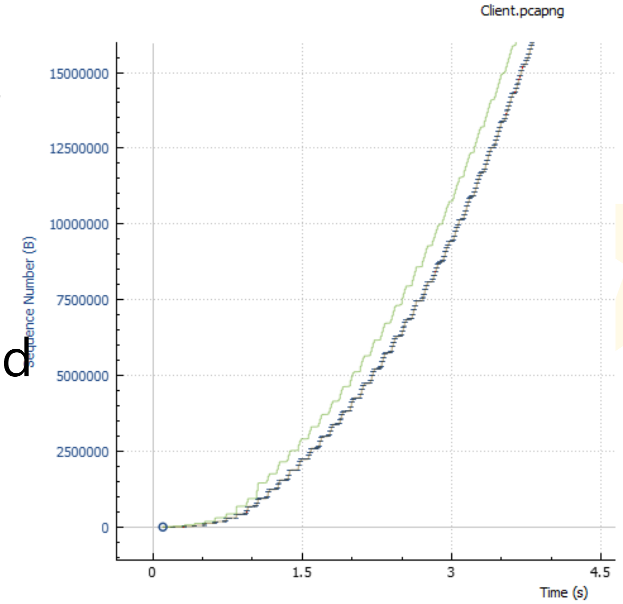
- **cwnd:**
  - The congestion window (**cwnd**) is a sender-side limit on the amount of data the sender can transmit into the network before receiving an acknowledgment (ACK)
  - The **cwnd** size is a stack variable which **CAN NOT** be **seen** in a trace, it just can be guessed
  - Can be made visible by the command “`ss -i`” on linux systems

- The **minimum of cwnd and rwnd** governs data transmission.



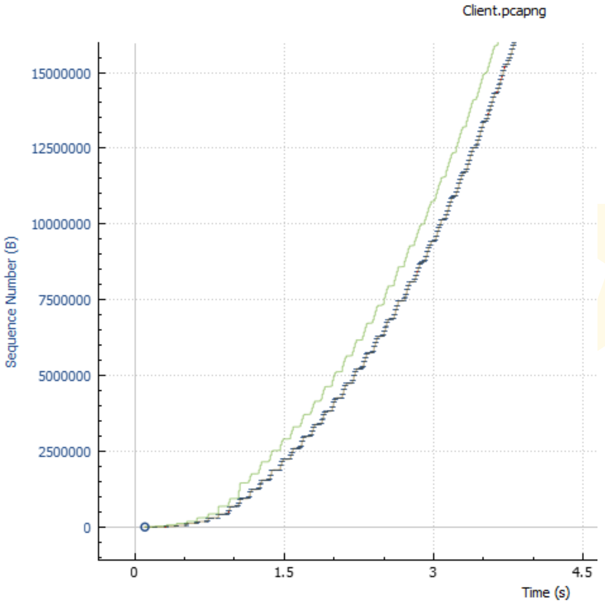
ssthresh = Slow Start Threshold

- **ssthresh**
  The slow start threshold (**ssthresh**), says if slow start algorithm is used or not.
- The **initial** value of **ssthresh** SHOULD be set **HIGH ENOUGH**
  (e.g., to the size of the largest possible advertised window)



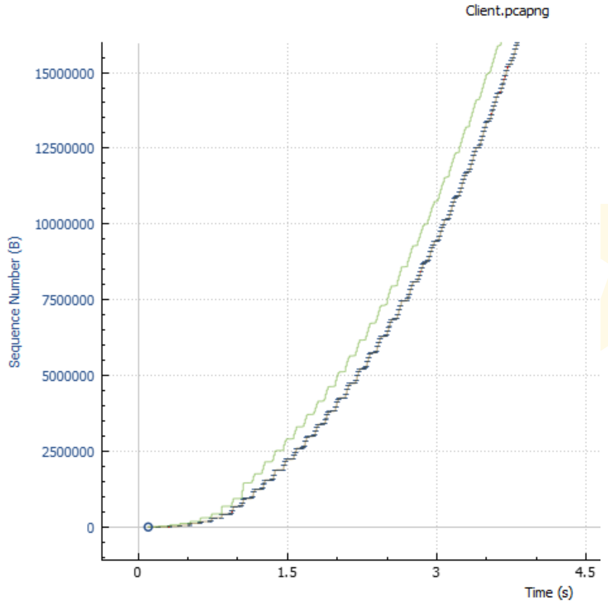
IW = Initial Congestion Window

- **IW** MUST be set using the following guidelines as an upper bound.
  - If **SMSS <= 1095 bytes**:
    IW = **4 * SMSS** bytes and **MUST NOT** be more than **4 segments**
  - If (**SMSS > 1095 bytes**) and (**SMSS <= 2190 bytes**):
    IW = 3 * SMSS bytes and **MUST NOT** be more than **3 segments**
  - If **SMSS > 2190 bytes**:
    IW = 2 * SMSS bytes and **MUST NOT** be **more** than **2 segments**



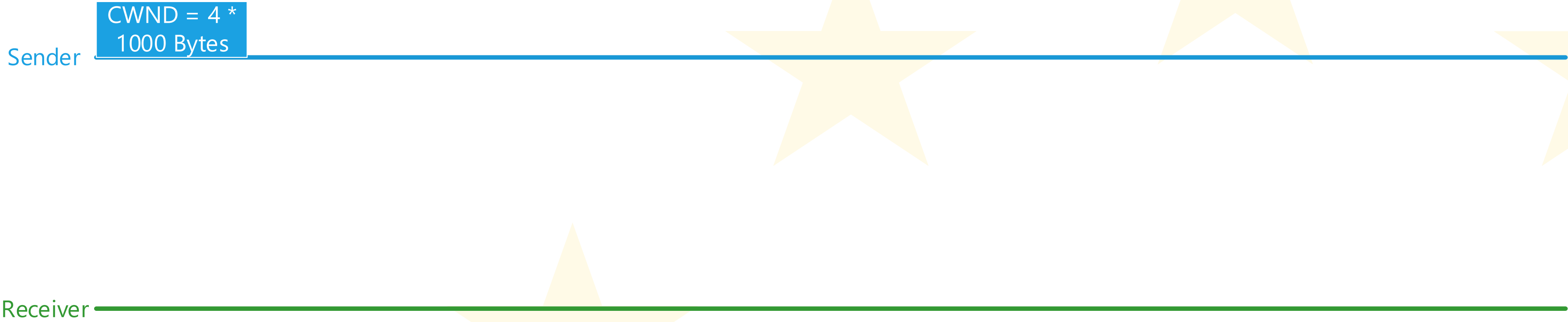
When do we use Slow Start

- If **cwnd < sstresh** then **Slow Start** is used
- If **cwnd > sstresh** then **congestion avoidance** is used



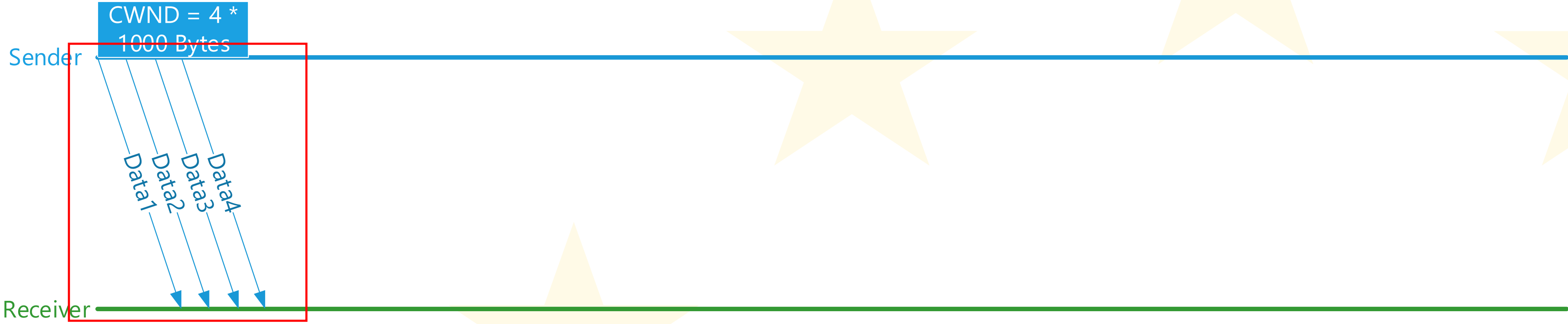
Ideal Slow Start 1 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



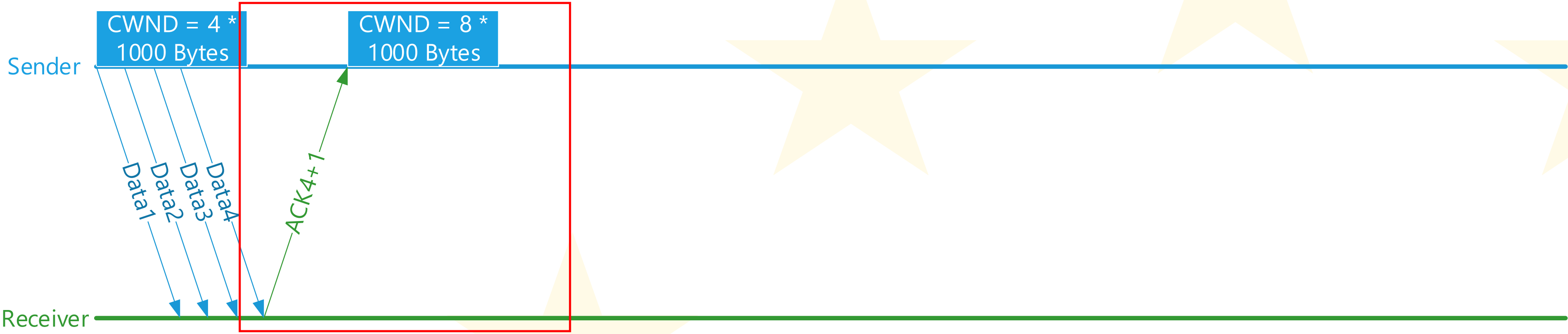
Ideal Slow Start 2 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



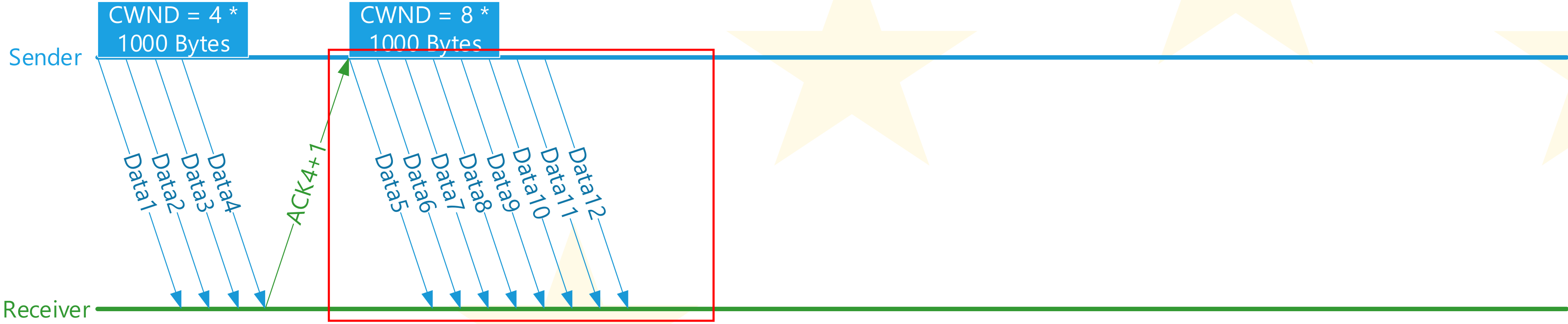
Ideal Slow Start 3 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



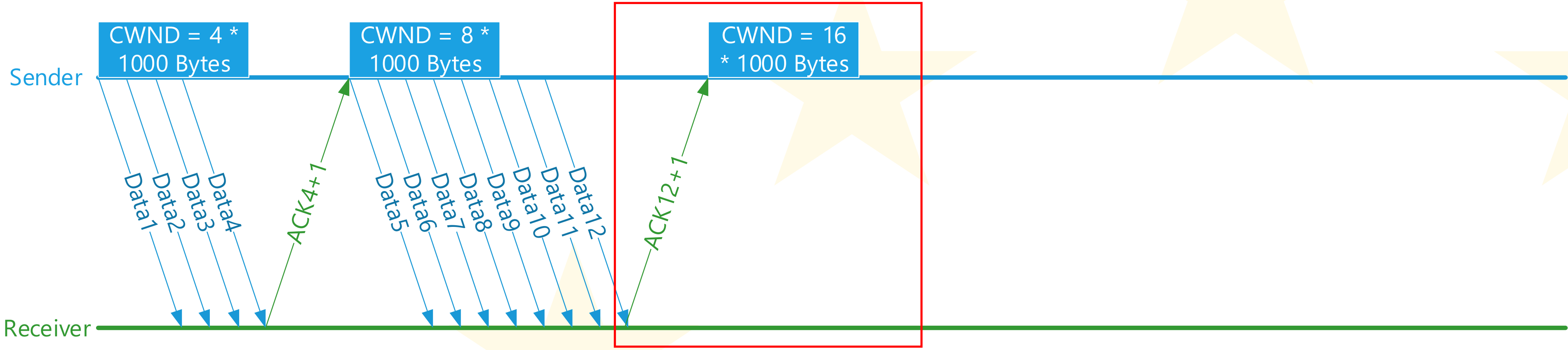
Ideal Slow Start 4 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



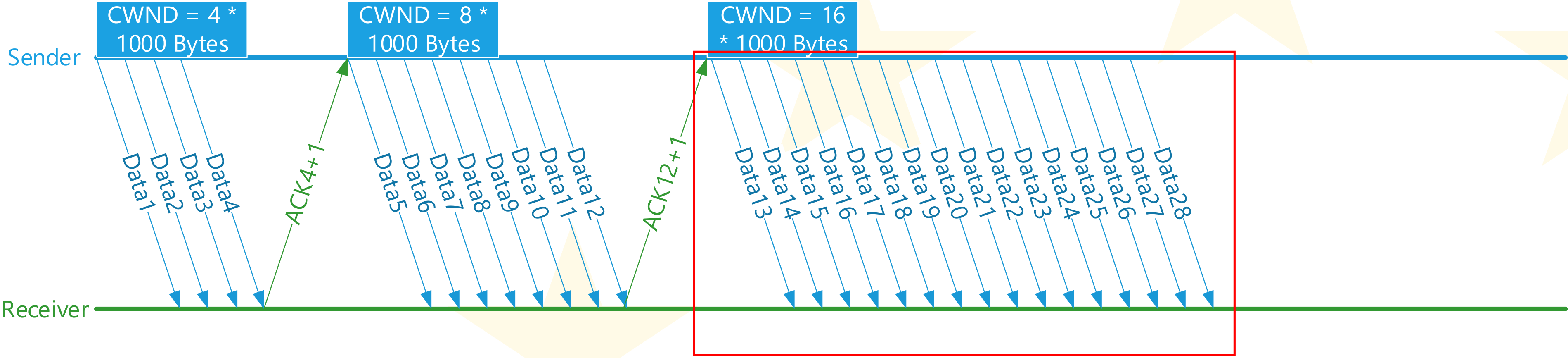
Ideal Slow Start 5 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



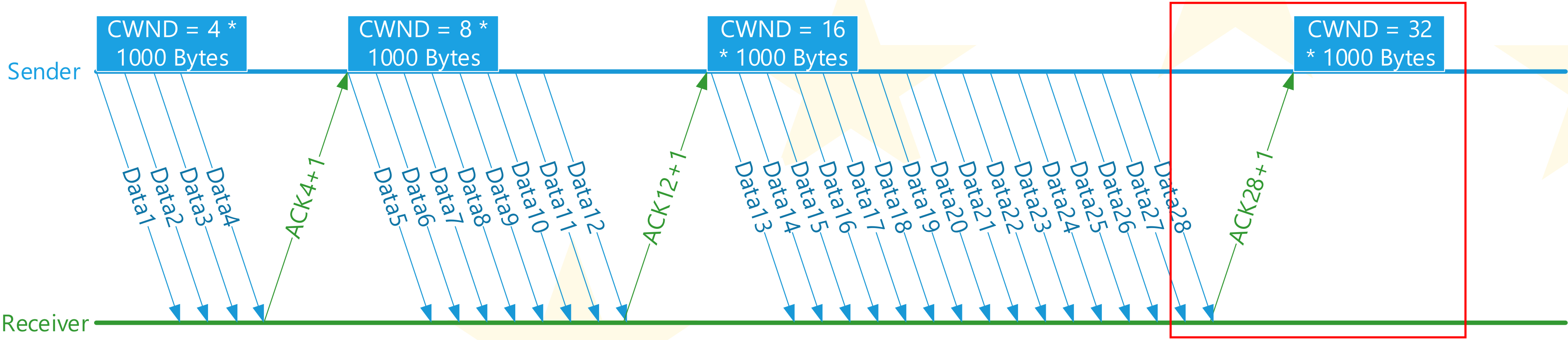
Ideal Slow Start 6 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



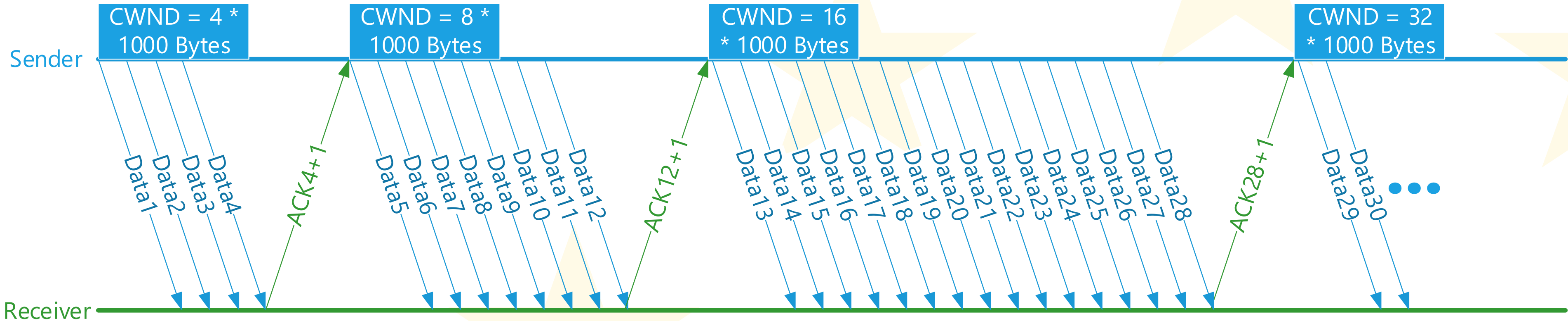
Ideal Slow Start 7 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



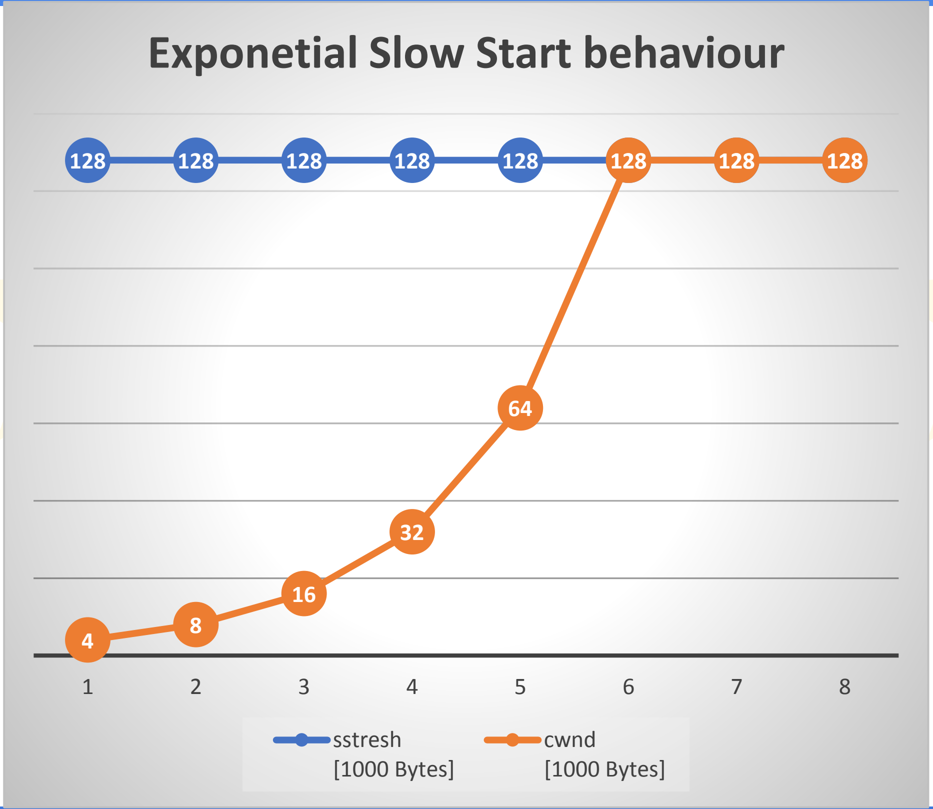
Ideal Slow Start 8 / 9

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



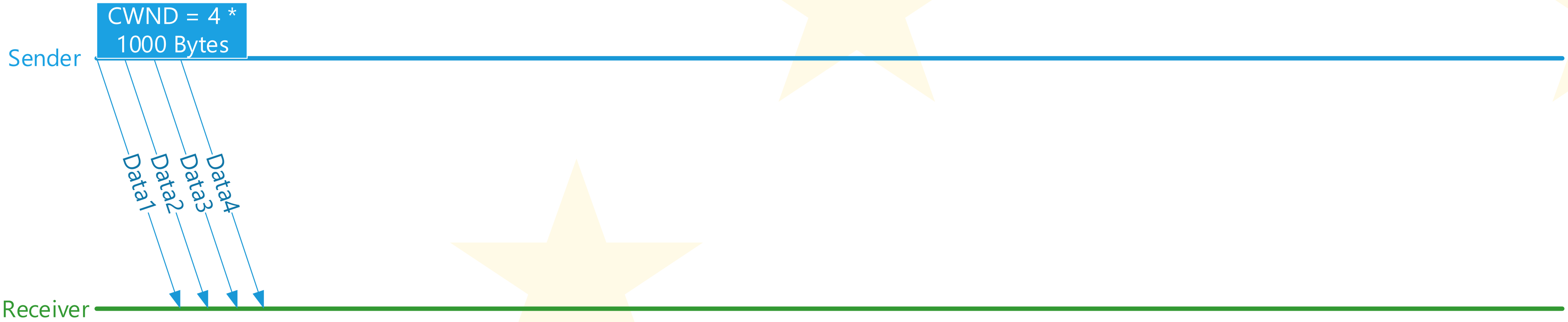
Slow Start cgwnd Growth

Time [cycles]	sstresh [1000 Bytes]	cwnd [1000 Bytes]	SMSS [1000 Bytes]	rwnd [1000 bytes]
1	128	4	1	256
2	128	8	1	256
3	128	16	1	256
4	128	32	1	256
5	128	64	1	256
6	128	128	1	256
7	128	128	1	256
8	128	128	1	256
9	128	128	1	256



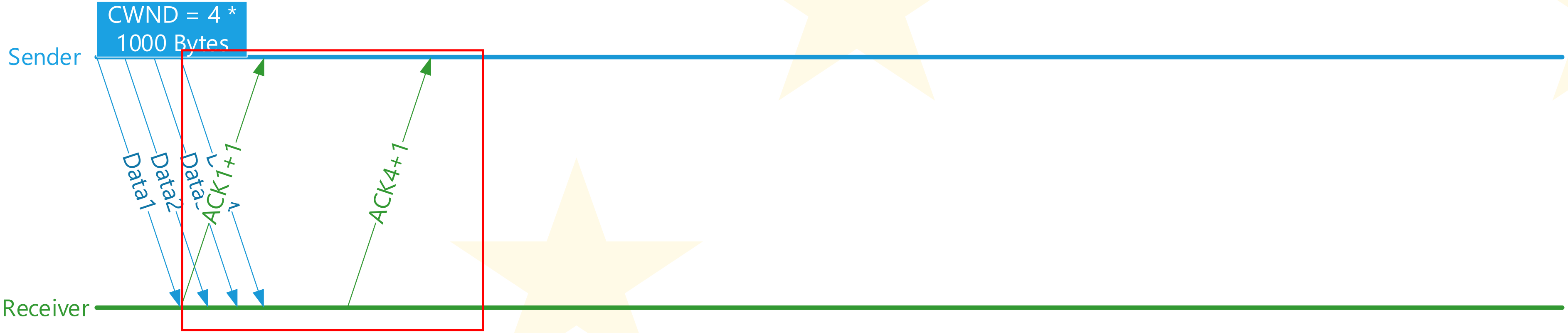
Slow Start Variation 1 / 7

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



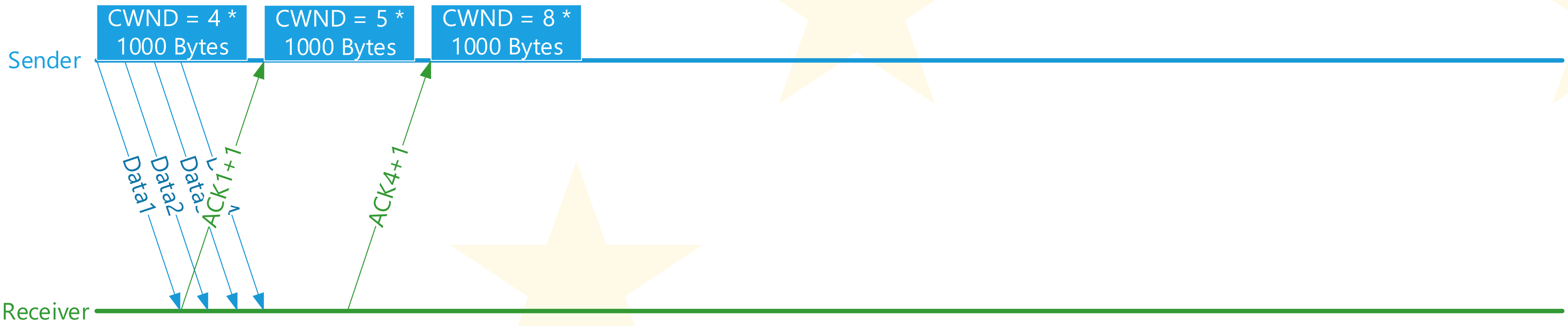
Slow Start Variation 2 / 7

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



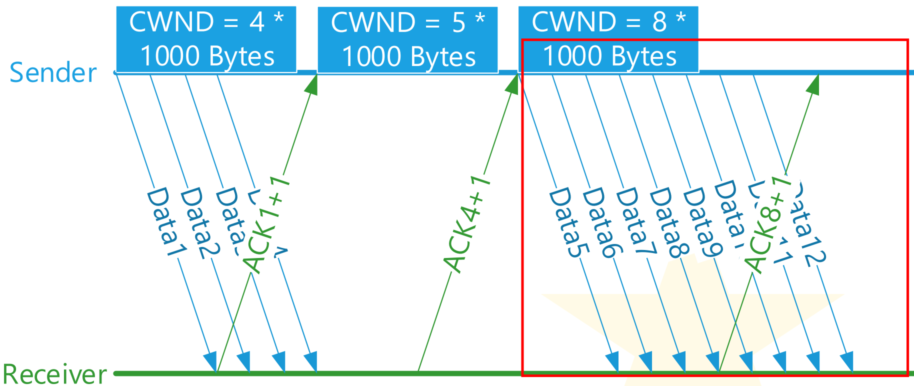
Slow Start Variation 3 / 7

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



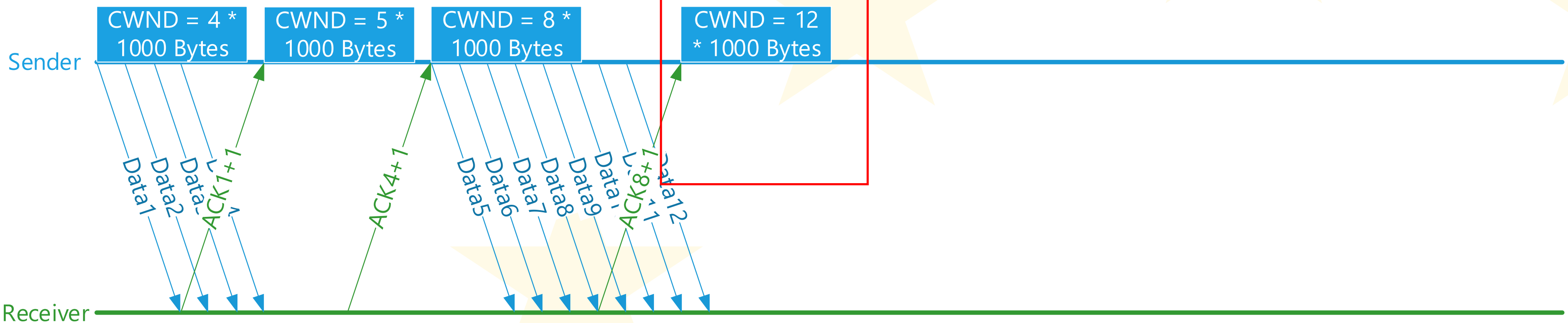
Slow Start Variation 4 / 7

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



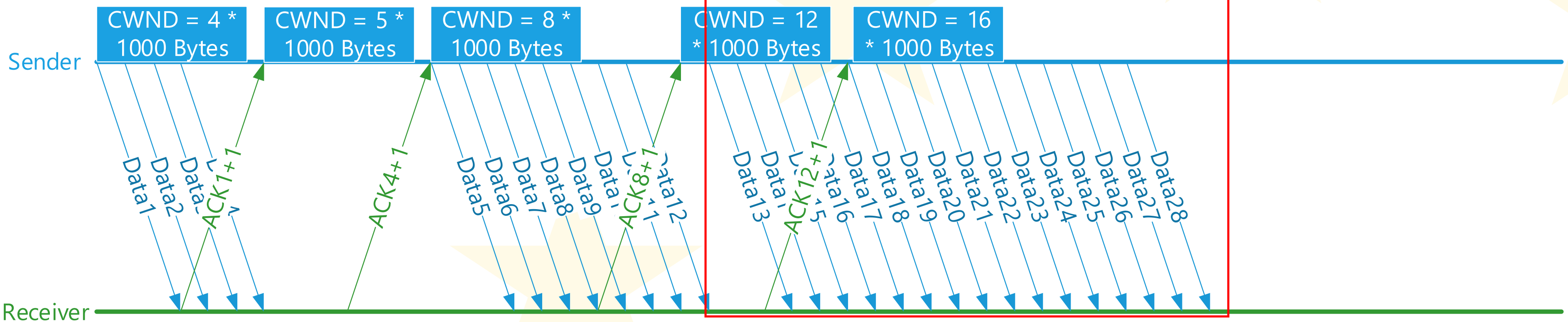
Slow Start Variation 5 / 7

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



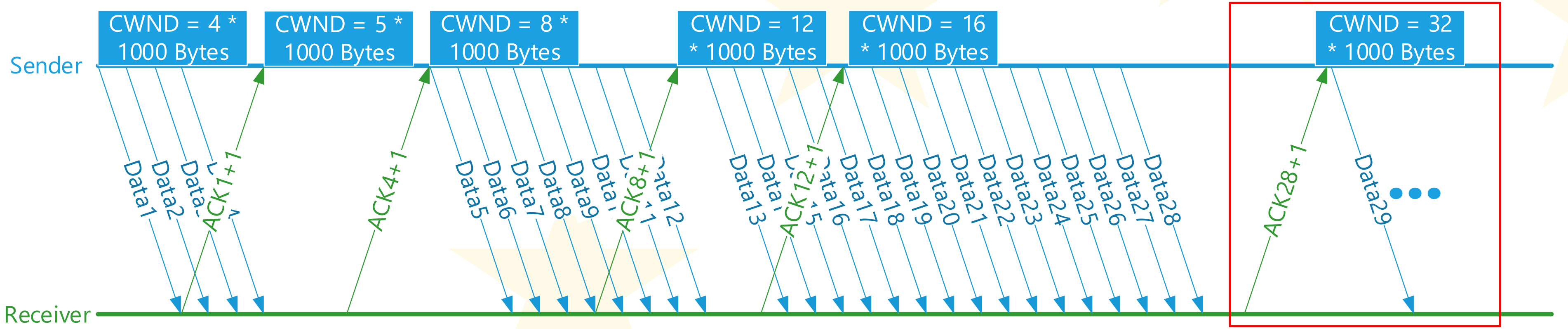
Slow Start Variation 6 / 7

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



Slow Start Variation 7 / 7

SMSS = 1 * 1000 Bytes
IW = 4 * 1000 Bytes
SSTRESH = 64 * 1000 Bytes



Slow Start Formulas

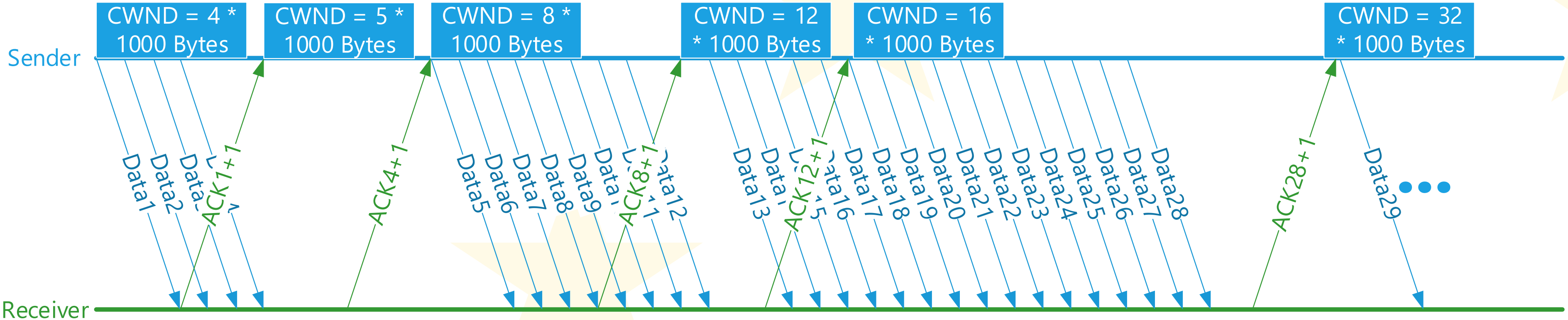
Growthrate of the cwnd:

- per window basis

cwnd = cwnd * 2

- per ACK basis

cwnd = cwnd +1



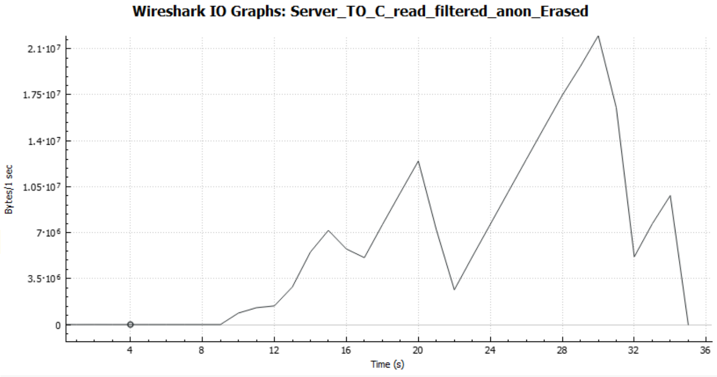
Slow Start

DEMO

SlowStart.pcapng

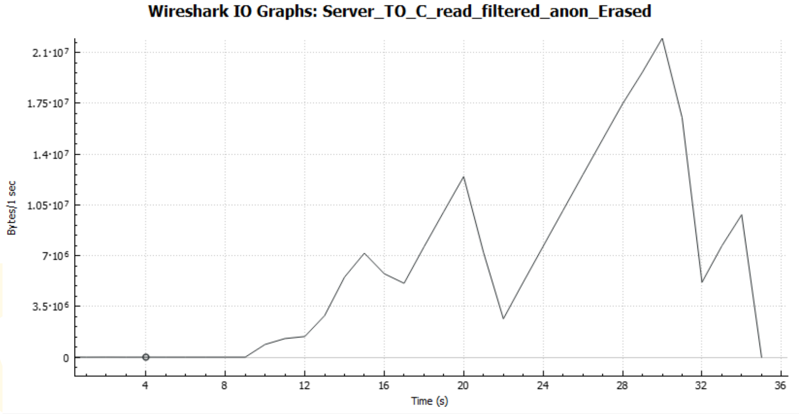
TCP Reno History

- Evolution of TCP Reno
  - TCP Tahoe (1988)
  - TCP Reno (1990) RFC2001
  - TCP New Reno (1999)
    - Frirstly described in RFC 2582
    - Now **RFC 6582**
- TCP Tahoe and Reno where originally the codenames of their initial Berkeley Unix Distributions



TCP Reno

- TCP Tahoe
  - Slow Start
  - Fast Retransmission
  - Additive Increase Multiplicative Decrease (AIMD)
- TCP Reno
  - Fast Recovery
- TCP New Reno
  - Partial ACK

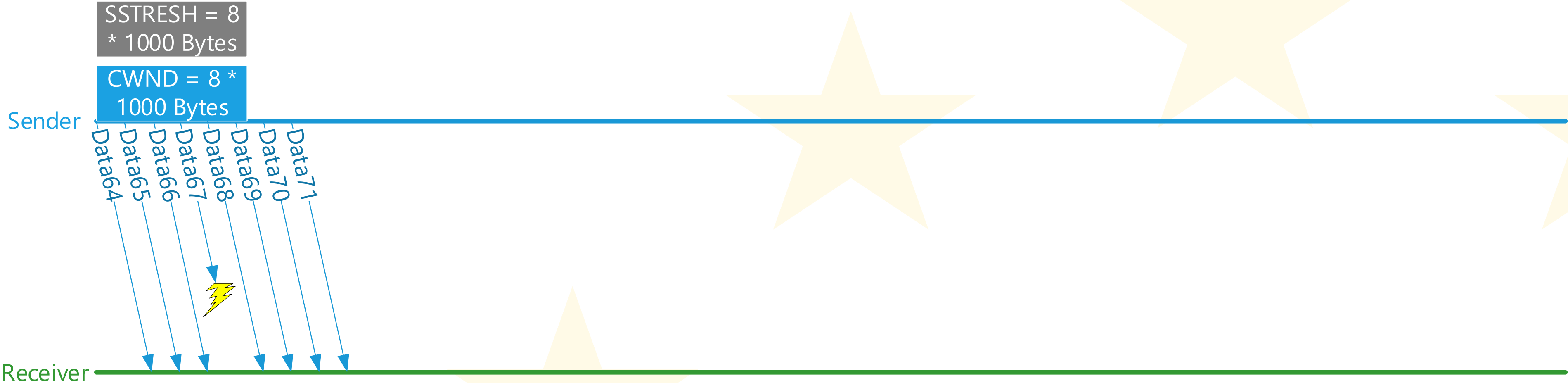


TCP Reno

- Selective Acknnowledgements (**SACK**)
  - Is no direct Part of TCP Reno History
  - It is a „side improvement" which is defined in **RFC7323** (RFC1323 <- old one)

TCP Tahoe: Fast Retransmit 1 / 5

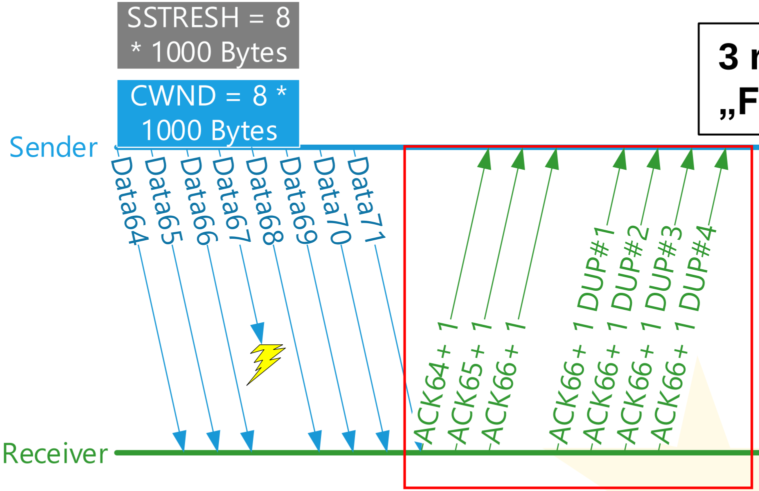
SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes



TCP Tahoe: Fast Retransmit 2 / 5

SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes

3 received Duplicate ACKs will cause a „Fast Retransmission“!

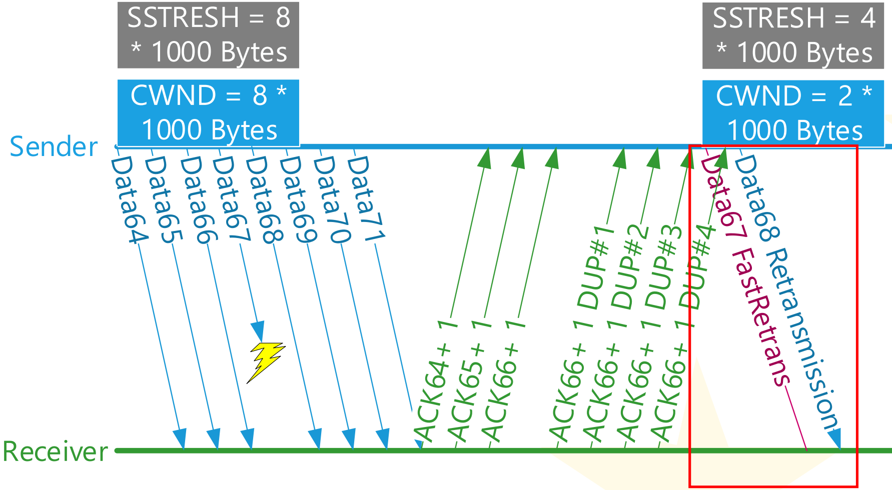


TCP Tahoe: Fast Retransmit 3 / 5

SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes

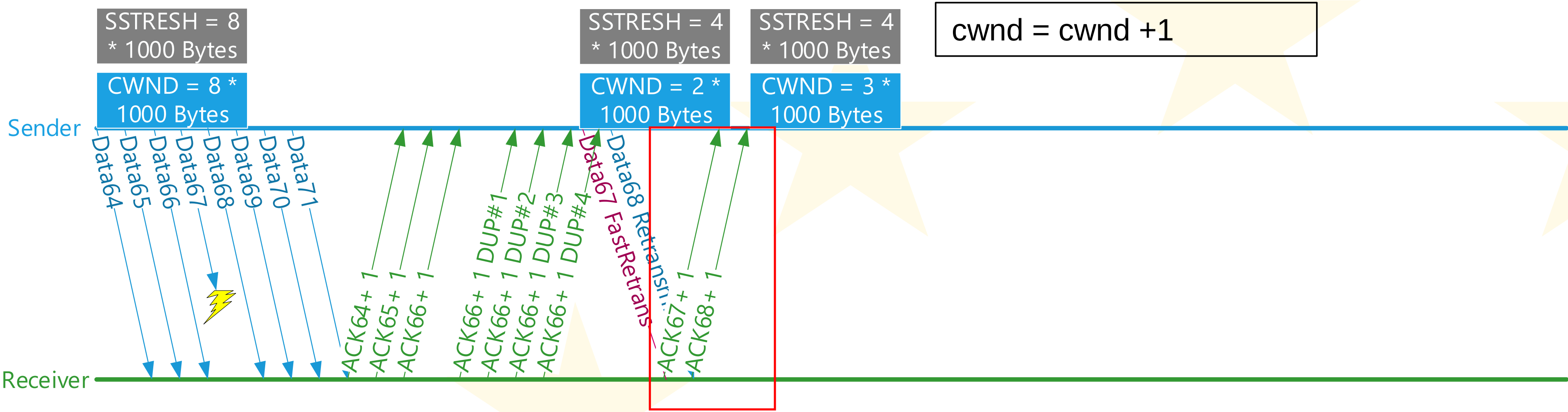
ssthresh = max (½ BytesInFlight, 2*SMSS)

cwnd = IW,
due to Tahoe algorythm,



TCP Tahoe: Fast Retransmit 4 / 5

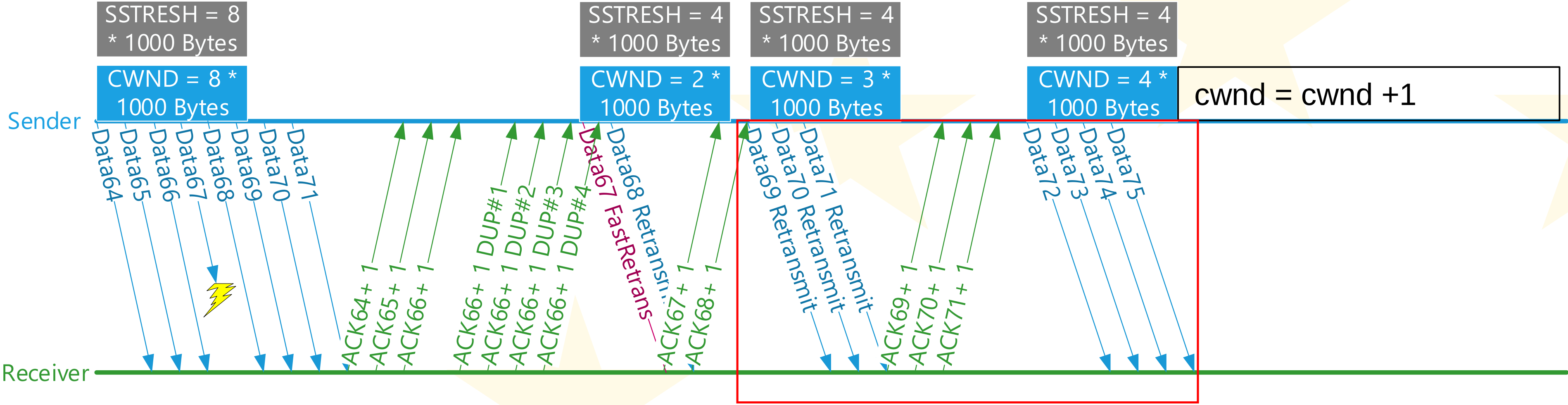
SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes



cwnd = cwnd +1

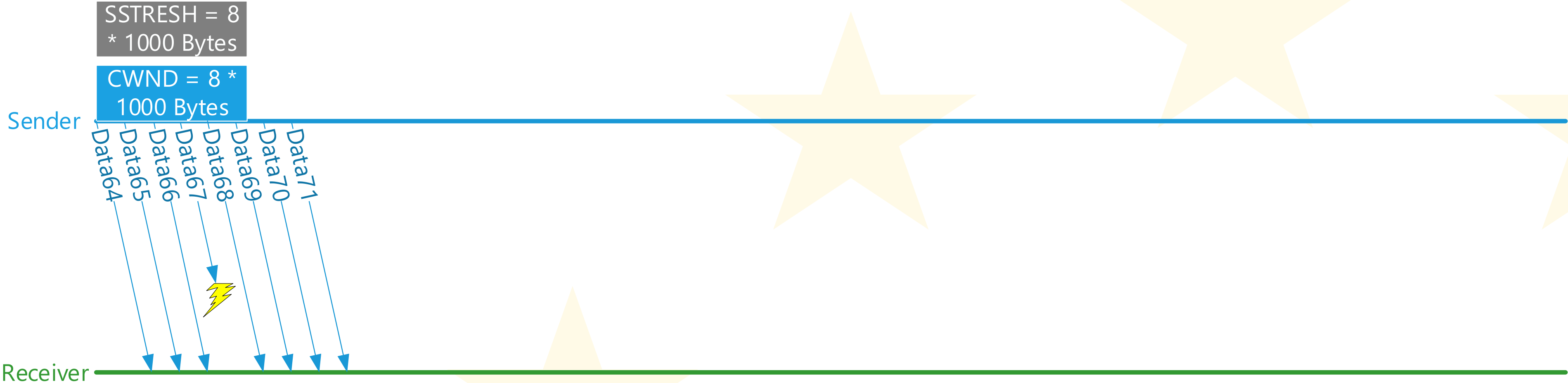
TCP Tahoe: Fast Retransmit 5 / 5

SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes



TCP Reno: Fast Recovery 1 / 5

SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes

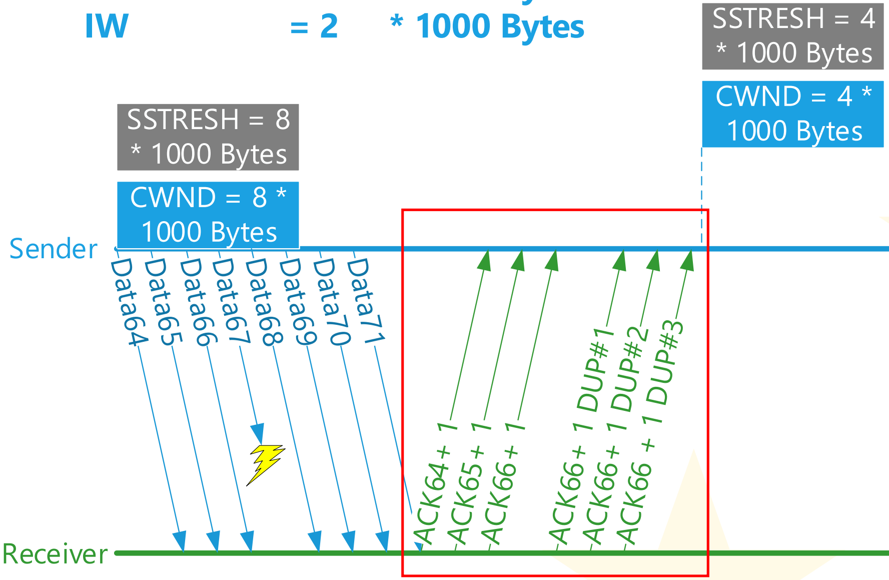


TCP Reno: Fast Recovery 2 / 5

SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes

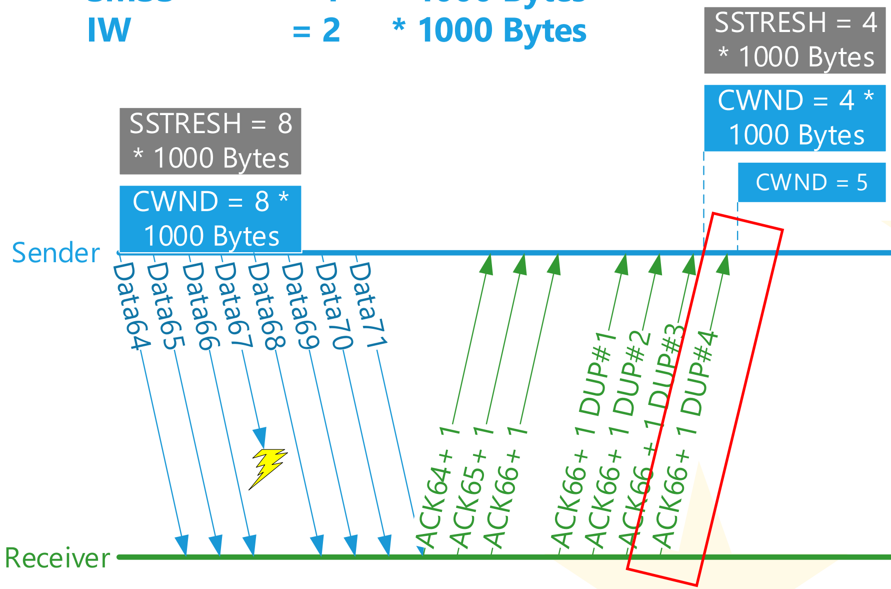
SSTRESH = ½ CWND

CWND = ½ CWND



TCP Reno: Fast Recovery 3 / 5

SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes

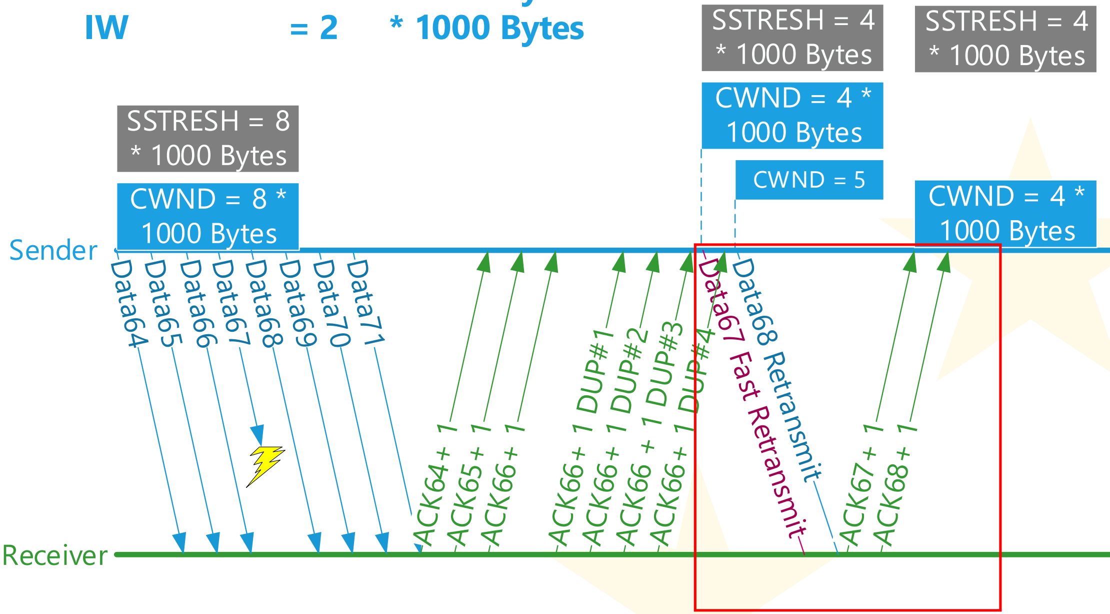


SSTRESH = ½ CWND

CWND = CWND + 1
by receiving on each DUP ACK after the DUP ACK #3

TCP Reno: Fast Recovery 4 / 5

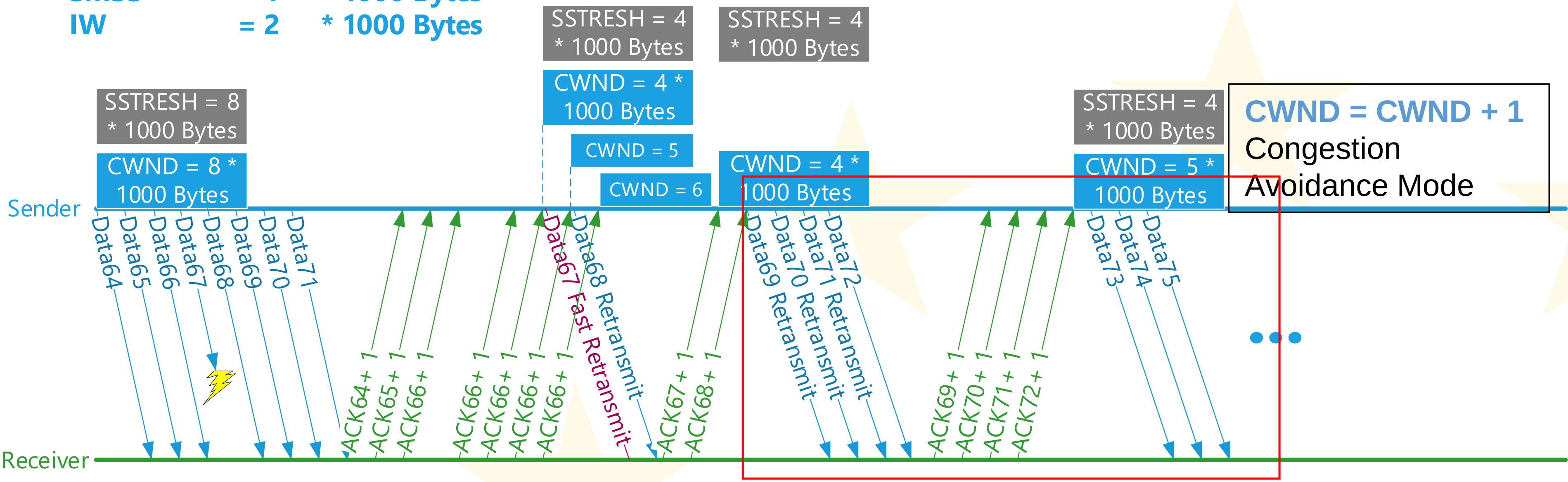
SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes


SSTRESH = 8 * 1000 Bytes
CWND = 8 * 1000 Bytes
SSTRESH = 4 * 1000 Bytes
CWND = 4 * 1000 Bytes
CWND = 5
SSTRESH = 4 * 1000 Bytes
CWND = 4 * 1000 Bytes
Sender
Receiver
Data64
Data65
Data66
Data67
Data68
Data69
Data70
Data71
ACK64+ 1
ACK65+ 1
ACK66+ 1
ACK66+ 1 DUP#1
ACK66+ 1 DUP#2
ACK66 + 1 DUP#3
ACK66+ 1 DUP#4
Data67 Fast Retransmit
Data68 Retransmit
ACK67+ 1
ACK68+ 1

CWND = SSTRESH
by receiving ACK for retransmitted Packet 67
-> Fast Recovery has ended

TCP Reno: Fast Recovery 5 / 5

SMSS = 1 * 1000 Bytes
IW = 2 * 1000 Bytes

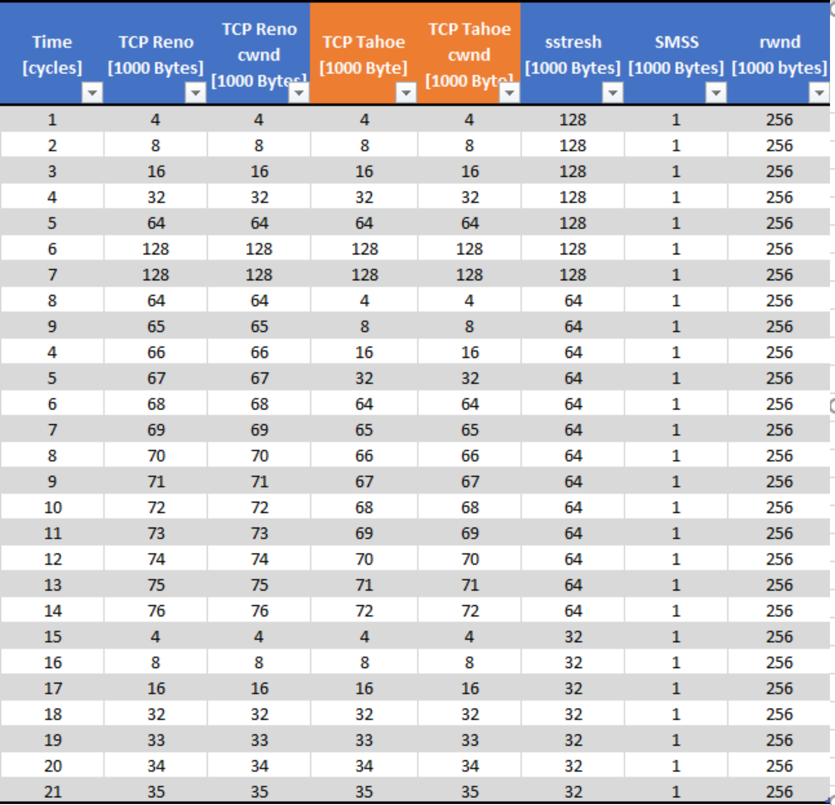


CWND = CWND + 1
Congestion Avoidance Mode

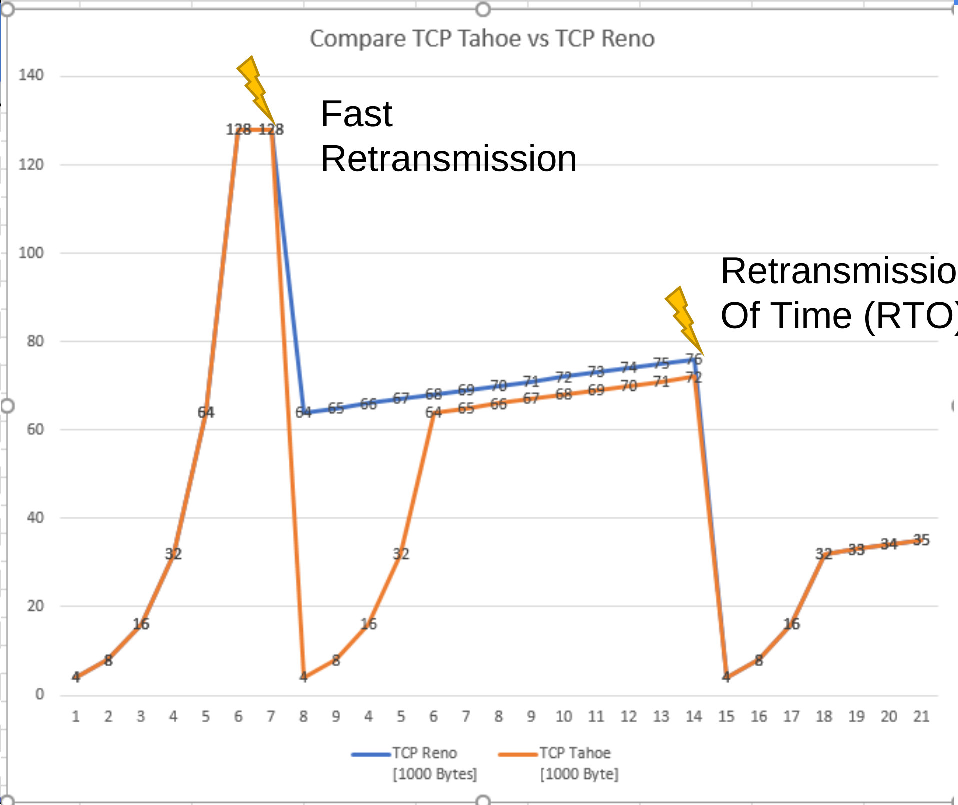
TCP New Reno vs. SACK

- Selective Acknowledgements (SACK)
  - Allows us to send DUP ACK to a specific Paket
    - E.g. we send 31, 32, 33, 34
    - Packet 32 is lost
    - With SACK we can say:
      - We are awaiting 35 but we are still missing 32
- New Reno introduces Partial ACK
  - Workaround when SACK is not supported

Tahoe and Reno Graphs



Time [cycles]	TCP Reno [1000 Bytes]	TCP Reno cwnd [1000 Byte	TCP Tahoe [1000 Byte]	TCP Tahoe cwnd [1000 Byt	sstresh [1000 Bytes]	SMSS [1000 Bytes]	rwnd [1000 bytes]
1	4	4	4	4	128	1	256
2	8	8	8	8	128	1	256
3	16	16	16	16	128	1	256
4	32	32	32	32	128	1	256
5	64	64	64	64	128	1	256
6	128	128	128	128	128	1	256
7	128	128	128	128	128	1	256
8	64	64	4	4	64	1	256
9	65	65	8	8	64	1	256
4	66	66	16	16	64	1	256
5	67	67	32	32	64	1	256
6	68	68	64	64	64	1	256
7	69	69	65	65	64	1	256
8	70	70	66	66	64	1	256
9	71	71	67	67	64	1	256
10	72	72	68	68	64	1	256
11	73	73	69	69	64	1	256
12	74	74	70	70	64	1	256
13	75	75	71	71	64	1	256
14	76	76	72	72	64	1	256
15	4	4	4	4	32	1	256
16	8	8	8	8	32	1	256
17	16	16	16	16	32	1	256
18	32	32	32	32	32	1	256
19	33	33	33	33	32	1	256
20	34	34	34	34	32	1	256
21	35	35	35	35	32	1	256



Fast Retransmission

Retransmission Of Time (RTO)

Congestion Avoidance

DEMO

- **FastRetransmission yes/no (SlowStart)**
- **Linear Growth**
- **ZigSawGraph**

TCP Reno Summary

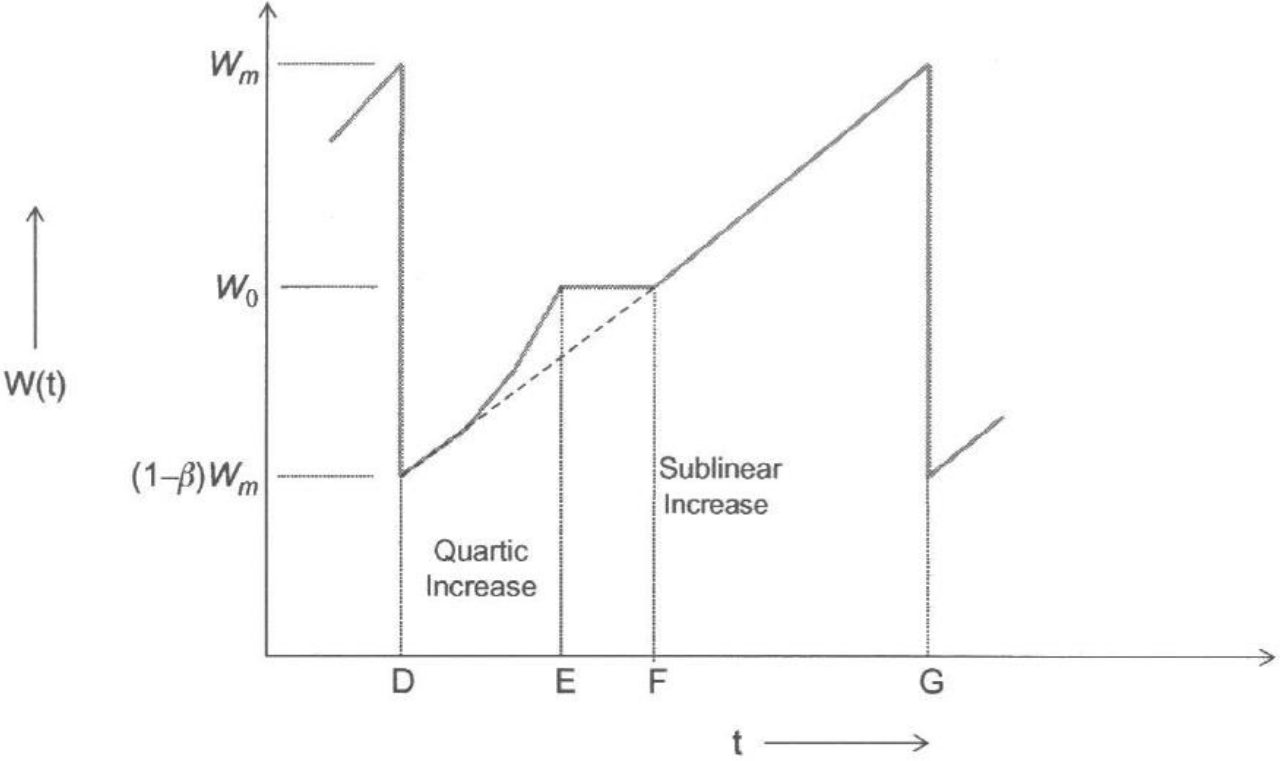
- **Fast Retransmit causes Fast Recovery:**
  - **cwnd = ½ cwnd + 1 per every DUP ACK > #3**
  - **ssthresh = max (BytesInFlight / 2,**
    2*SMSS)
  - Fast Recovery ends with receiving ACK for all lost packets with
    - cwnd = sstresh
  - **CWND growth after that**
    - **CWND = CWND + 1** per Window cycle
- **RTO causes:**
  - **ssthresh = max (BytesInFlight / 2,**
    2*SMSS)
  - **cwnd = Initial Window**

TCP Reno Summary

- If **cwnd** < **sstresh** then
  Slow Start is used
- If **cwnd** > **sstresh** then
  congestion avoidance is used
- After Packet Loss **sstresh** can never grow larger then **½ of cwnd**
- **1 Retransmitted can significant Slow Down whole Session**

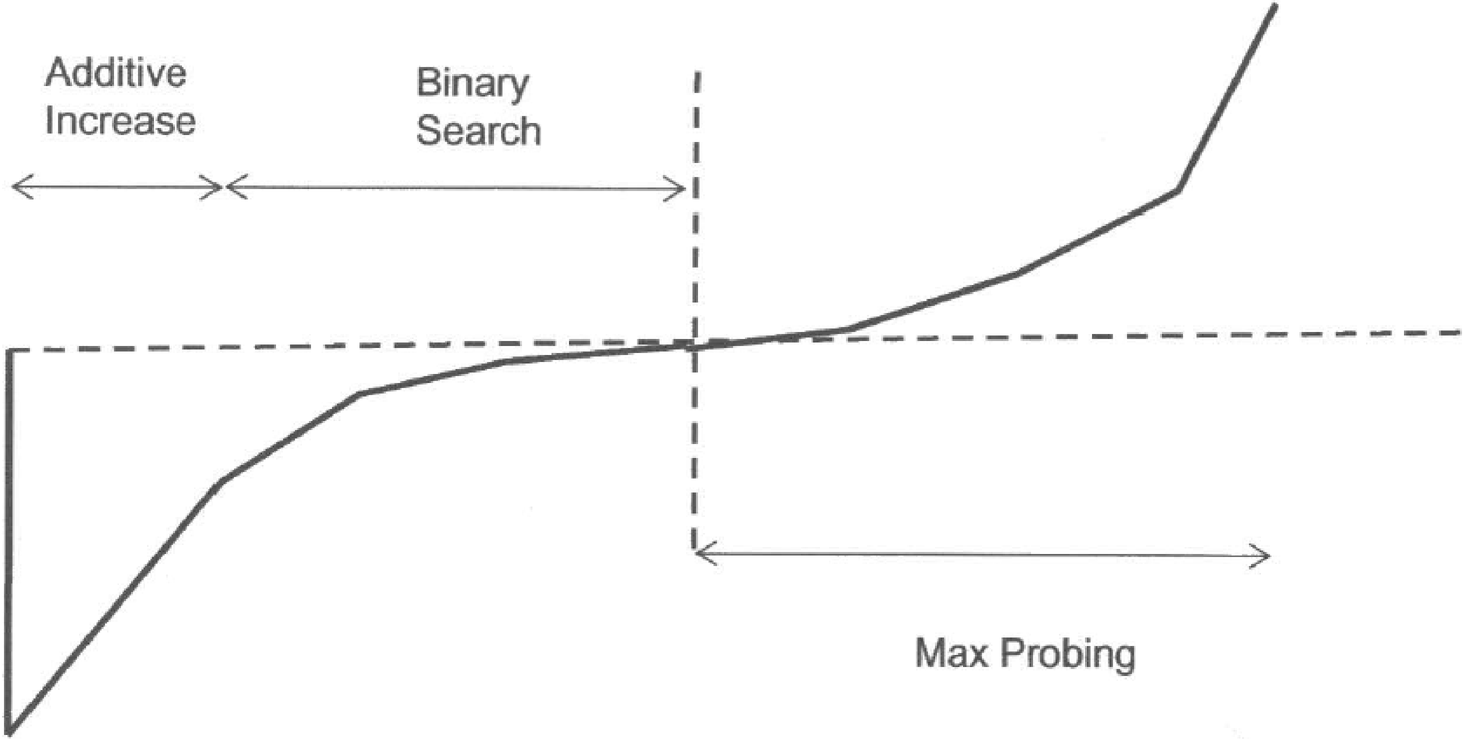
Further Congestion Algorythms

- **CTCP**



Further Congestion Algorythms

- BIC



Thank You!

- Thank you!

- I am happy about **FEEDBACK** at Sharkfest Europe Guidebook

About me?

- Christian Reusch
  - Analyzing Networks since 1999
  - Twitter: @crnetpackets
  - Web: crnetpackets.com

  - If you like you can send my Traces and I try to answer
    - creusch@crnetworks.com