

# Flow and Congestion Control for IS-IS

Guillaume Solignac – Bruno Decraene

`draft-decraene-lsr-isis-flooding-speed`

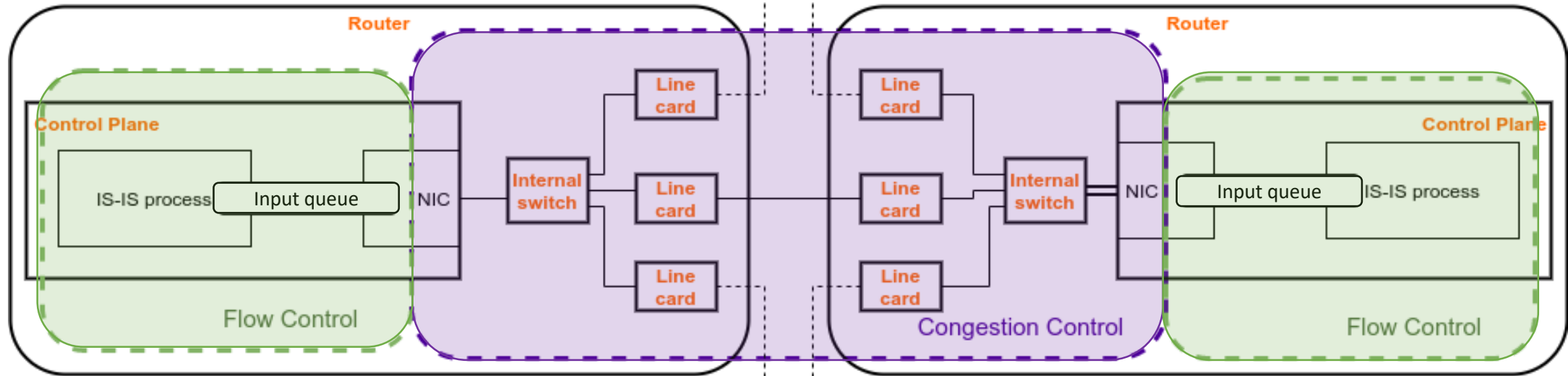
- 1. Goals, problem space**
2. Flow Control with a Receive Window
3. Operation modes

# Goals

- ❑ Faster flooding when the receiver has free cycles
- ❑ Slower flooding when the receiver is busy/congested
- ❑ Avoiding/minimizing the parameters the network operator has to tune
- ❑ Avoiding/minimizing the loss of LSPs
- ❑ Robust to a wide variety of conditions (Good & bad ones)
- ❑ Simplicity of implementation

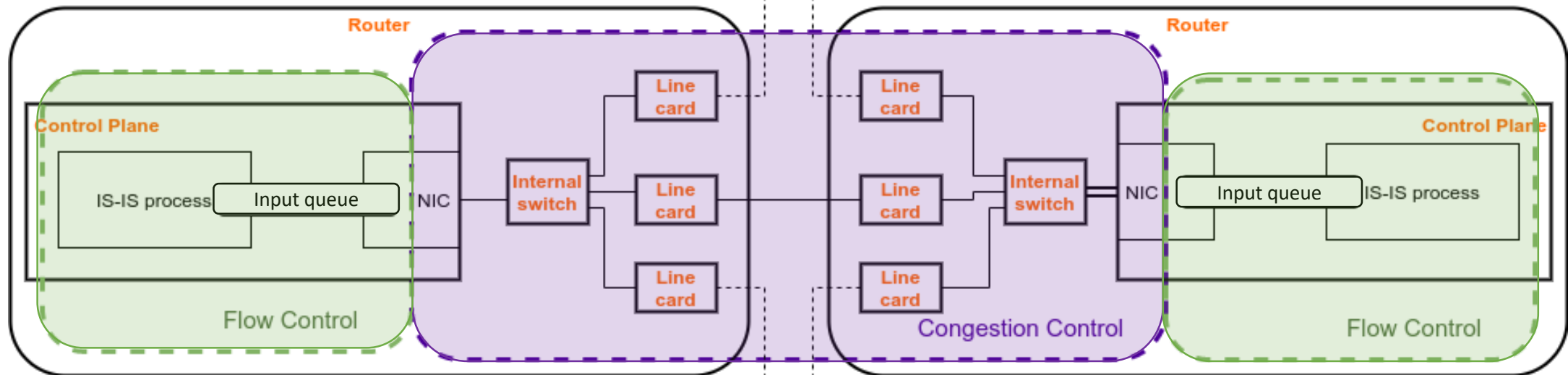
## 2-problem space

- “Control plane”: CPU speed & input queue on receiver IS process
- “Data plane”: IO path between both IS processes
- A very simplified model, to highlight the disjoint spaces



# 2 solutions

- **Flow Control:** a control loop between a single sender and single receiver.
  - Prevents the sender from overwhelming the receiver's control plane
- **Congestion Control:**
  - Prevents senders from overwhelming the receiver (any resource)
- For any I/O congestion, both drafts propose a congestion control.
  - No disagreement.
- For control plane congestion, flow control is the key difference between both drafts:
  - draft-decreaene: uses flow control with a receive window (RWIN) for E2E control
  - draft-ginsberg: uses congestion control to handle the E2E control

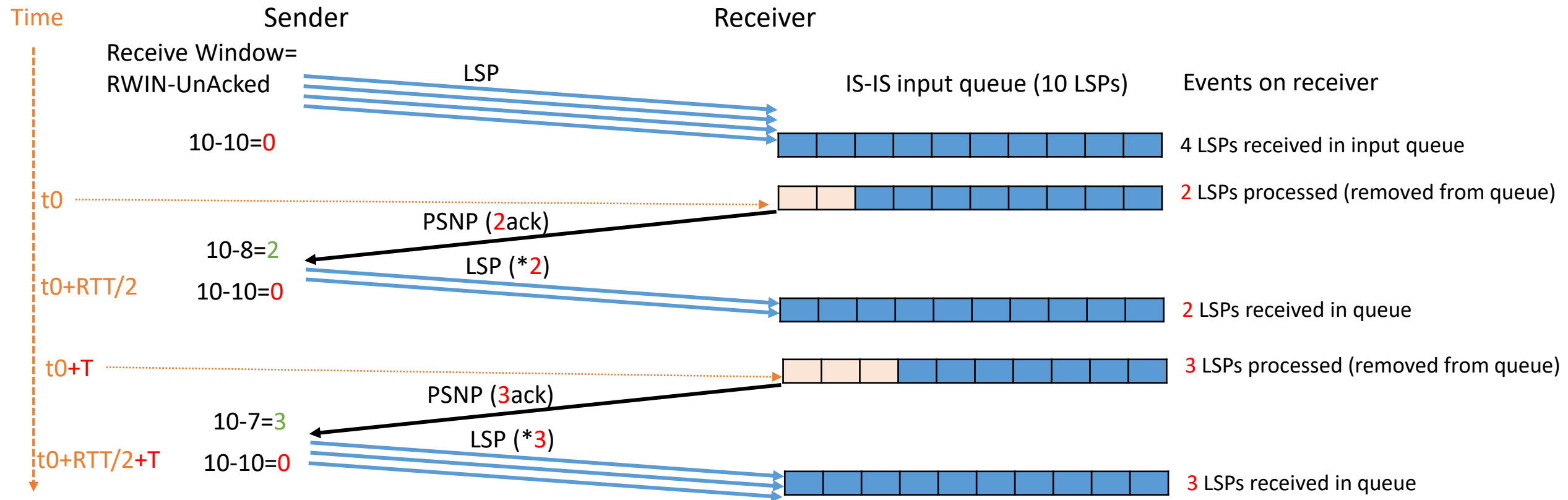


1. Goals, problem space
- 2. Flow Control with a Receive Window**
3. Operation modes

# Flow Control

- Goal : Avoid loosing LSPs in the control plane of the receiver.
  - Receiver max throughput reached, receiver halts because of another task...
- Classic Receive Window algorithm :
  - Receiver advertises a Receive window (RWin).
    - e.g., size of the control plane input queue allocated to this sender
  - Sender computes the number of unacknowledged LSPs (UnAked).
  - Sender pauses when  $\text{UnAked} \geq \text{Rwin}$
- New specification required on both sender and receiver
  - Advertise Receive Window in Hello PDU
  - Change PSNP behaviour to get faster feedback (otherwise only every n seconds)

# Flow Control



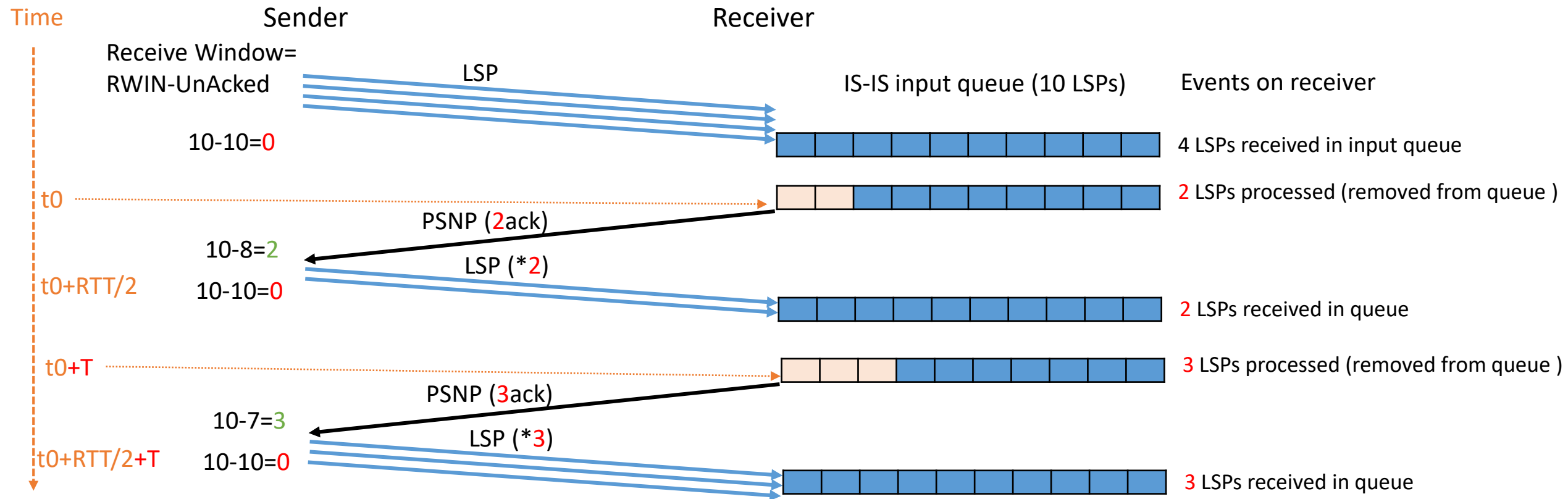
RWIN is static (e.g., 10)

On the sender, *UnAcked* is dynamically updated as PSNP are received (-) and LSP sent (+)

→ Dynamic flow control with a static RWIN



# Flow Control



- N** LSPs processed by receiver → **N** LSPs ack'ed PSNP by receiver → **N** LSPs sent by sender
- Number of LSP sent by sender (**N**) == Number of LSP (**N**) ack'ed. Chosen by the receiver.
  - Time between LSPs sent by sender (**T**) == Time between PSNPs (**T**). Chosen by the receiver
  - Rate of the sender (**N/T**) = Processing/acknowledging rate of the receiver (**N/T**)

Sender's behavior is driven by receiver with no additional signaling (just regular PSNP)

# RWIN provides interesting properties by design

- Sender cannot overwhelm the receiver control plane (CPU, input queue)
- Sender rate is controlled by the receiver and aligned with receiver performance.
- Sender pauses (durations, numbers) are controlled by the receiver

1. Goals, problem space
2. Flow Control with a Receive Window
- 3. Operation modes**

# Three modes of operation depending on receiver limit

- CPU bound
  - Receiver IS-IS rate is limited by its CPU
- I/O bound
  - Receiver IS-IS is limited by its incoming I/O path (max rate, or loss of LSP...)
- RTT bound
  - Rate is limited by RTT. (IS-IS RTT: link delay + time for the receiver to ack LSPs)
  - $\text{Rate} = \text{RWIN} / \text{RTT}$

# I/O bound: pick your favorite congestion control algo

- Not handled by flow control
  - In fact, indirectly and partially handled as loss of LSP reduces the available window hence tends to reduce the rate.
- To be handled by congestion control which is a local algo on the sender. E.g.,
  - draft-decreaene-lsr-isis-flooding-speed §5
  - draft-ginsberg-lsr-isis-flooding-scale-03 §3.1
  - Your favorite one (e.g., your current one)
- Pick your favorite shaping option
  - E.g., shape a given rate in bursts of N LSPs every S milli-seconds.

# CPU bound : RWIN provides optimal behavior

- By design, the flow control receive window provides :
  - No loss of LSP
  - Optimal rate (exactly matches the receiver's rate)
  - Any pause on the receiver translates in the same pause on the sender
- IOW, behavior is optimal by design.
- Also highlighted in our evaluations

Scenario	RTT	RWin	LPP	IO rate (LSP/s)	IO buffer	Bottleneck	Achieved rate (LSP/s)	% lost LSPs
10v1	1ms	25	1	80k	Inf	CPU	2000	0
10v1	1ms	50	1	80k	Inf	CPU	2000	0
10v1	1ms	100	1	80k	Inf	CPU	2000	0

# Without RWIN, sender adaptation may lead to loss of LSP

- Receiver slowing down leads to loss of LSP. E.g.,
  - 13% (300 LSP/s → to 100 LSP/sec)
  - 74% (1000 LSP/s → 100 LSP/sec)

Flooding	Start/End LSPTxRate	LSPTxMax	Time [ms]	Retrans
Base	33/33	NA	66268	0
Base	333/333	NA	20988	2439 (122%)
Optimized	300/100	300	20076	257 (13%)
Optimized	1000/100	1000	19456	1475 (74%)

Source: IETF 111, LSR WG, draft-ginsberg-lsr-isis-flooding-scale

- Receiver pausing leads to loss of LSP
  - No experimental data.
  - Likely around  $\text{SendingRate} * \text{Delay\_of\_the\_feedback\_loop} - \text{input\_queue\_on\_the\_receiver}$

# RTT bound: a rate limitation introduced by RWIN

- Cost of RWIN is that rate is limited by  $RWIN/RTT$  hence by RTT.
- Similar to TCP with Long Fat Networks but with significant differences
  - Network delay is a single link (not across the whole Internet)
  - All neighbors send the same LSPs hence one neighbor with a high RTT does not increase LSDB sync time in the presence of other neighbors with low RTT.
  - IS-IS implementations seem typically CPU bound so one neighbor with a high RTT does decrease receiver total capacity. Actually it increases the rate of other neighbors.
  - Control plane time to ack will be higher for IS-IS compared to TCP in kernel.
- This is the (main) cost introduced by RWIN in order to gain optimal behavior for the CPU-bound case
  - Needs receiver to send PSNP faster.
  - Seem like a theoretical limitation of the delay of the feedback loop: either you wait for knowledge or you guess (and be either too fast or too slow)



# Conclusion

- Key difference between both drafts is the flow control based on RWIN proposed in draft-decreane-lsr-isis-flooding-speed
- Key benefit: when the receiver is CPU-bound, the sending behavior is optimal
  - No loss of LSP
  - Exact max rate supported by receiver
  - A pause on the receiver processing translates into the same pause on the sender.
- Key cost: when the receiver is RTT-bound, sending rate is sub-optimal (but with no loss of LSP)
  - Requires a change on the receiver (fast ack)
  - Rate may be reduced with a high RTT link (alternatively high RWIN required)
- Both points are confirmed and quantified by experiments

# Next steps

- We believe that in many cases receiver is CPU-bound rather than RTT-bound hence RWIN is very applicable
  - Low RTT link are a majority (intra-cities, intra-region, intra-country, intra-DC)
  - IS-IS implementations seem primarily CPU bound
    - Relatively easy to reach the limit especially when flooding over 10s of IS-IS adjacencies.
- We see no reason to forbid flow control with RWIN
- Asking for WG adoption
  - LSR IS-IS experts have denied code point allocation for our implementation because draft was not a WG document.

Thank you

# Appendix

## **2. Congestion control**

# Congestion control

- congestion control is a local algo on the sender. E.g.,
  - draft-decreaene-lsr-isis-flooding-speed §5
  - draft-ginsberg-lsr-isis-flooding-scale-03 §3.1
  - Your favorite one (e.g. your current one)
- Congestion control algorithm does not seem like a differentiator between drafts
  - Not a priority to discuss at this point

# How to choose RWin ?

By order of preference :

1. Input queue size / 2
2. Use TCP value (used by BGP)
3. Conservative value (10) – Existing default in a popular implementation

Software parameter, hardware-independent.

# When to send a PSNP ?

- Every **LPP** LSPs received.  $LPP = \text{LSPs per PSNP}$
- Timeout

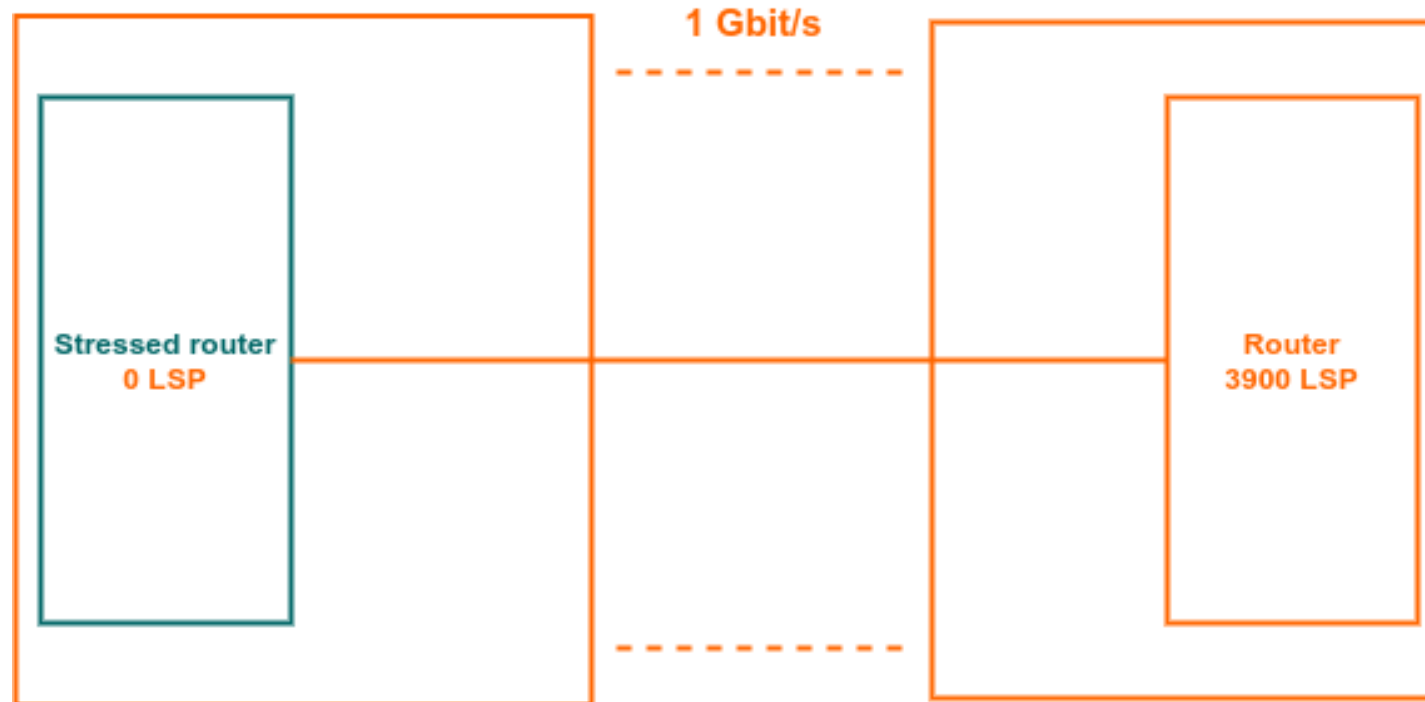


# Flow Control

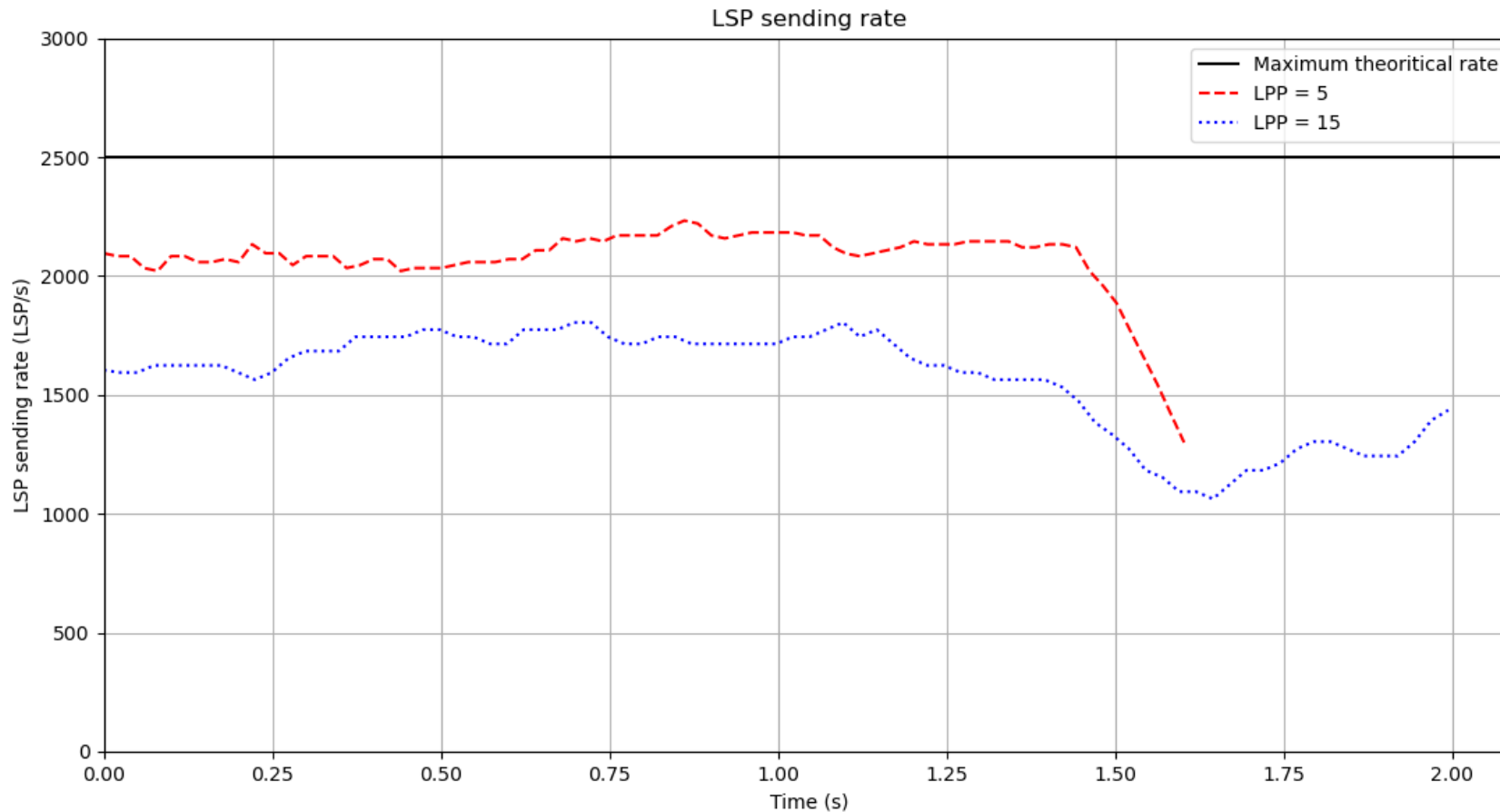
- Relies on one static information : size of socket buffer
- Multiple identified parameters influence behavior :
  - RWin
  - RTT
  - Number of LSP per PSNP (LPP)

$$rate < \frac{RWin}{RTT} = \text{Theoretical rate}$$

# Experimental setup – 1vs1

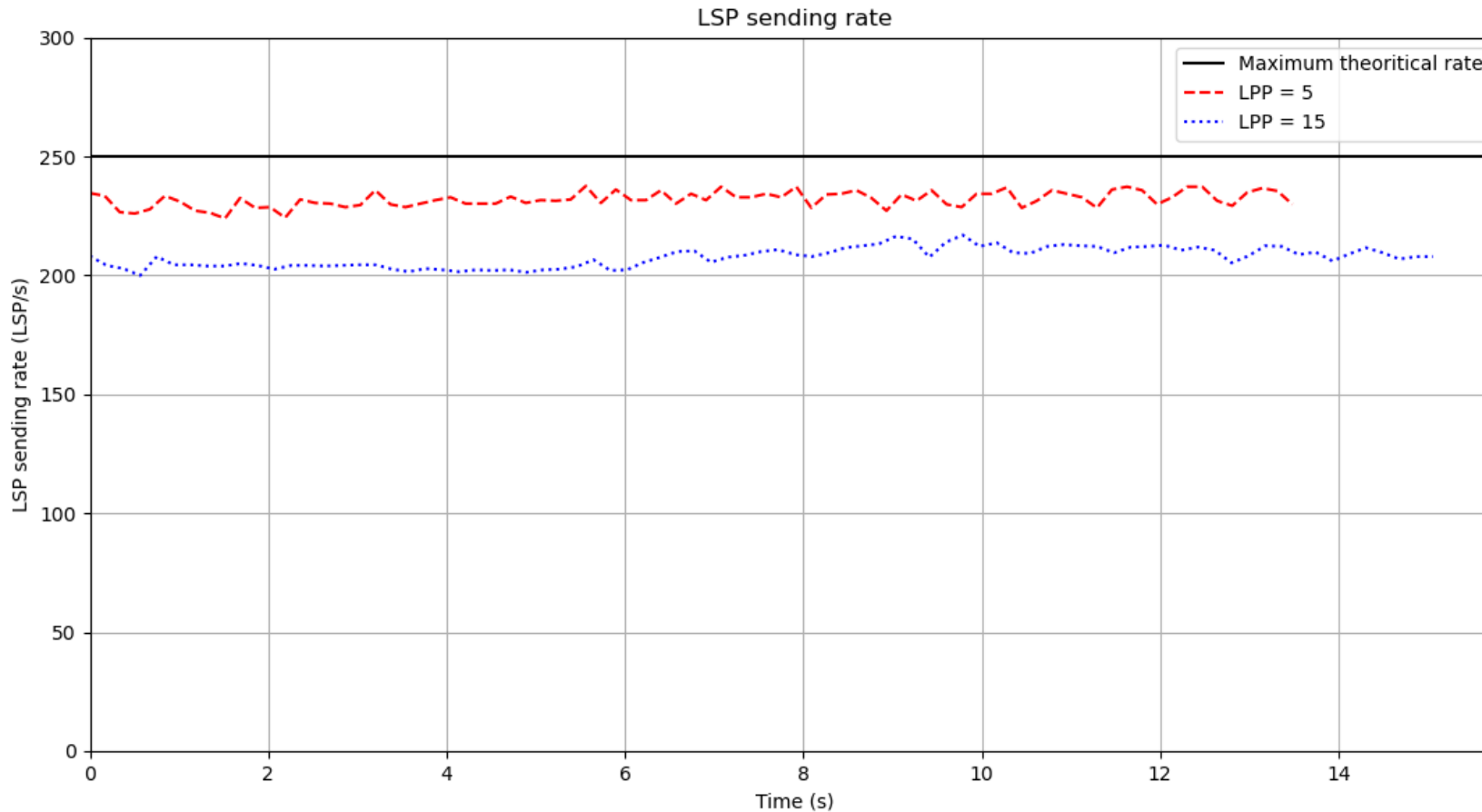


# Flow Control – Results 1vs1 – RTT 20ms – RWin 50



- Close but lower than theoretical rate (50/20ms = 2500 LSP/s), more on that later
- No LSP loss

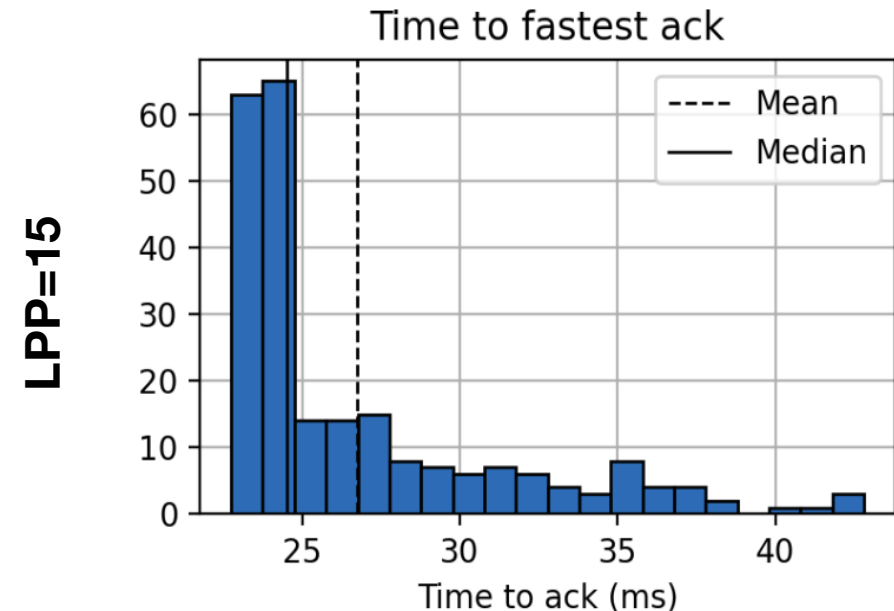
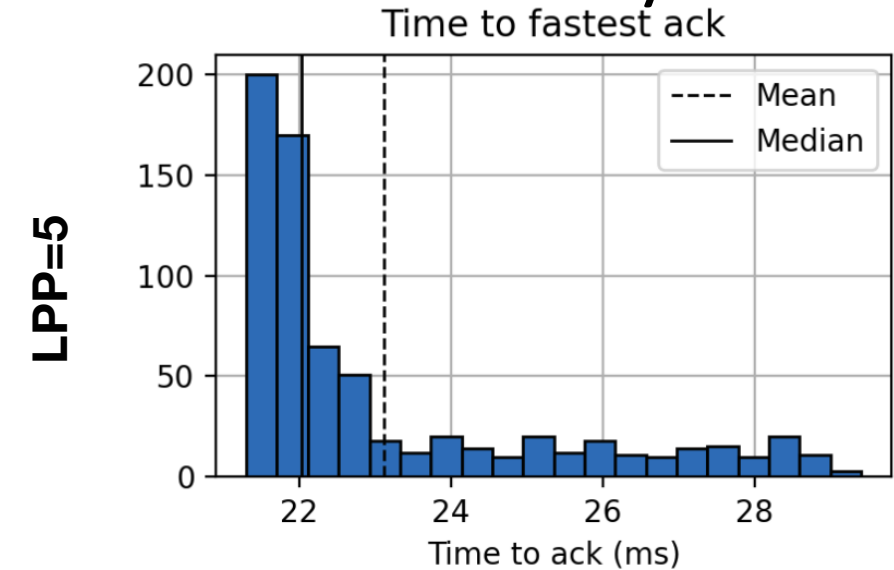
# Flow Control – Results 1vs1 – RTT 200ms – RWin 50



- Close to theoretical rate ( $50/200\text{ms} = 250 \text{ LSP/s}$ )
- No LSP loss

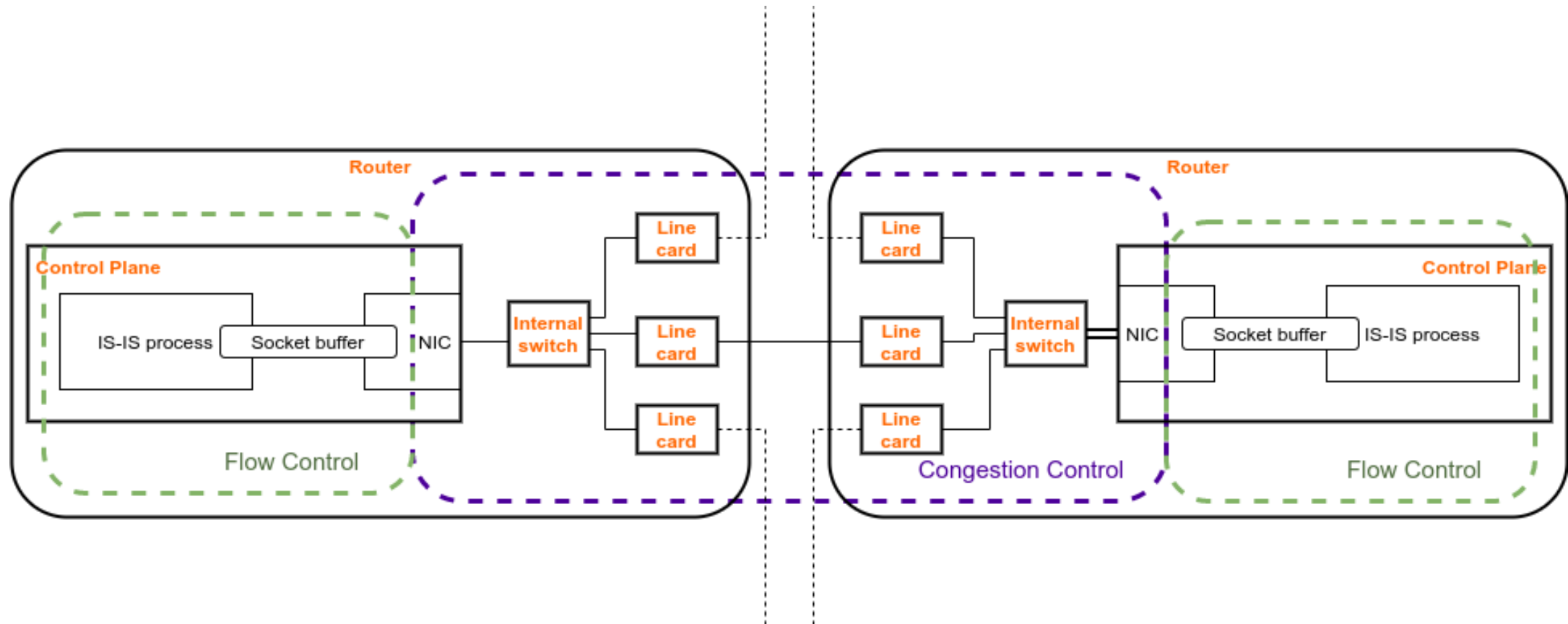
# Flow Control – Results 1vs1 – PSNP delay

- Metric : inside a PSNP, time to latest included LSP
- Effective RTT is higher than 20ms due to computation time of LSPs & PSNP crafting
- RWin should be a multiple of LPP
  - Avoids delaying unfilled PSNP

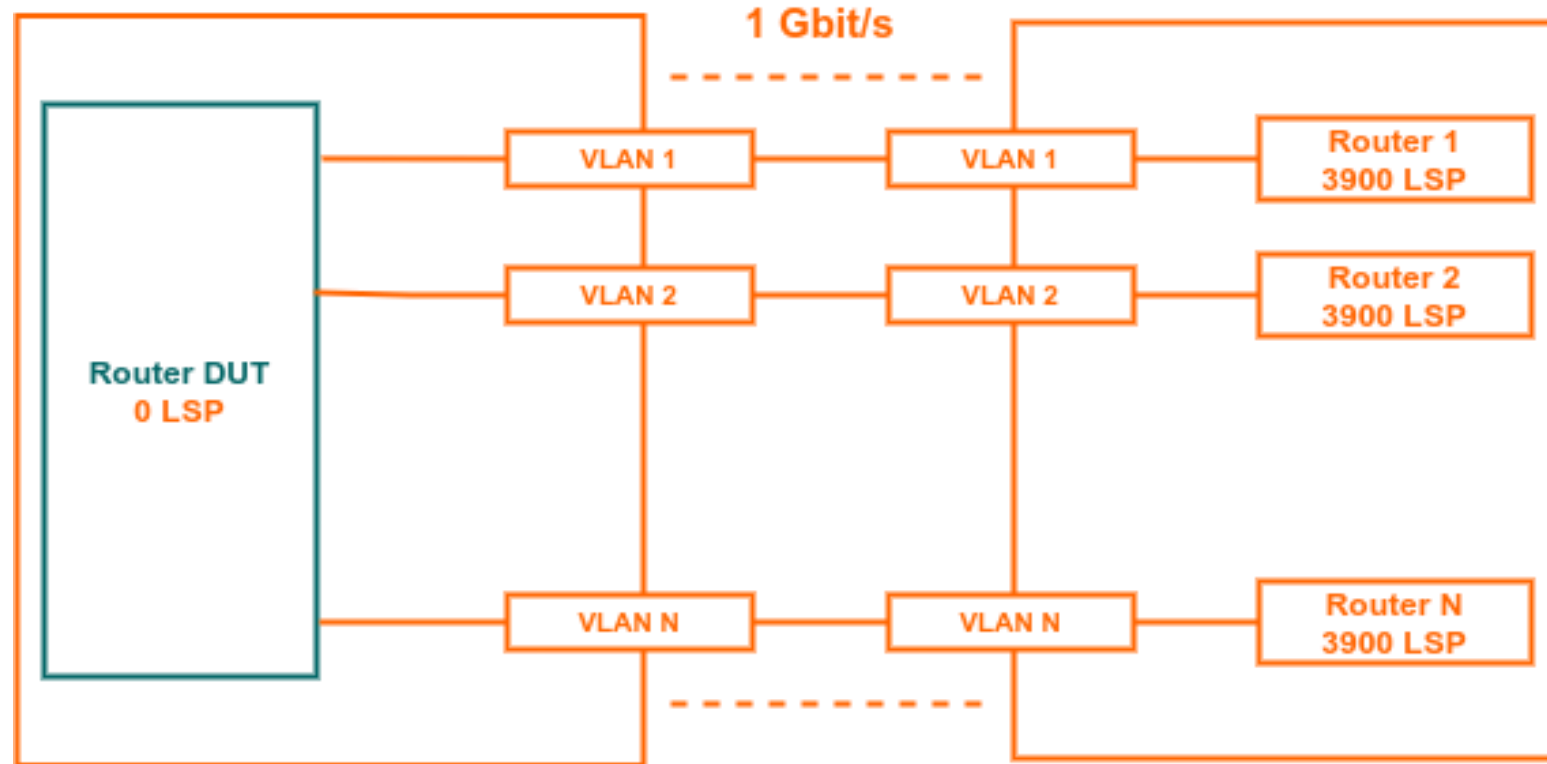


# Increased sending rate hazard

Increased sending rate means more stress, thus potentially losses in IO paths.

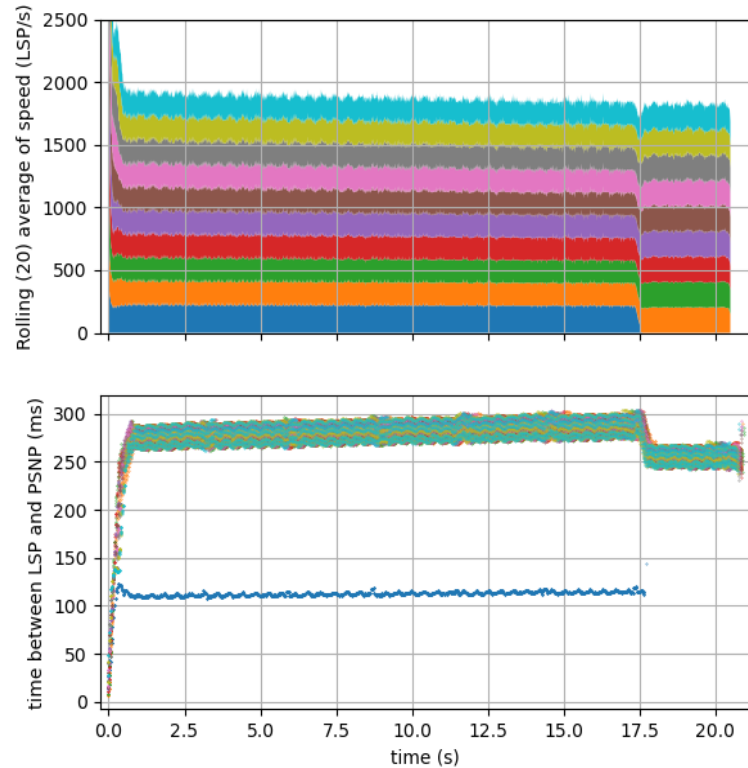


# Experimental setup – Nvs1

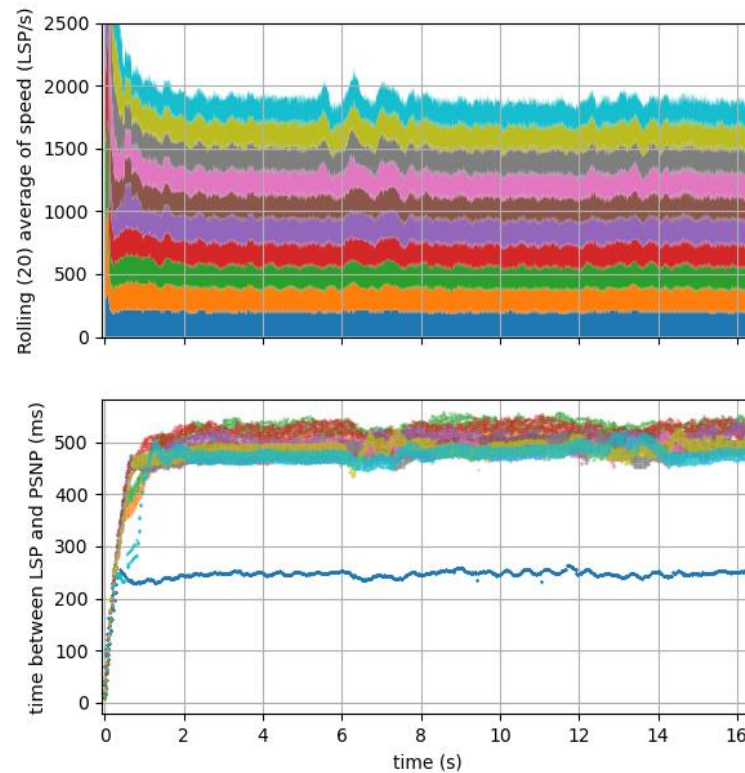


# Flow Control – Results 10vs1– LPP = 1 – RTT = 1ms

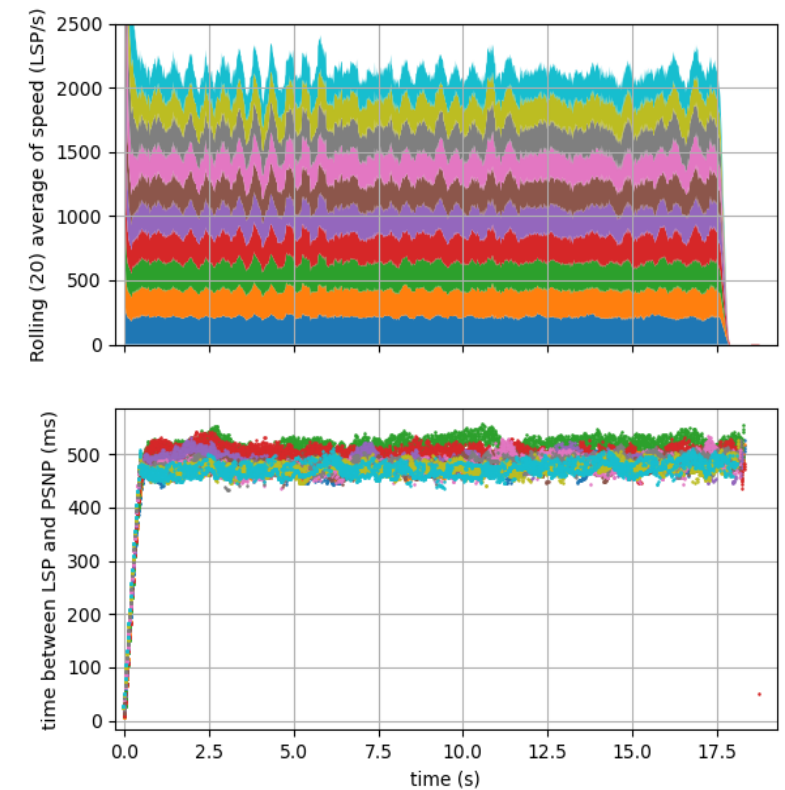
**RWin=25**



**RWin=50**



**RWin=100**



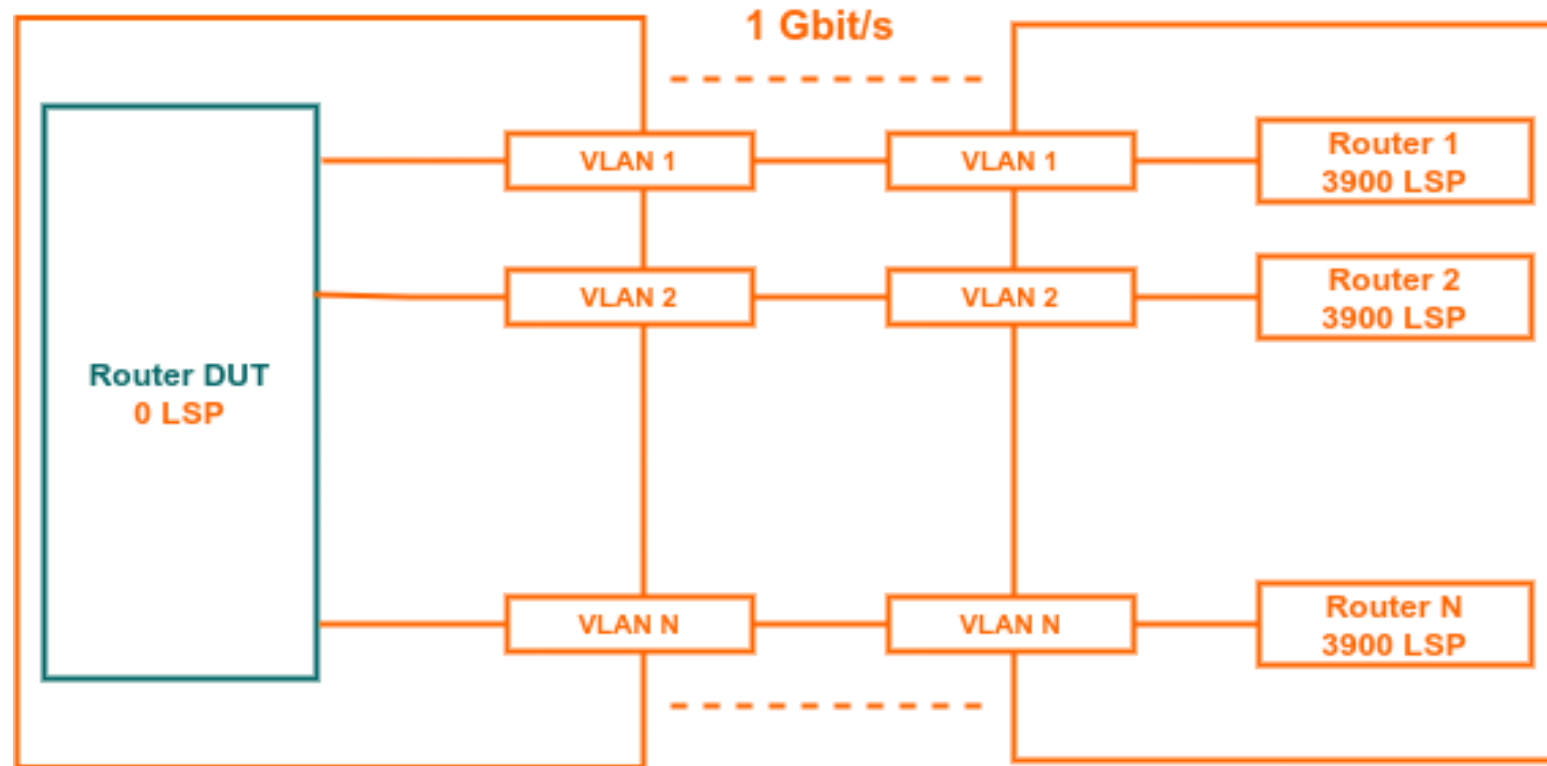
- Good point : no LSP loss, senders pace well  $\frac{200 \text{ LSP/s}}{30 \text{ LSP/s}} = 6.7$  speedup compared to default rate
- CPU Bound. Increasing Rwin only increases latency.

$$\text{max\_rate} = \frac{\text{RWin}}{\text{RTT}}$$



## **2. RWin with IO bottleneck**

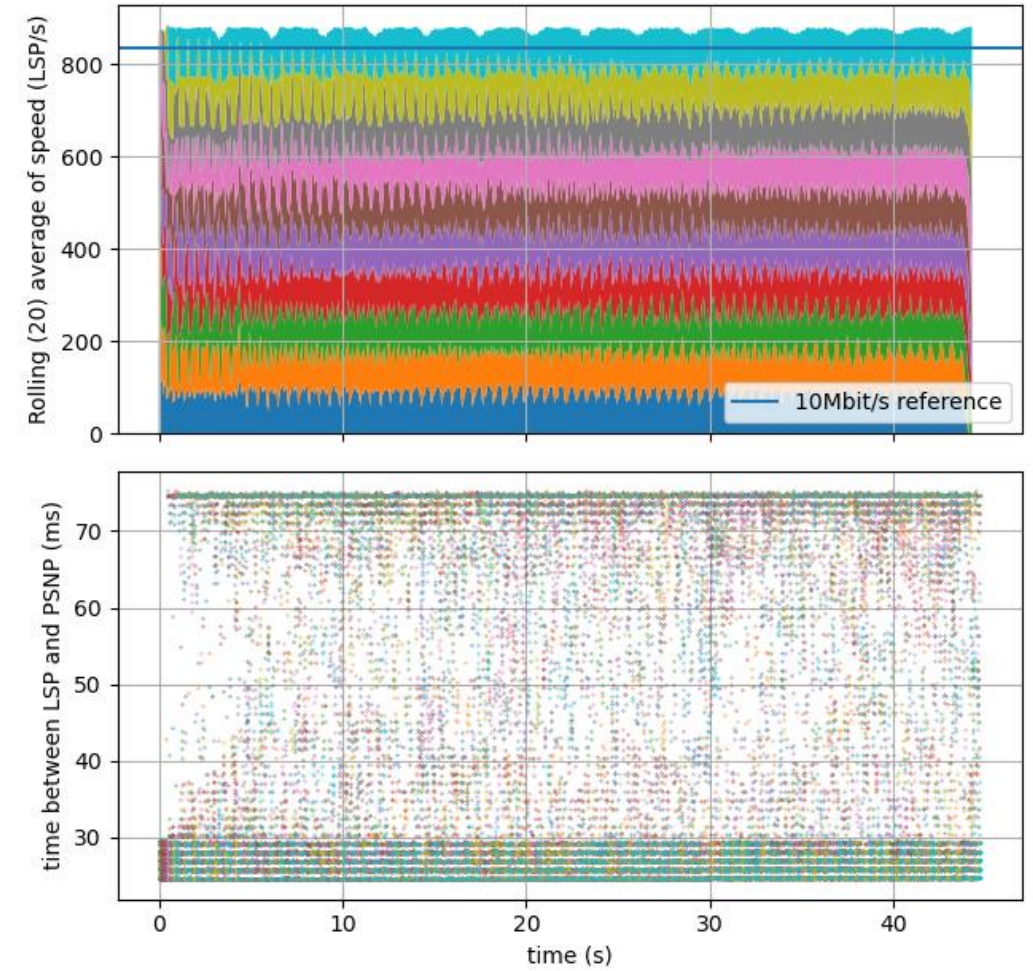
# Experimental setup – Nvs1



Flow Control – Results 10vs1 – LPP = 5 – RTT = 25ms  
Bottleneck = **10Mbit/s**, buffer = 2600 packets

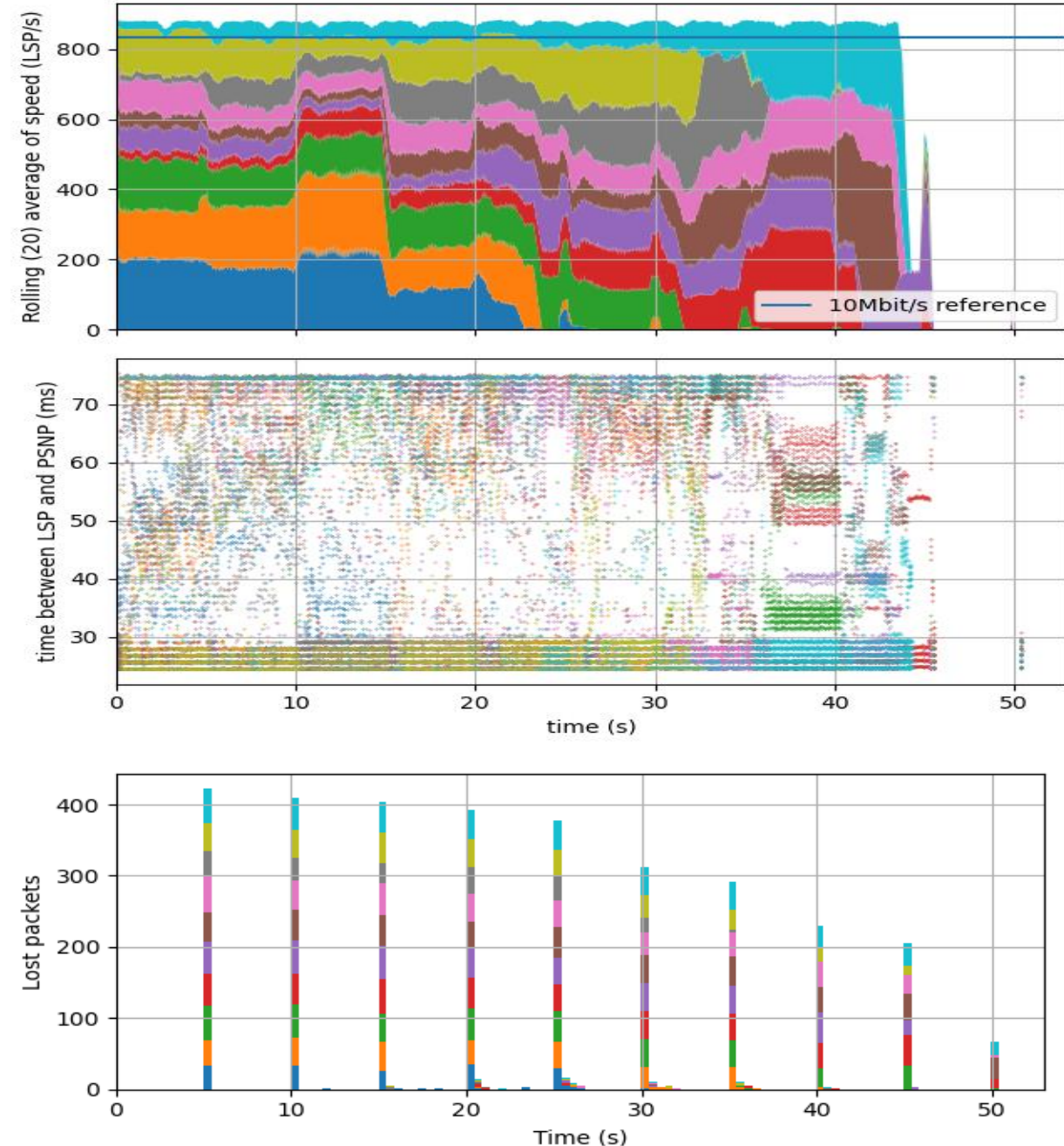
- Good point : no LSP loss because :

$$\begin{aligned} \text{nb\_senders} * \text{RWin} &= 10 * 50 \\ &< 2600 = \text{bottleneck\_buffer} \end{aligned}$$



Flow Control – Results 10vs1 – LPP =  
5 – RTT = 25ms  
Bottleneck = 10Mbit/s  
buffer = **64** packets

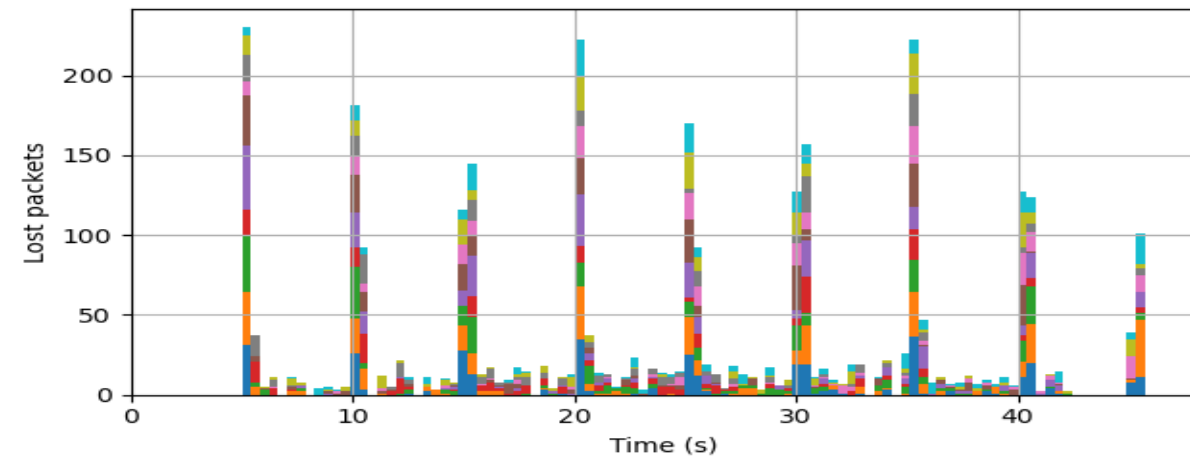
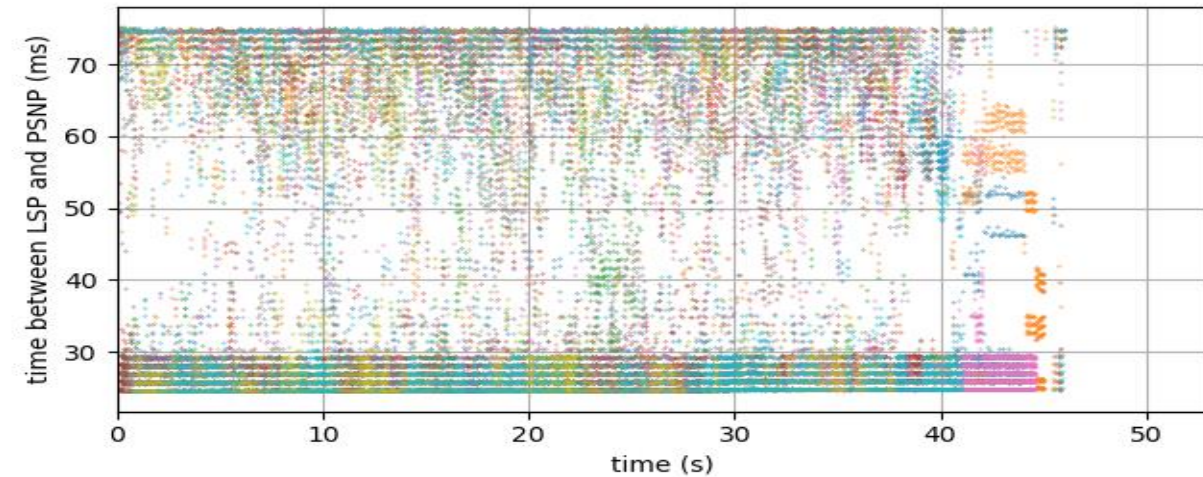
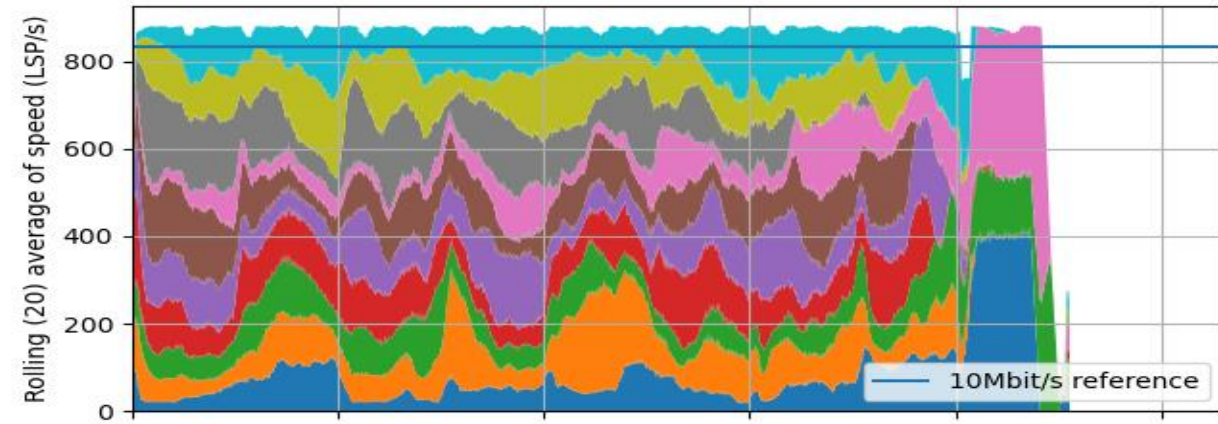
- 3218 losses (8.2% of 39000 LSPs)
- Losses are bursty : corresponds to the LSP timeout
- At every burst, approx.  $500 - 64 = 436$  LSPs lost.





Flow Control – Results 10vs1 Multiple  
senders – LPP = 5 – RTT = 25ms  
Bottleneck = 10Mbit/s, buffer = 64  
packets, **slow-start**

- 3023 losses (7.7% of 39000 LSPs)
- Slow start helps slightly mitigating the burstiness



# RWin algorithm behavior

- 3 sending rates possible :
  - RWin limit ( $\text{RWin} / \text{RTT}$ )
  - CPU limit
  - IO limit
- Effective rate =  $\min(\text{RWin}/\text{RTT}, \text{CPU}, \text{IO})$
- First two cause no LSP loss – and we think CPU will be the bottleneck in most (if not every) implementations today
- Only IO bottleneck is addressed by congestion control

# Flow Control – Results 10vs1 Multiple senders

Recap :

- Cost:
  - New TLV
  - Socket buffer
  - More PSNPs (with LPP=15, 6 times more PSNPs)
- Gain:
  - No LSP loss due to socket buffer exhaustion
  - Speed paced by receiver ACKs → “CPU congestion” is dealt with
  - Dropped LSPs artificially fills RWin → “Internal congestion” causes speed to drop –which is good-.

# **3. Congestion Control with CWin**



# Congestion Control algorithms

Extensively studied in the case of TCP

3 steps : slow-start, end of slow-start, congestion avoidance

Various approaches : losses, delay, bandwidth

Losses : for TCP, packet reordering (not available here) & timeouts

Delay : Try to detect queuing delay, not necessarily good here because IS-IS processing time will be the bottleneck in many cases → not a general solution

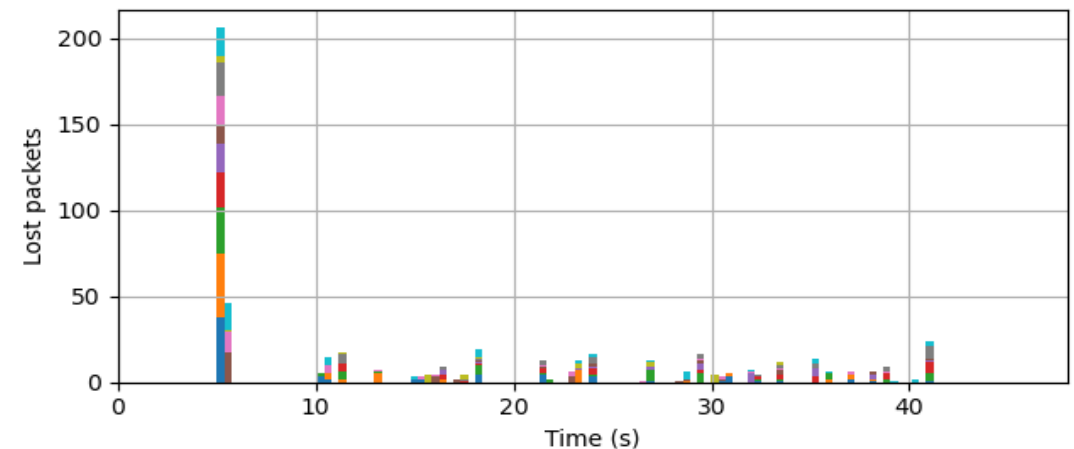
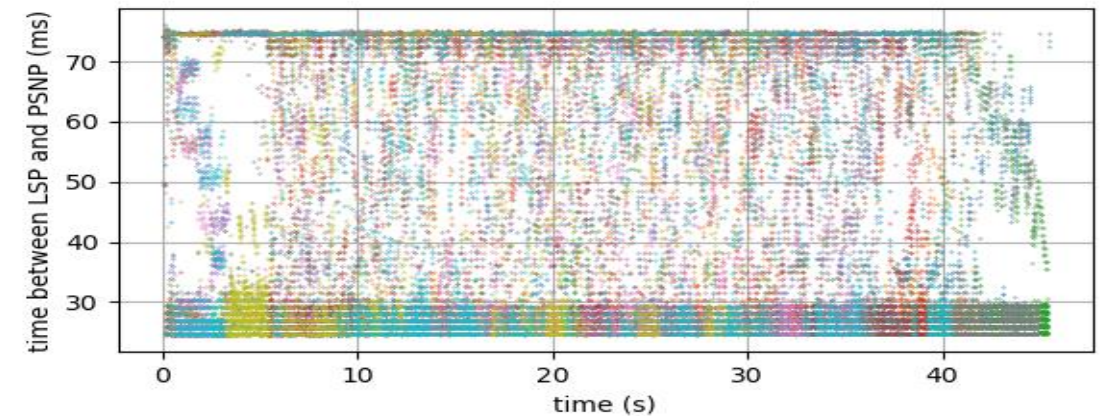
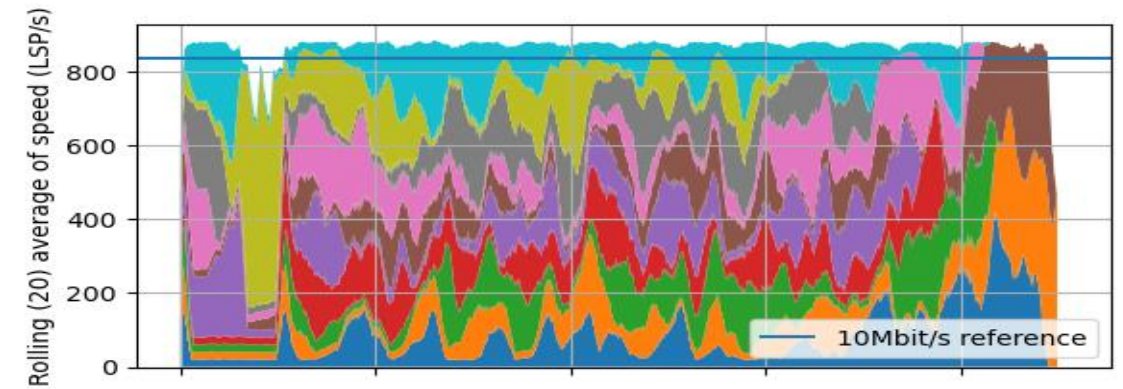
Bandwidth : interesting but needs enough data to stress the bottleneck. Unfavorable case for IS-IS as it depends on the neighbors.

Overall, the more reactive, the more cycles/memory is needed

The algorithm tries to deal with buffers. But for internal IO-path, might be very small buffers → hard to deal with in any case.

Flow Control – Results 10vs1 Multiple  
senders – LPP = 5 – RTT = 25ms  
Bottleneck = 10Mbit/s, buffer = 64  
packets, **congestion avoidance**

- 524 lost LSPs (1.3% of 39000 LSPs)
- Congestion control helps a lot in avoiding losses
- Large overshoot at the end of the slow-start (252 losses on first round only)
- Slow-start could be removed but helps scaling to larger links (otherwise rate of growth is slow)



# Why not only CWin/Congestion Control ?

- Implementations are likely to be CPU bound : RWin is perfect for this case.
- CPU can be busy doing something else than processing LSPs : RWin will naturally pause the sending, thus avoiding losses that a CC algorithm alone will take some time to detect.
  - e.g BGP, SPF + TI-LFA +  $\mu$ -loop, C-SPF TE
- Congestion control will have to loose packet to detect CPU slowness. It will detect CPU busyness as congestion while it really **is not**.
- They can work together !

# Recap

Scenario	RTT	RWin	LPP	IO rate (LSP/s)	IO buffer	Bottleneck	Achieved rate (LSP/s)	% lost LSPs
1v1	20ms	50	5	80k	Inf	RWin	~2500	0
1v1	200ms	50	5	80k	Inf	RWin	~250	0
10v1	1ms	25	1	80k	Inf	CPU	2000	0
10v1	1ms	50	1	80k	Inf	CPU	2000	0
10v1	1ms	100	1	80k	Inf	CPU	2000	0
10v1 – RWin	25ms	50	5	833	2600	IO	860	0
10v1 – RWin	25ms	50	5	833	64	IO	860	8.2%
10v1 RWin + Slow-start	25ms	50	5	833	64	IO	860	7.7%
10v1 CWin + RWin	25ms	50	5	833	64	IO	860	1.3%

# Recap

- ✓ Faster flooding when the receiver has free cycles
- ✓ Slower flooding when the receiver is busy/congested
- ✓ Avoiding/minimizing the parameters the network operator has to tune
- ✓ Avoiding/minimizing the loss of LSPs
- ✓ Robust to a wide variety of conditions (Good & bad ones)
- ✓ Simplicity of implementation

# Recap

	RWin only	CWin <u>only</u>
CPU congestion on Receiver CP	Ok : RWin bounded and lower than socket size	Partial : CPU availability can change very fast
IO Bottleneck	Partial : losses bounded by sum of advertised RWins; lost packets trash RWin, inducing speed decrease	Ok (with hypothesis)
CPU resources	Low cost	Increases with algorithm complexity
Memory resources	Known buffer size for RWin	Increases with needed state

# Default optimized Fast-flood parameters

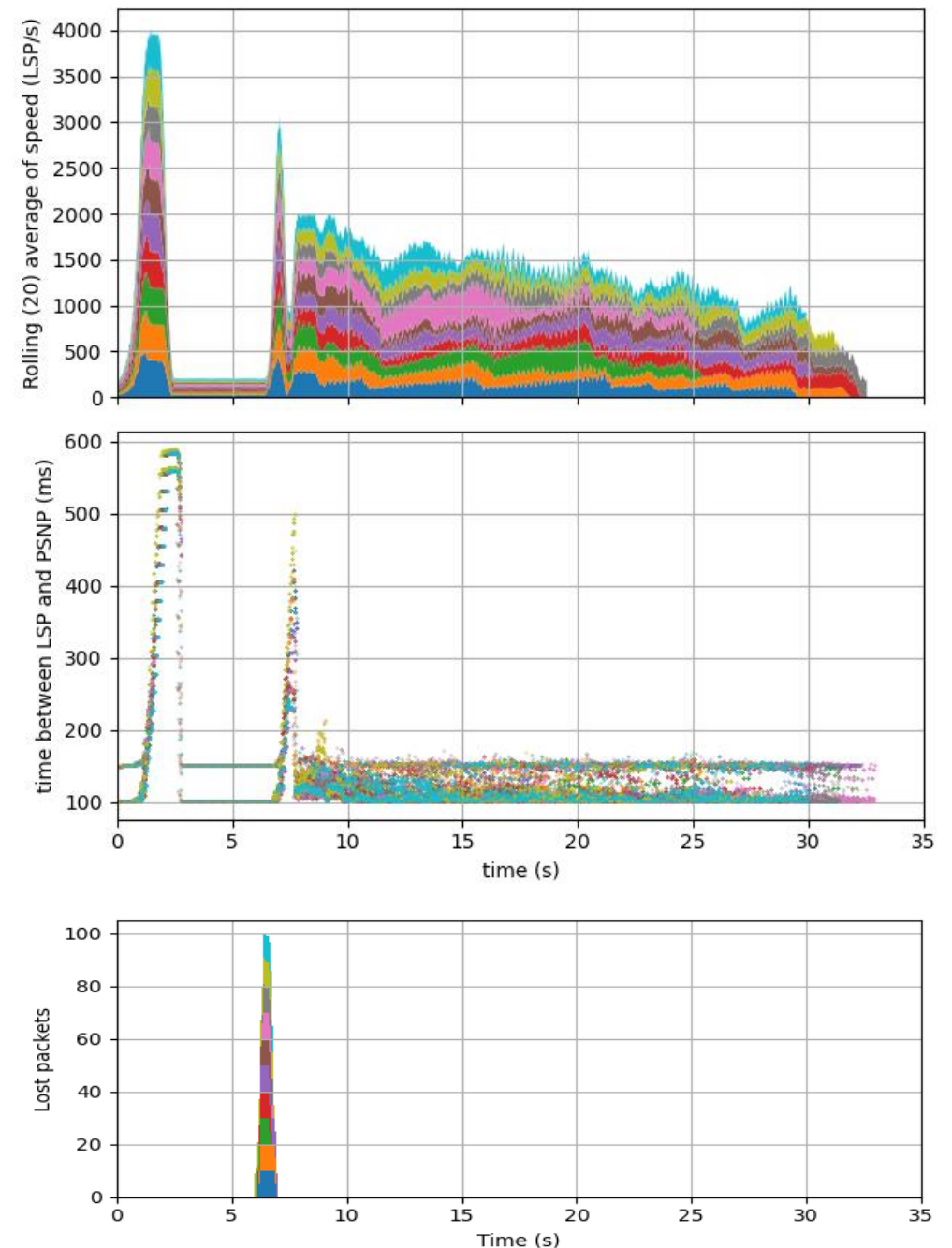
## Default Optimize Values for IS-IS

The following table summarizes the configuration impacted by default optimize:

IS-IS command	Parameters	Default optimize disabled	Default optimize enabled
fast-flood			
	# of lsps flooded back-back	Disabled	10

Flow Control – Results 10vs1 Multiple  
senders – LPP = 5 – RTT = **200ms**  
Bottleneck = 1Gbit/s, buffer = 2000  
packets, **congestion avoidance only**

- 1203 lost LSPs (3,1% of 39000 LSPs)
- Overshoot is needed to evaluate the speed
- Danger is when  $\#inflight > \text{socket size}$ . Here, to achieve 2000LSP/s, CWin must reach  $2000\text{LSP/s} / 0.2\text{s} = 10000$  LSP, way more than the socket size.
- This problem increases as processing goes faster and RTT goes up (more inflight packets), and when ACKs are slow.





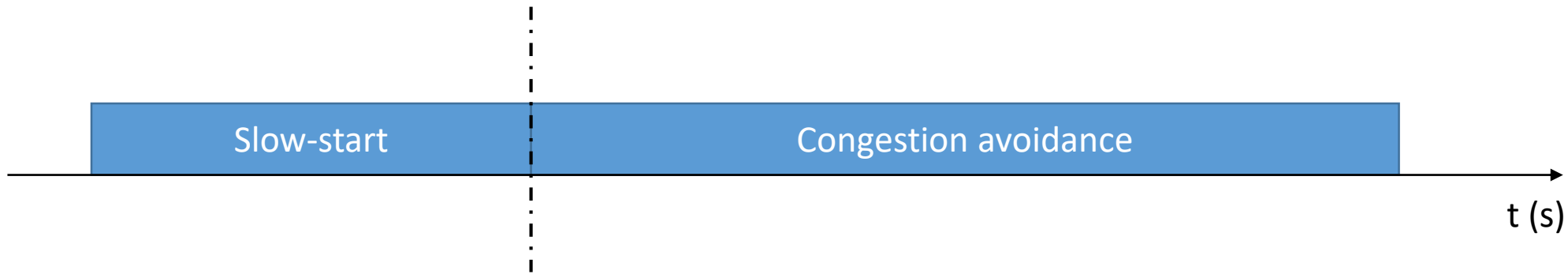
# Flow Control – Results 10vs1 Multiple senders

- LSP retransmitted
- per VLAN :

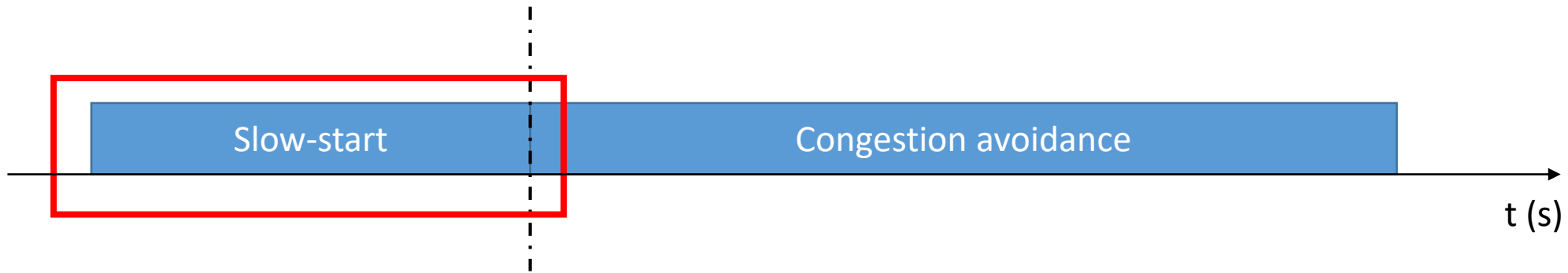
RWin	Retransmitted LSP Sender -> Receiver	Retransmitted LSP Receiver -> Sender
25	0	0
50	0	141
100	30	2343

- Socket size : 212992 bytes
- Overhead per packet : 576 bytes (sk\_buff, skb\_shared\_info)
- PDU size ~ MTU = 1500 bytes
- LSPs/socket :  $\frac{212992}{1500+576} = 102$  LSP -> not much room for PSNP and Hello !
- → Important to advertise a correct RWin to avoid overflowing the socket buffer

# Congestion Control – What is it ?

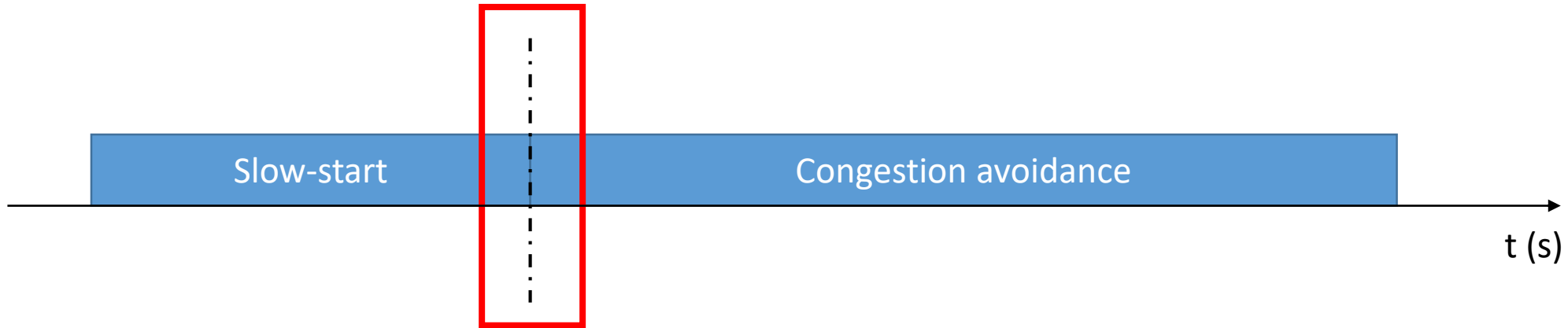


# Congestion Control – What is it ?



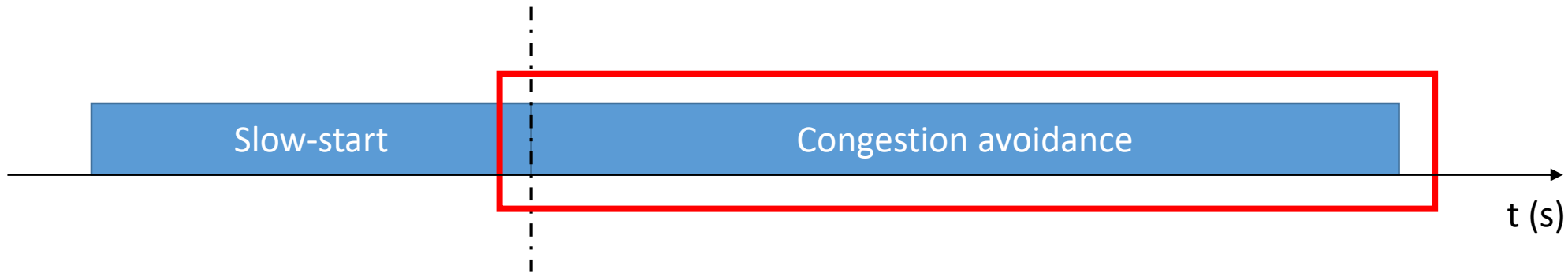
Exponential growth

# Congestion Control – What is it ?



Packet loss  
RTT measurements  
Bandwidth

# Congestion Control – What is it ?



Additive increase multiplicative decrease (AIMD)  
Bandwidth target & control loop