Distributed Flood Reduction: Empirical Measurements draft-ietf-lsr-distoptflood-11 (by now)

Tony P, HPE (by now ;-)

7/25 and now 11/25

Warning of Déjà Vu and “What’s New Since Then”?

- This is a slower replay with more detail of 7/25 IETF Presentation as requested by chairs
- What happened since then if you don’t want to hear it all again
  - -10 better double hashing
  - -11
    - clarifies a subtle detail with the shortest-path removal
    - clarifies that it is fine to full flood if e.g. there isn’t enough CPU to compute/memory to cache
- Lots of testing with CSNPs switched off and link flapping/different flooding speeds
  - Flawless convergence which proves implementation correctness (at least for the tests ;-)
  - No statistically measurable difference in RTX and PSNP volume with or without CSNPs
    - Seems to indicate even in dynamic topology double coverage does a job fast enough to not need any “patching”
    - Things are too dynamic to easily distinguish what was the “path PSNP that caused flooding” and “normal PSNP”

What's the Point?

- Empirical measurements of distributed flood reduction
  - Approximation of large customer networks
  - Stable topology under massive LSP reissue/injection
    - Few vs. many sources injecting
  - Impact of available CPU cores on a node
- Juniper ISIS implementation of current draft
- Questions?
  - Does it converge?
  - How fast does it converge?
  - How much CPU/LSP/SNP volume does it reduce?
  - How much CPU does it burn?
  - How much difference does faster flooding make?
  - How much does it RTX?

Testing Methodology

- Geometric graphs with lots redistributed prefixes
  - Properties very similar to modern backbones
- 48 physical threads at 4.2G with lots CPU and cRPD topology over k8s
  - CPUs limited to 1 or 2 cores per container (more than that is unrealistic)
- Measuring with fast (250pkts/sec) flooding and standard flooding (30 pkts/sec)
- Measurement after forcing LSP regeneration on 2 – 18 sources
  - LSDB sync delay after regeneration
  - CPU load distribution
  - TX load distribution
  - RX load distribution
  - RTX% distribution
  - Drop distribution
  - Reduction% distribution
  - Flooding pressure, i.e. reflooders distribution
- No logging, no SPFs, no flaps during measurements enforced
- Special thanks to the https://www.kathara.org folks for brilliant open source docker tooling

Spoilers Before Lots of Cool Graphs

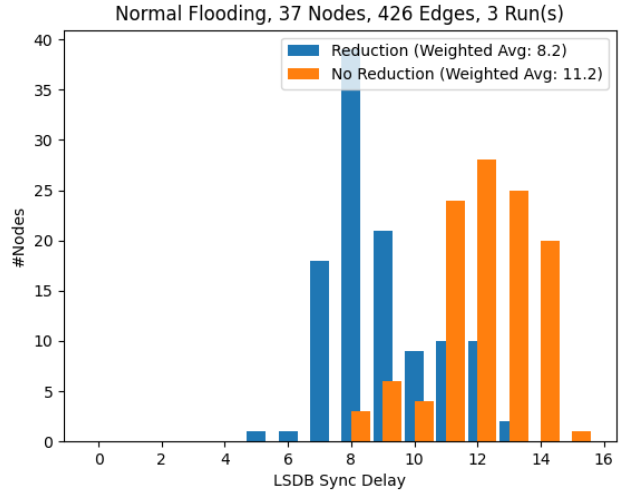
- Massive CPU saving with reduction in all scenarios (~60+%)
- Good LSP TX/RX Reduction (~25-70%)
- Decent to good LSDB Synchronization Speedup (~0-30%)
  - 2 cores vs. 1 neglectable differences
  - Difference larger with faster flooding and more sources/LSPs

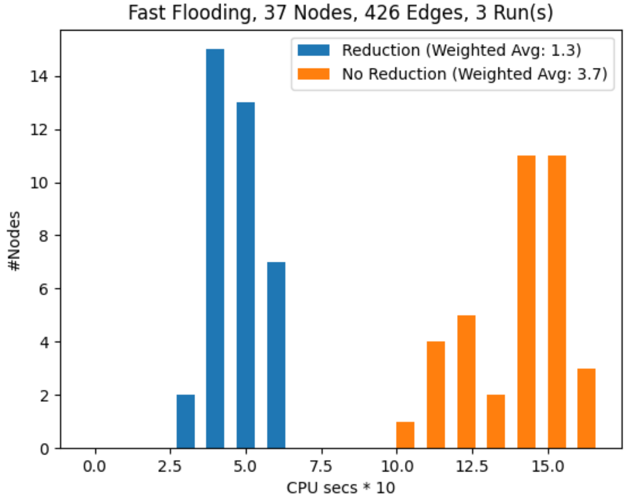
Warning

- Do NOT try at home unless you have either
  - a watercooled 32 cores Threadripper with 256G physical
  - equivalent Xeon
  - whole Docker swarm

Graph Smorgasbord

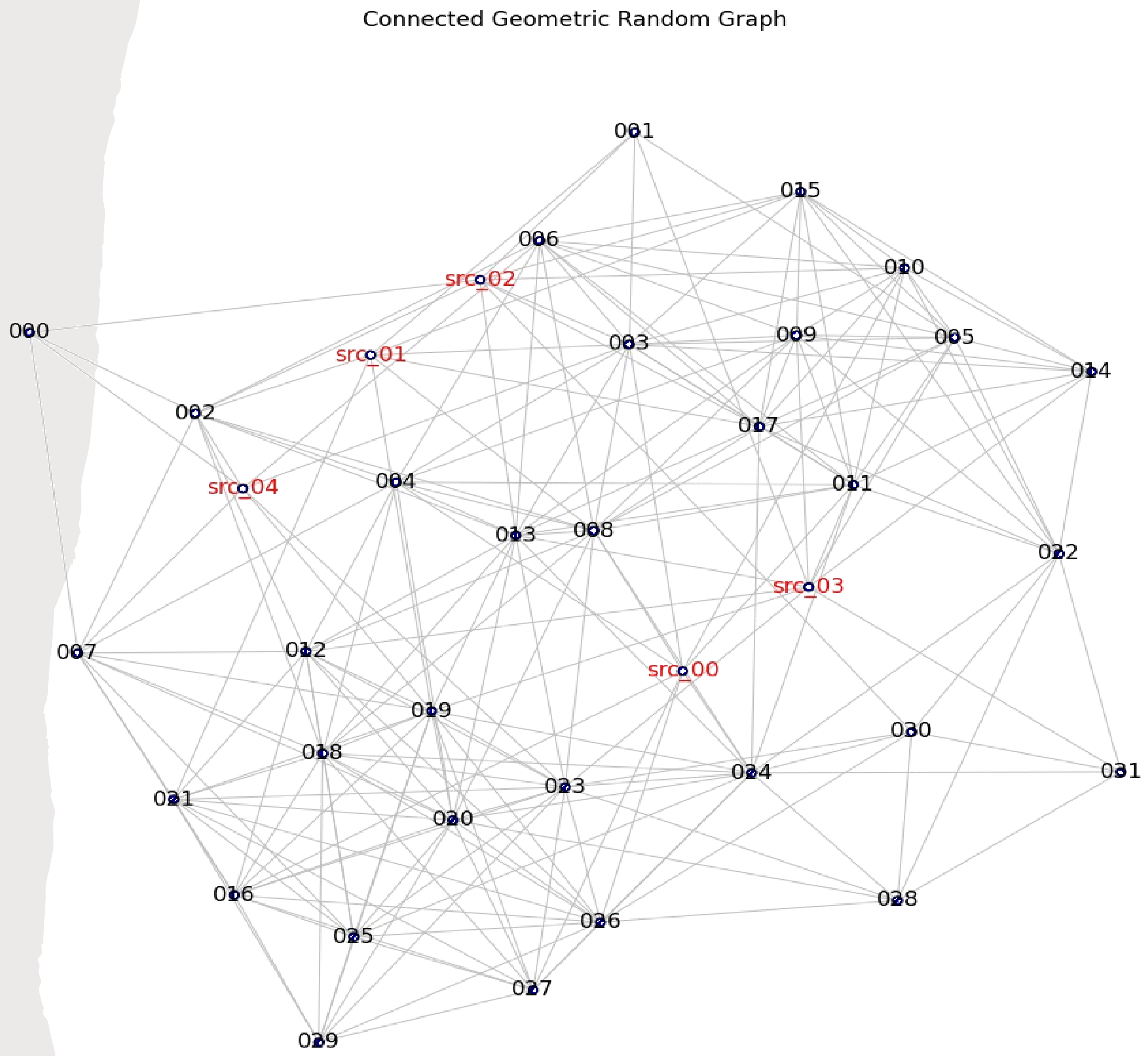
- How to Read
  - Reduced (Blue) and Classic (Orange) bars
  - Distributions of something vs. #Nodes with the value
  - Maybe %, maybe normalized over multiplication or division
  - Weighted average rough indication of difference in behavior
  - Examples
    - Normal flooding LSDB sync delay
    - Fast flooding CPU consumption





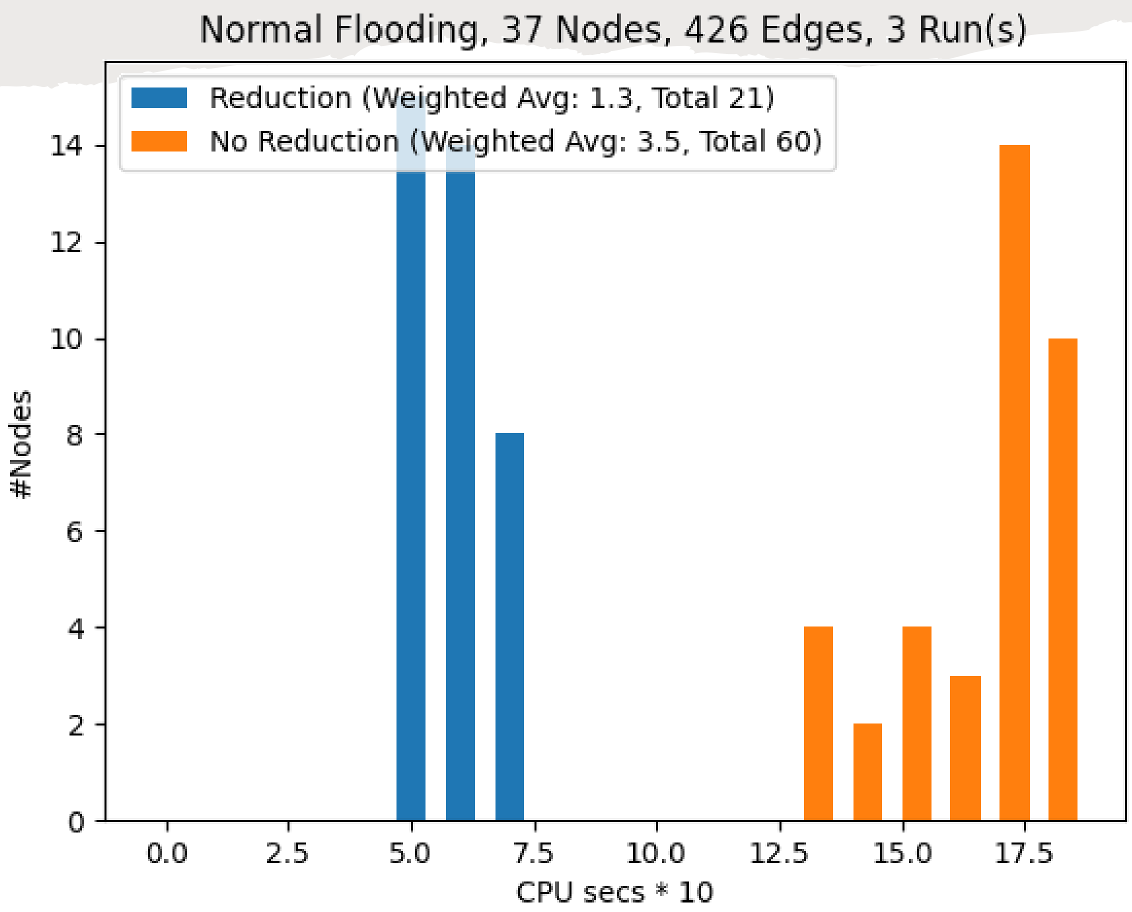
First Graph

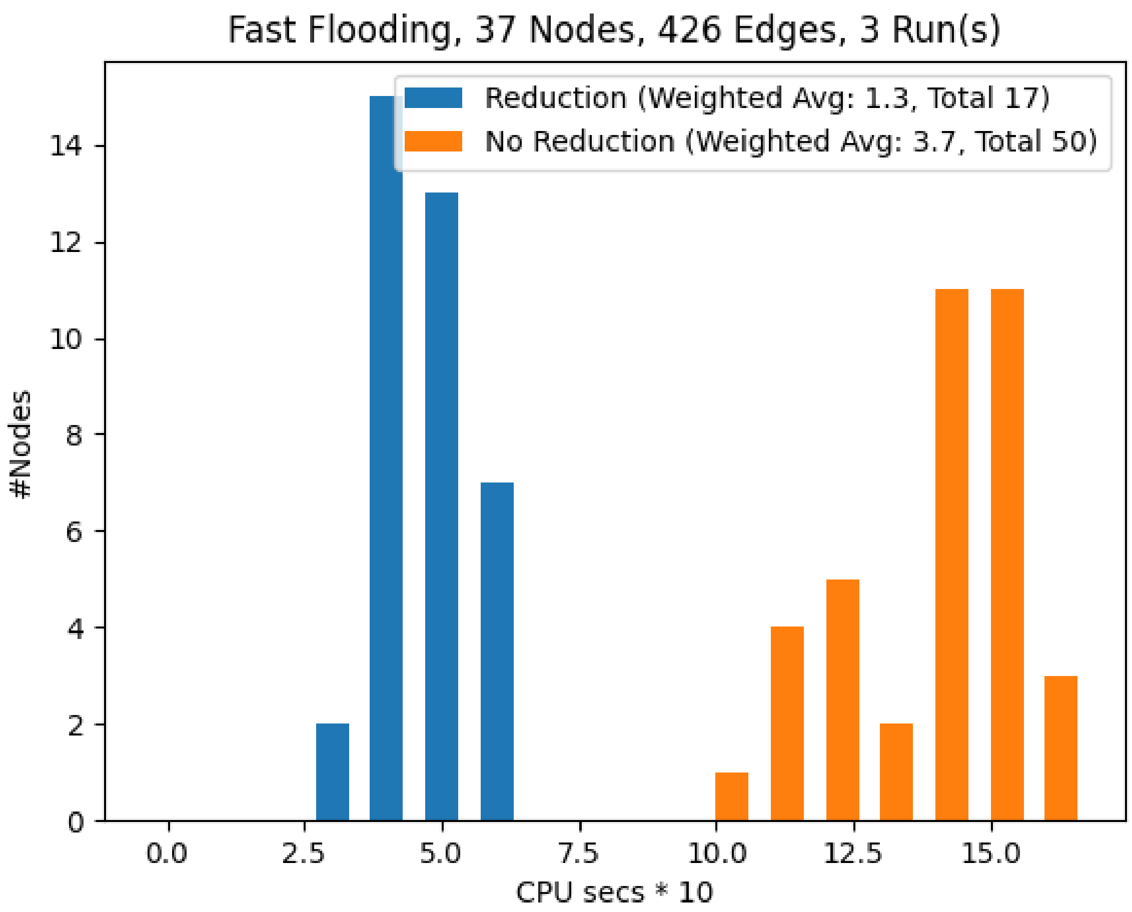
- 1K LSPs
- 37 nodes
- 5 sources
- 426 adjacencies



CPU Consumption (1 Core)

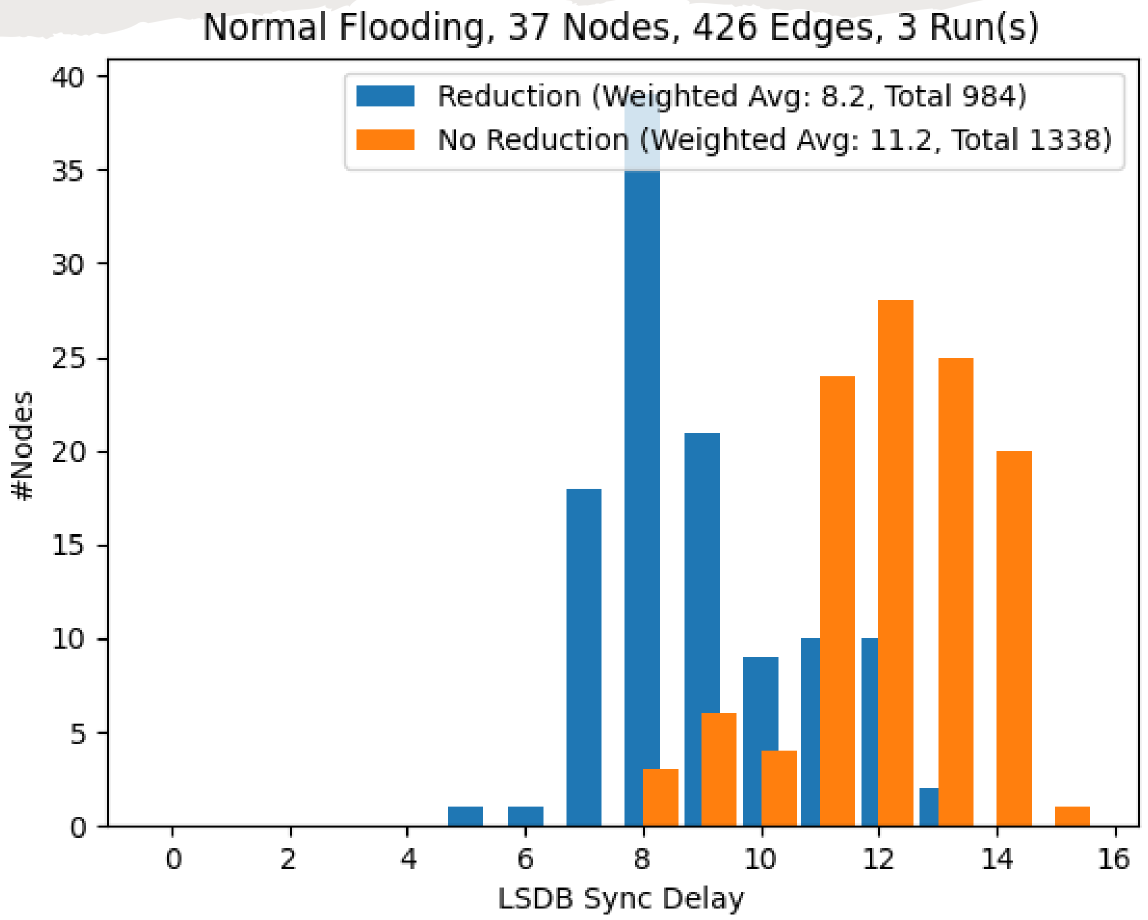
Reduction: 60%
Fast Flooding: another 20%

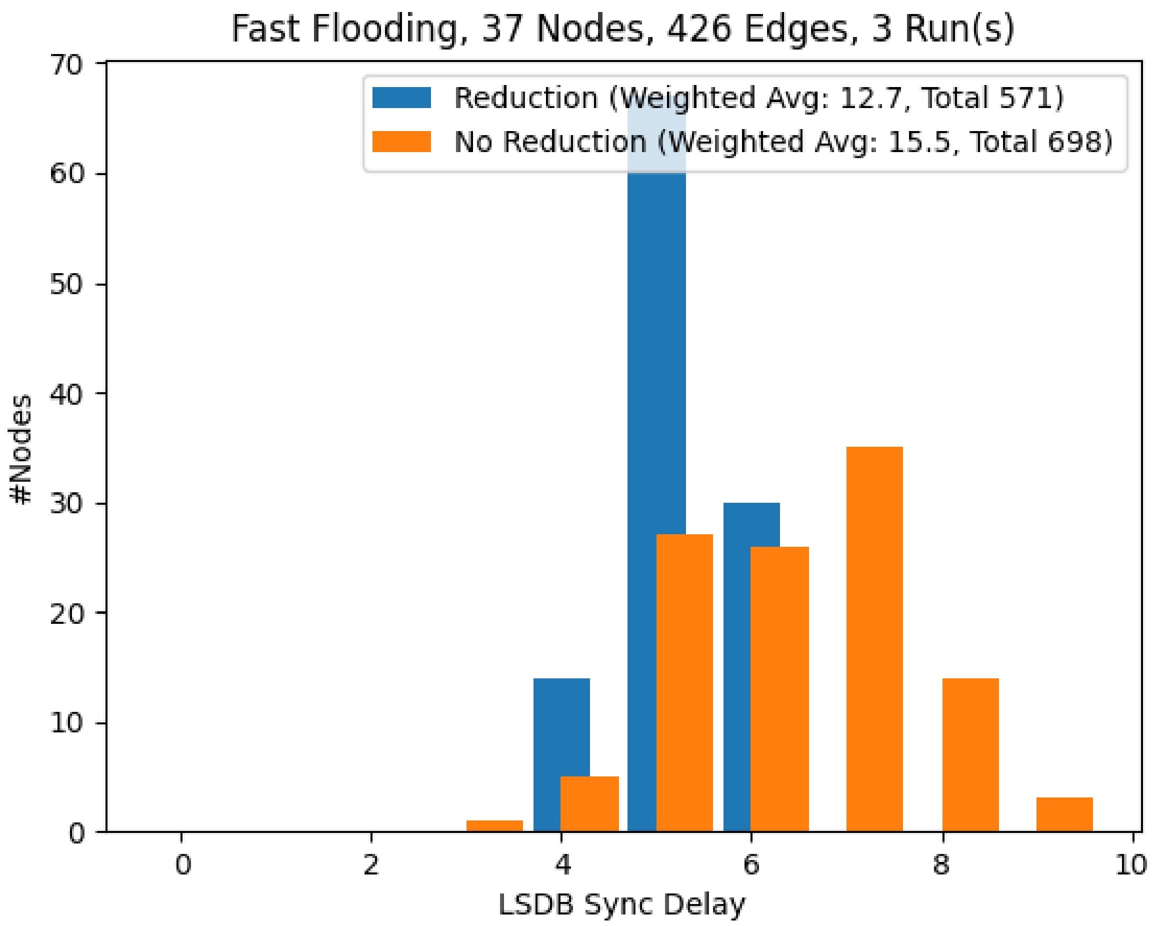




LSDB Sync Delay

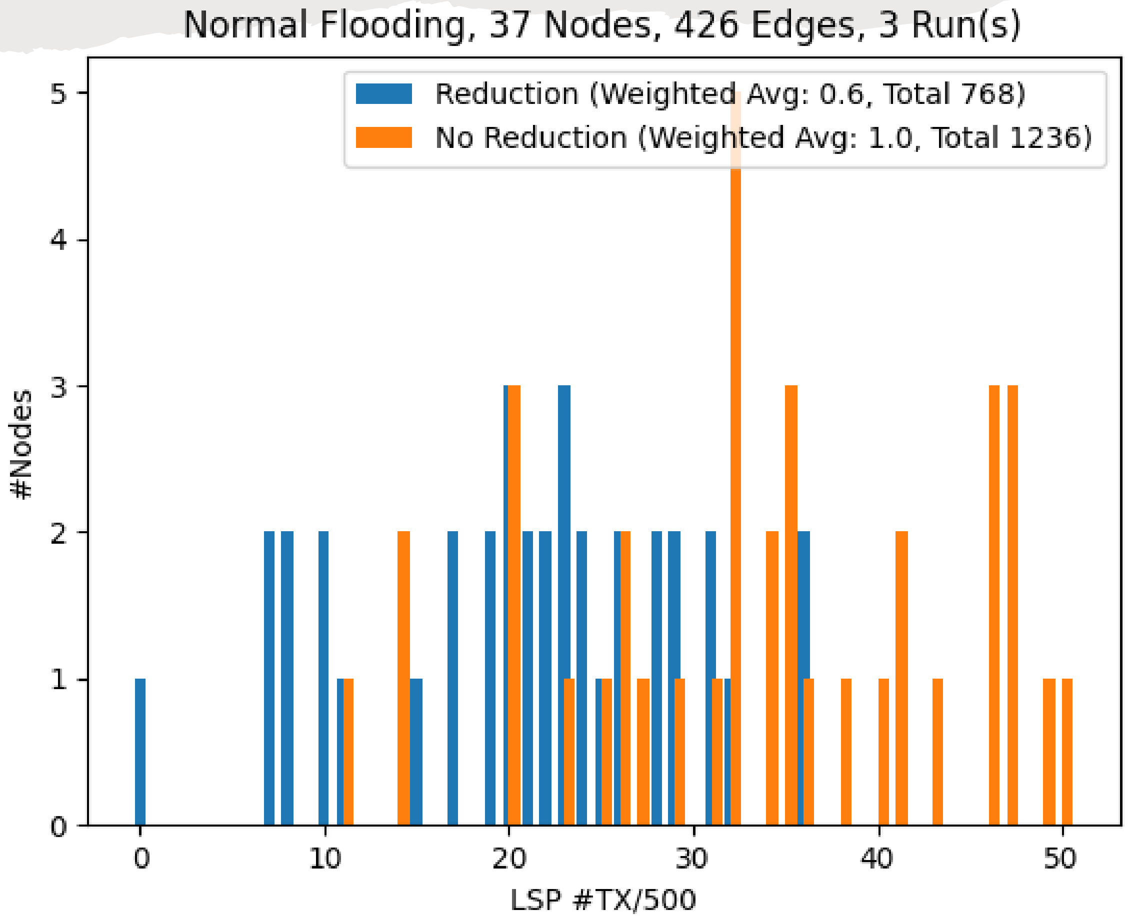
Reduction: 40% Faster
Fast Flooding: ~2x Faster

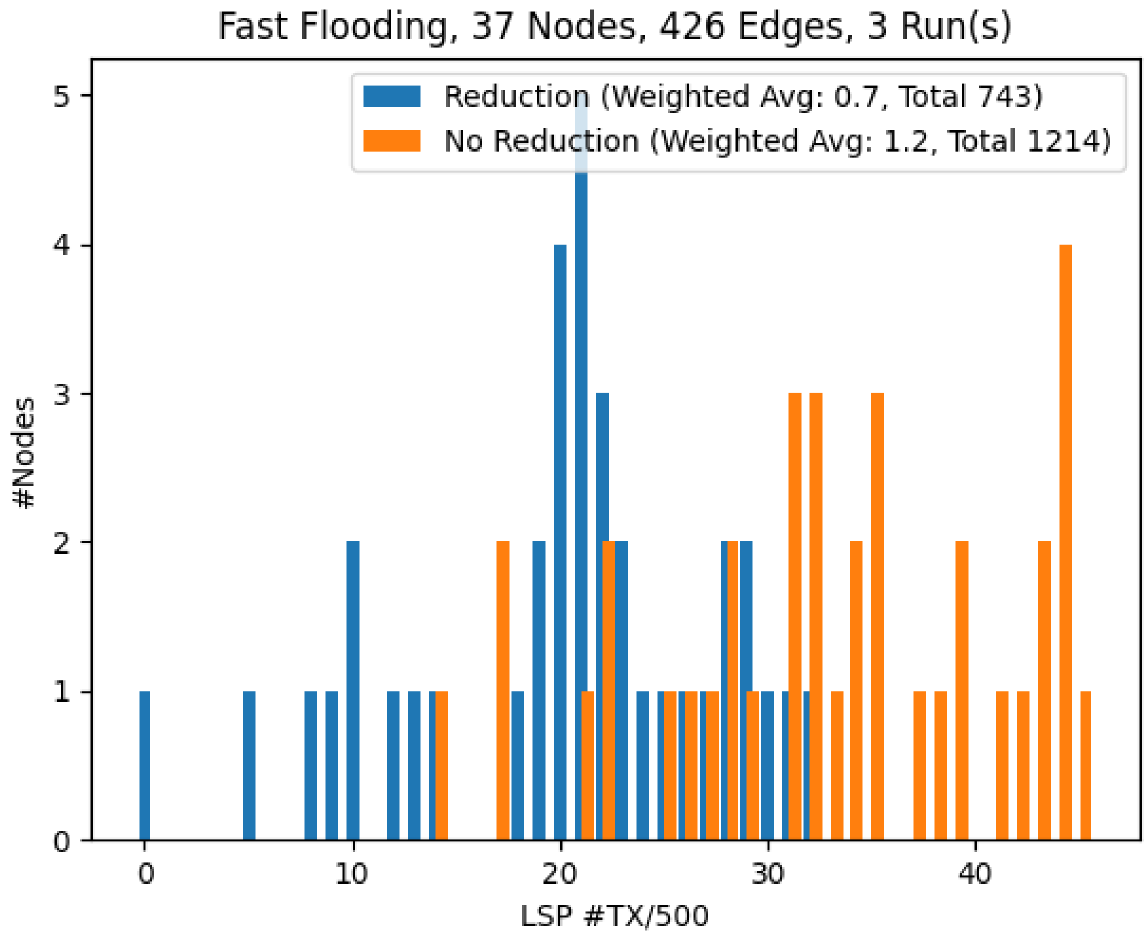




TX LSP (divided by 500)

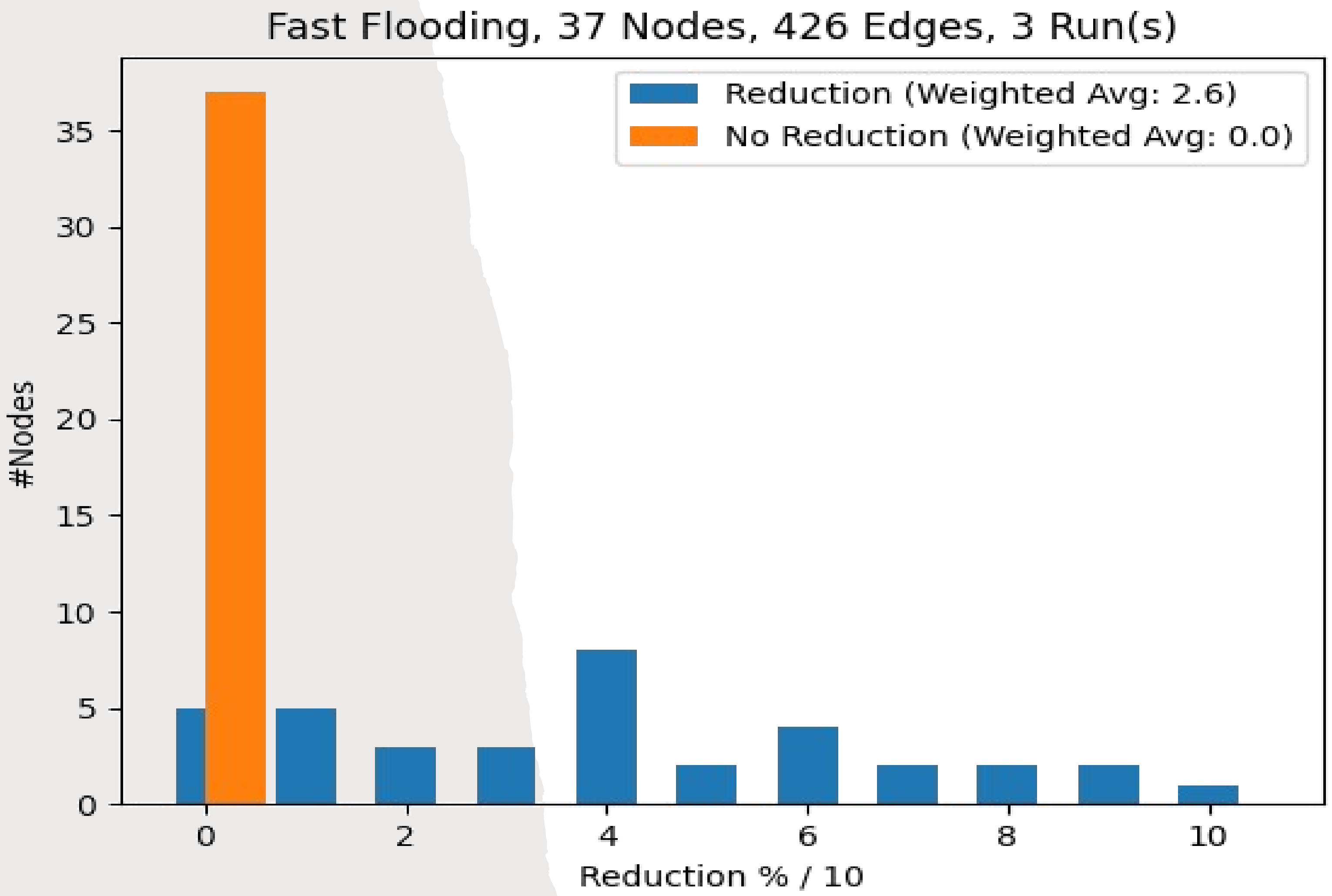
Reduction: 40%
Reduction: Much lower max. volume on nodes





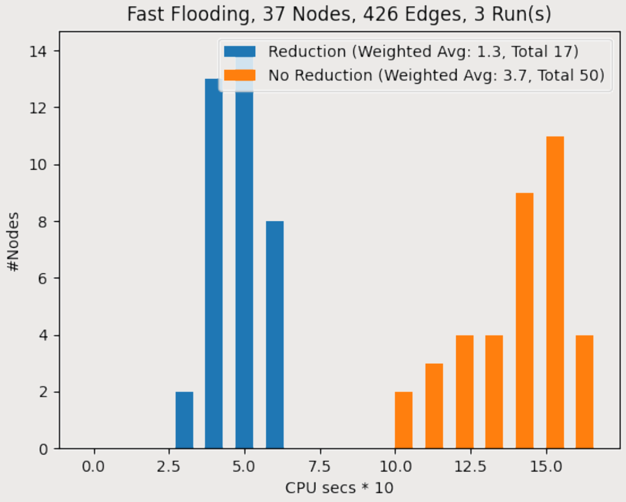
Reduction Distribution

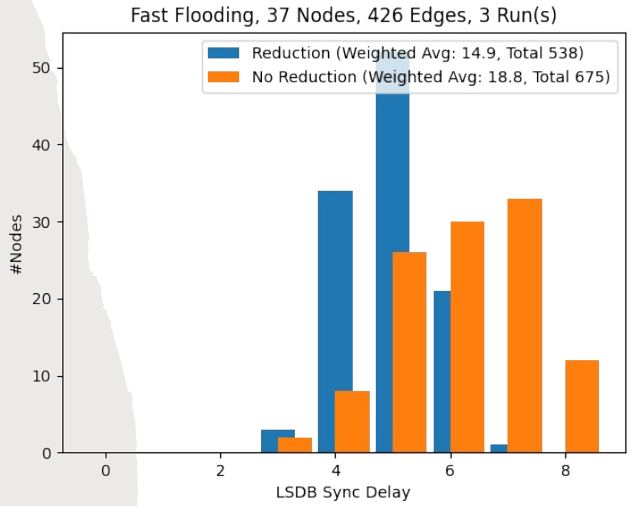
Reduction: Uneven, i.e. about 50% need to reduce to basically achieve the effects

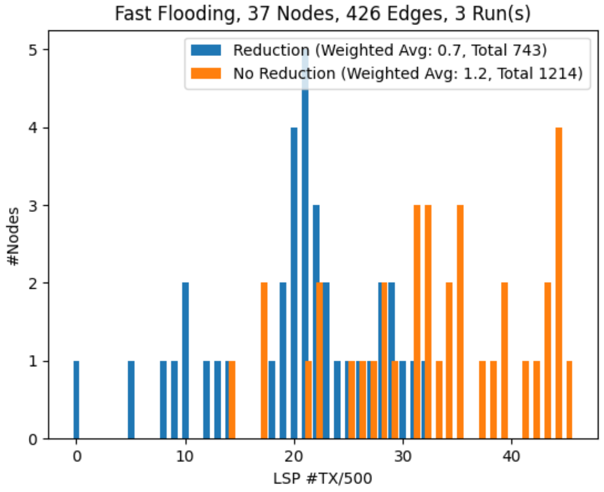


2vs1 Cores: No Help

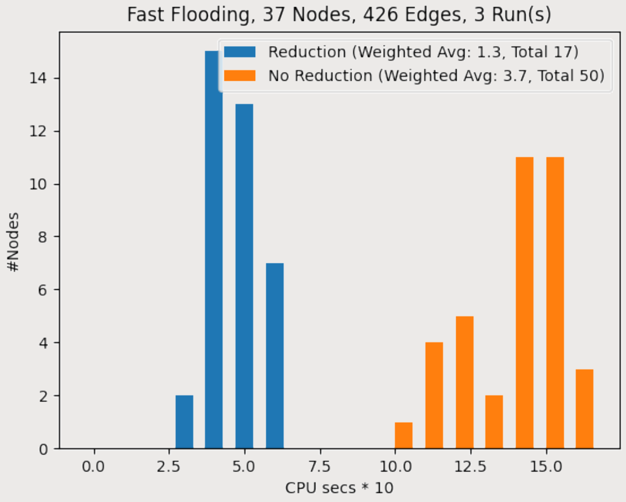
2

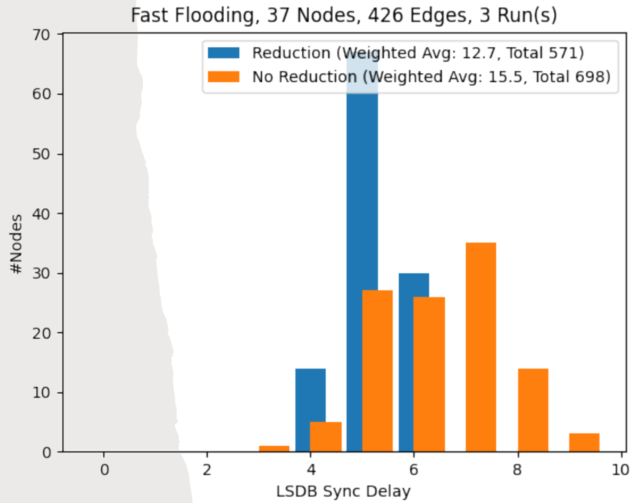


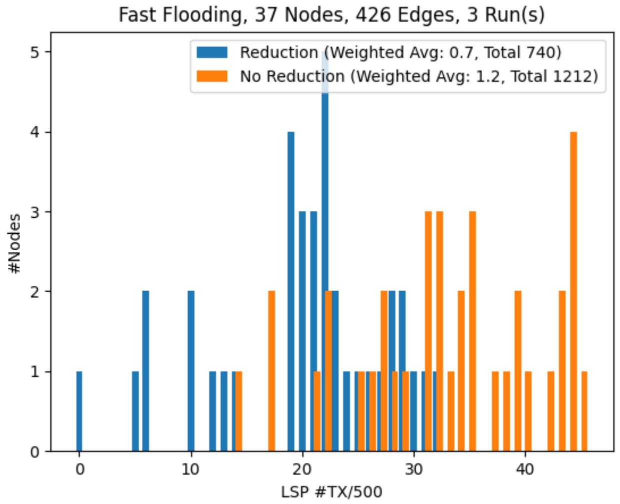




1

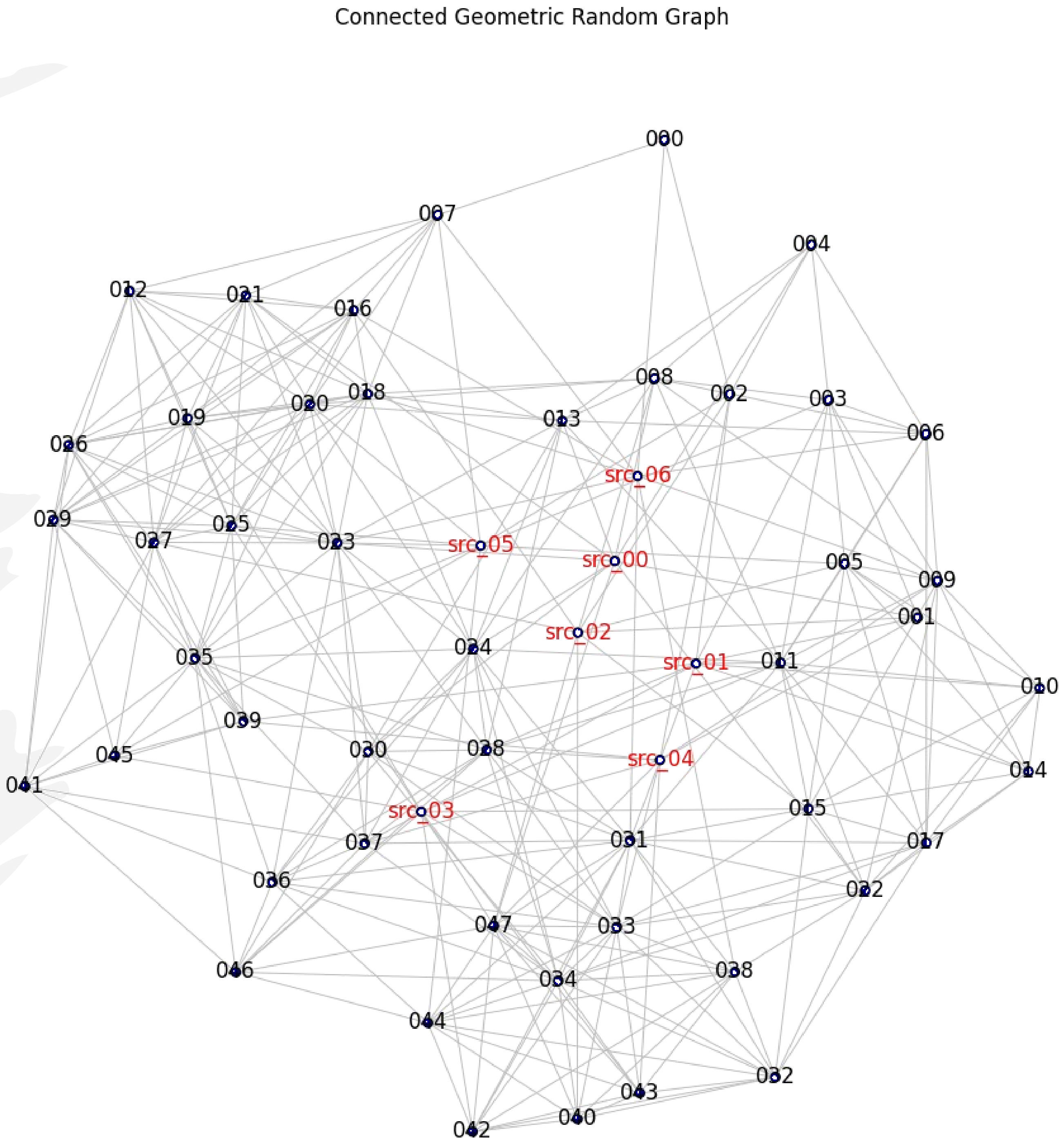






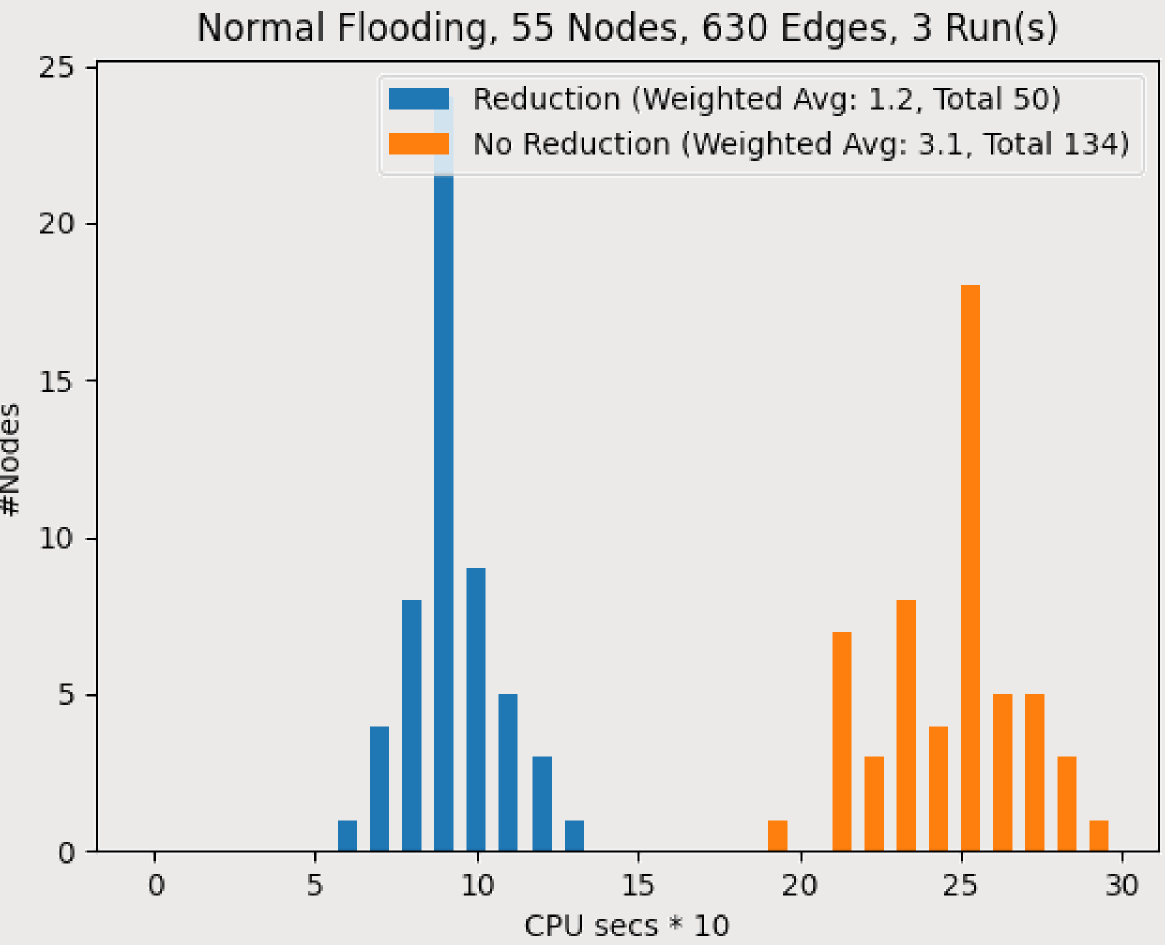
Bigger Graph

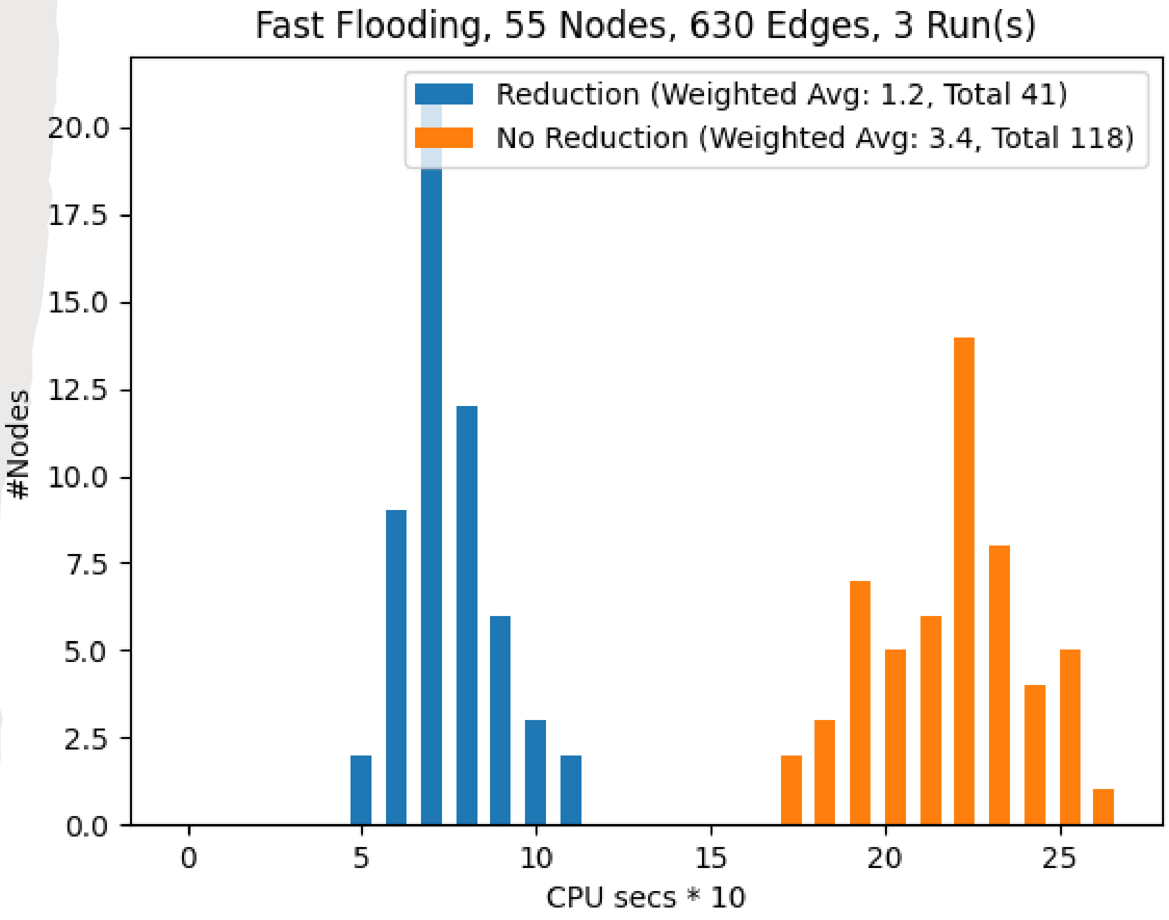
- 1.4K LSPs
- 55 nodes
- 7 sources
- 630 adjacencies



CPU Consumption

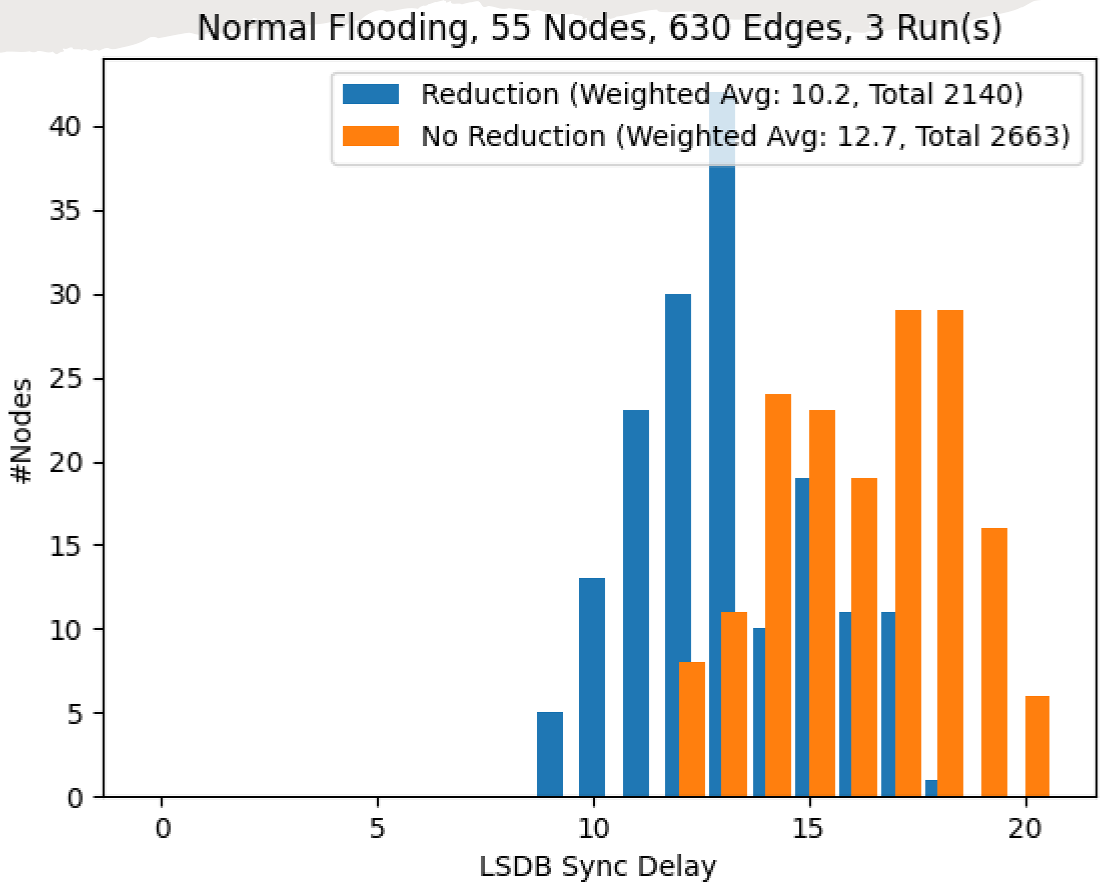
- Reduction: 60%
- Fast Flooding: another 20%

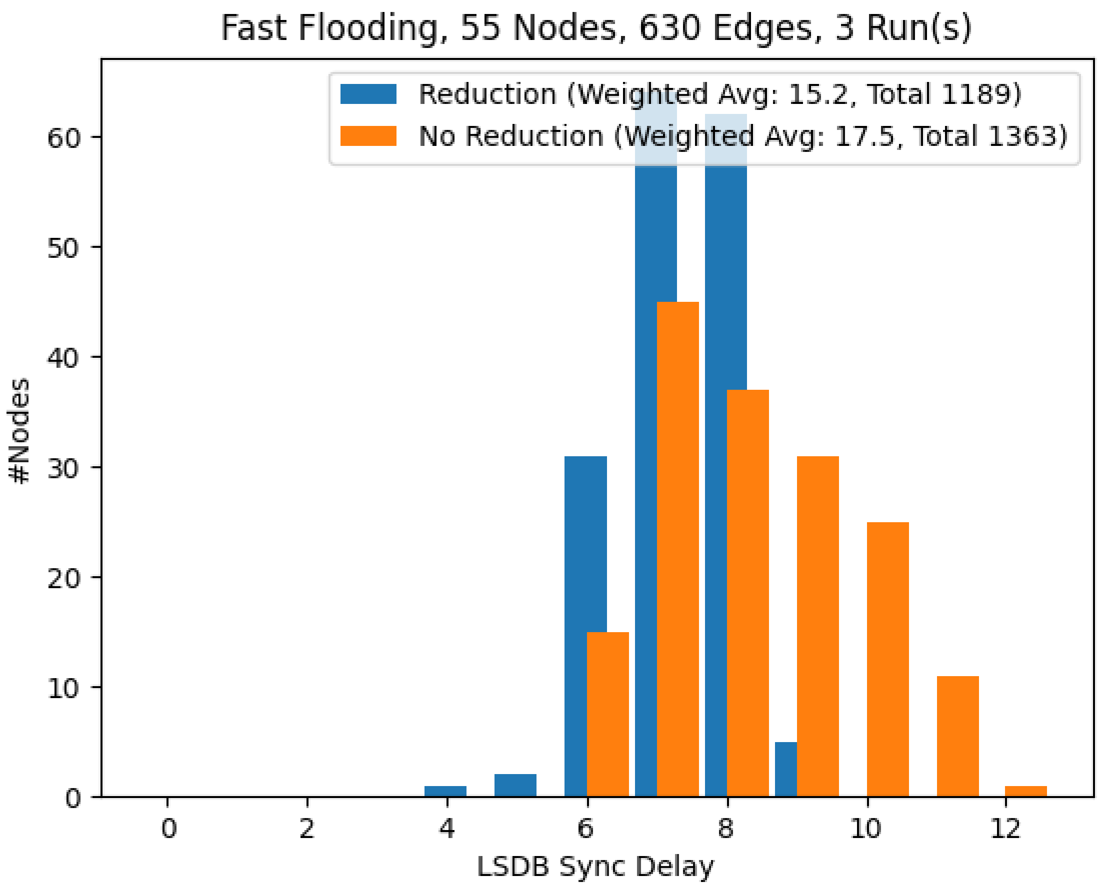




LSDB Sync Delay

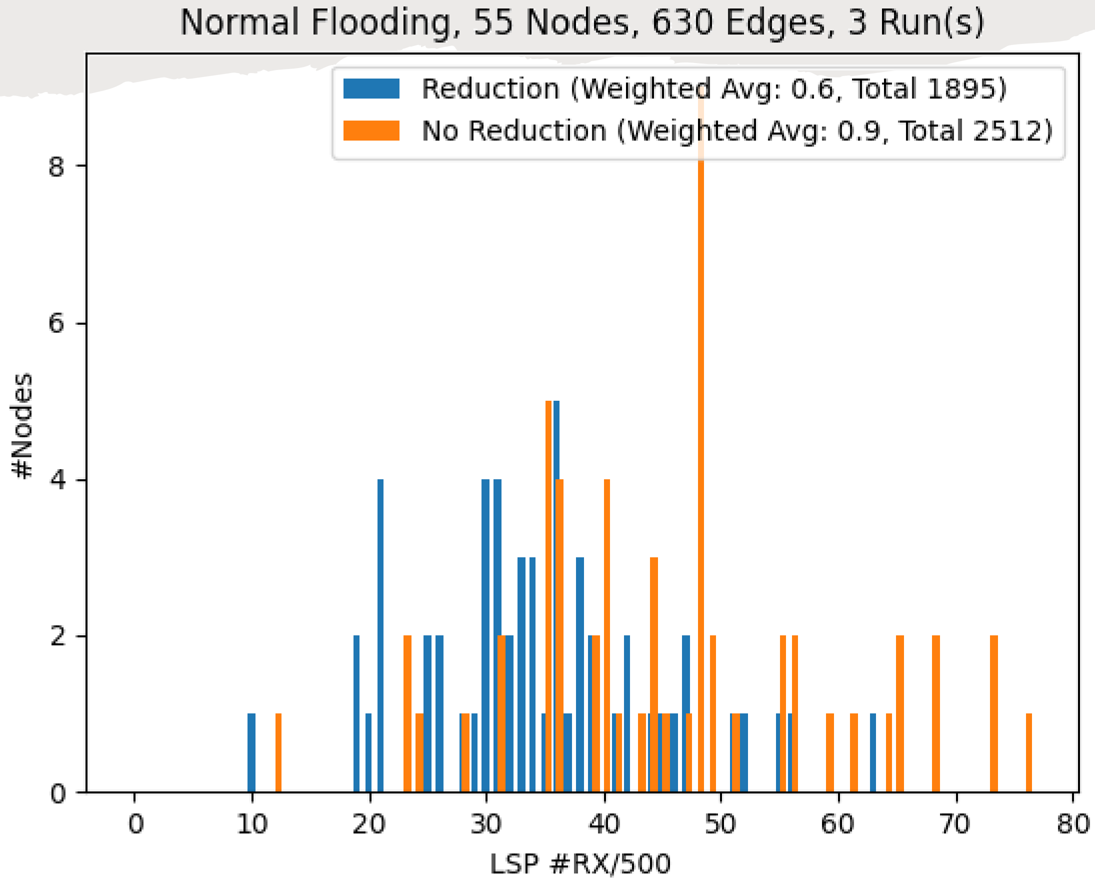
Reduction: 30% Faster
Fast Flooding: ~2x Faster

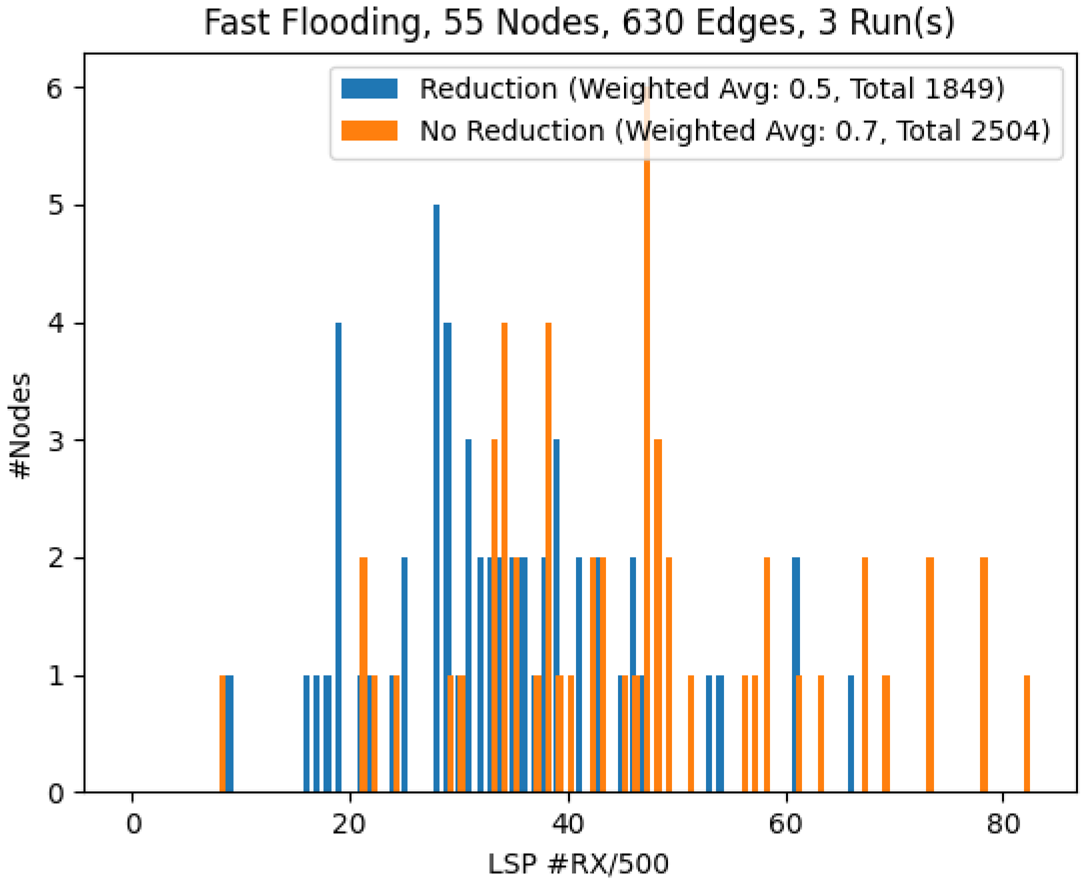




RX LSP (divided by 500)

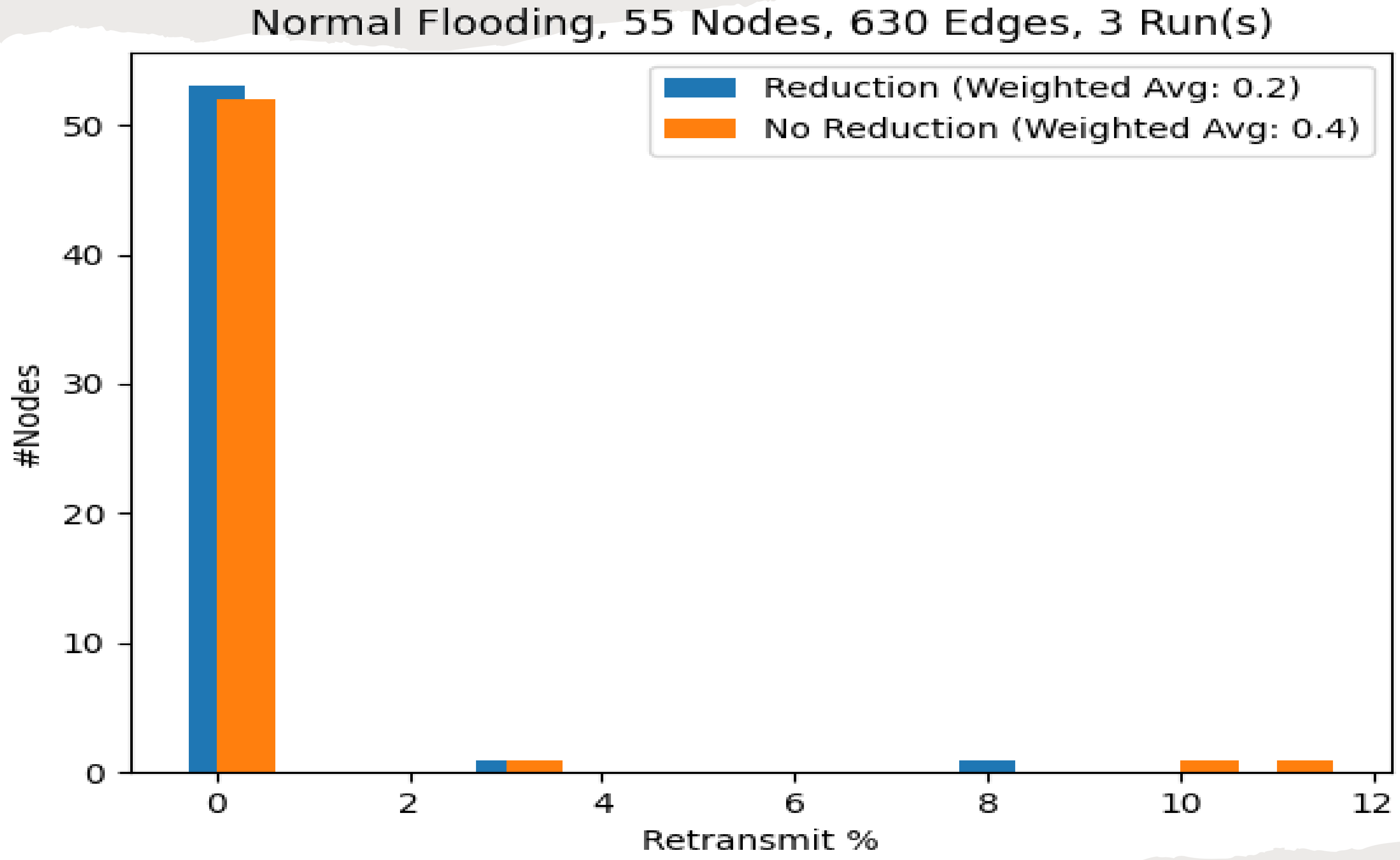
Reduction: 20%
Reduction: Much lower max. volume on nodes





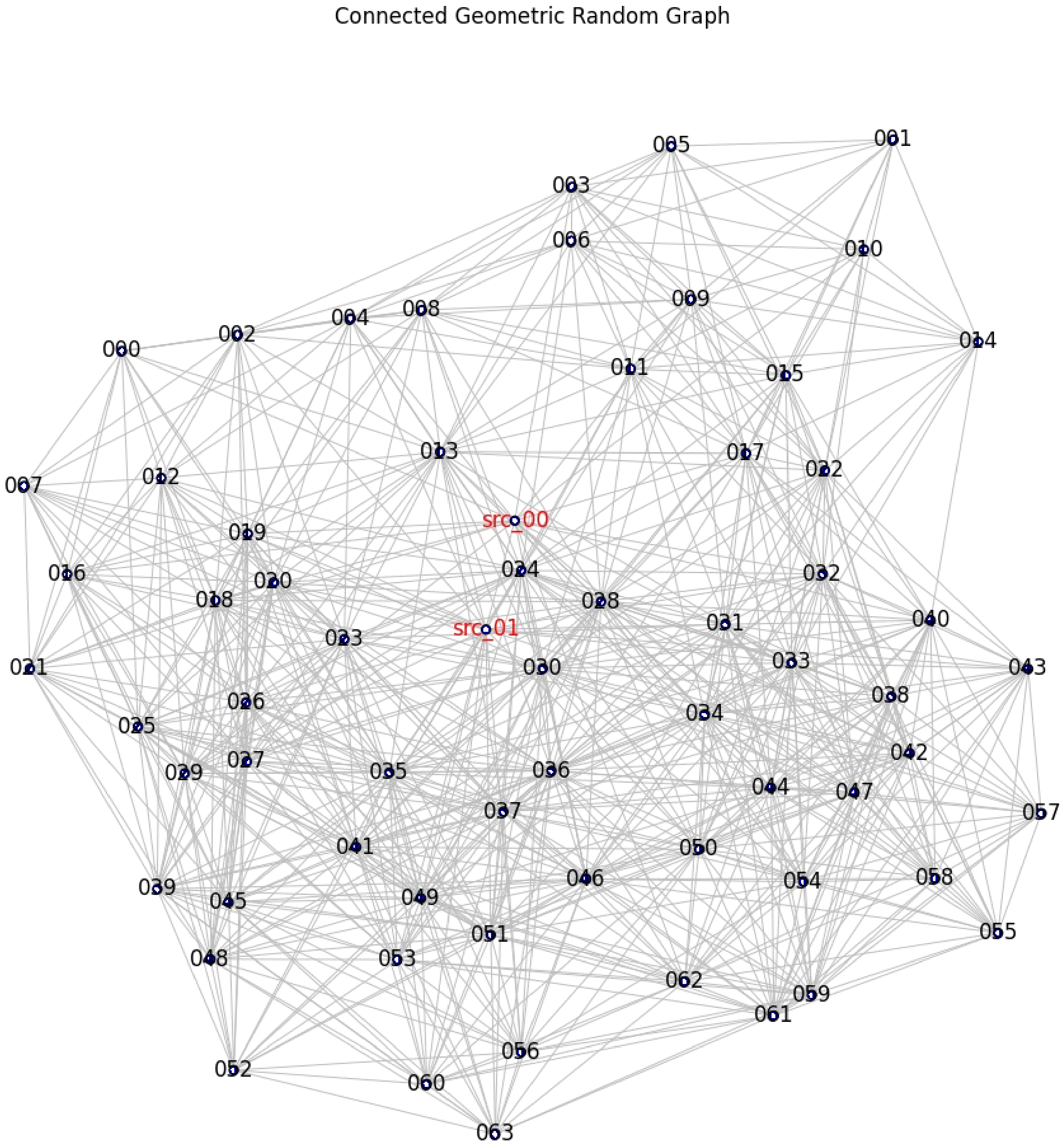
And Now RTX%

Reduction: 50% RTX only
Reduction: Max. RTX on a node 50% less



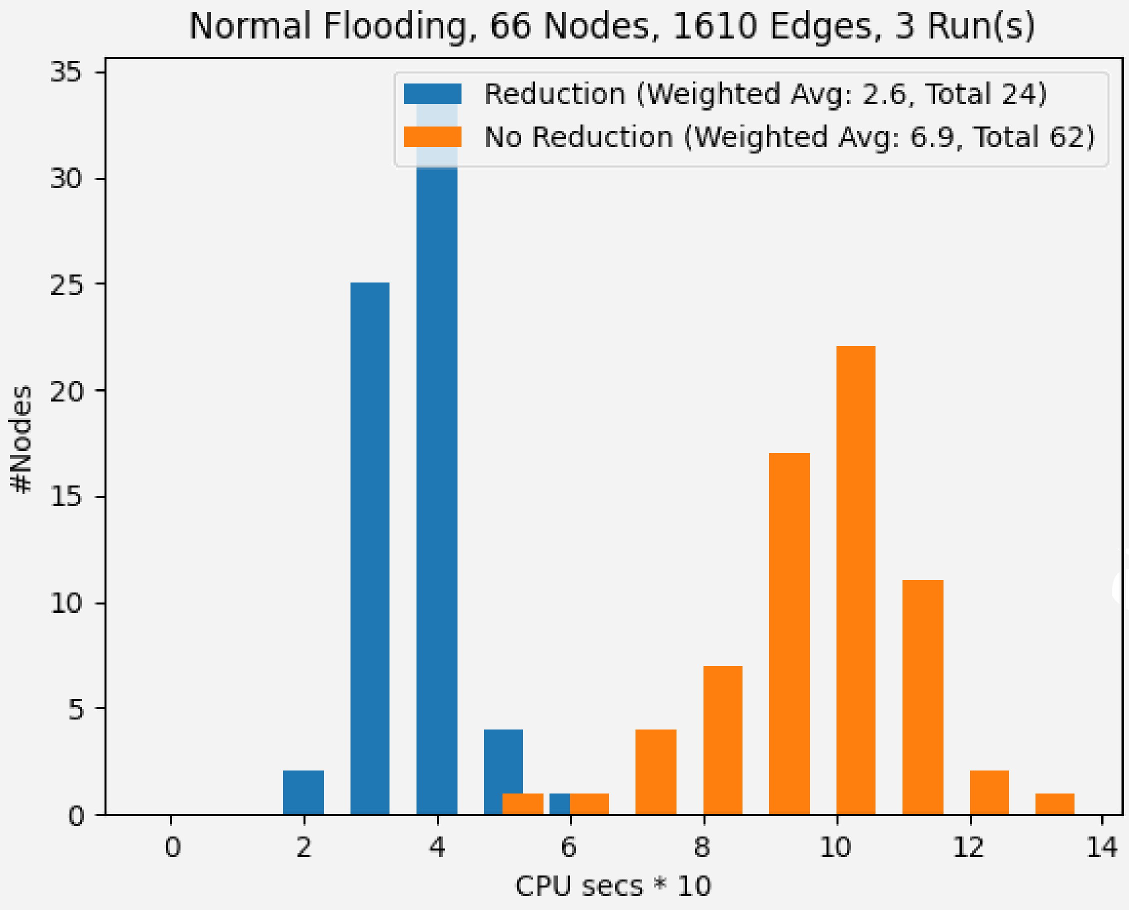
More Nodes
More Adjacencies
Less Sources & LSPs

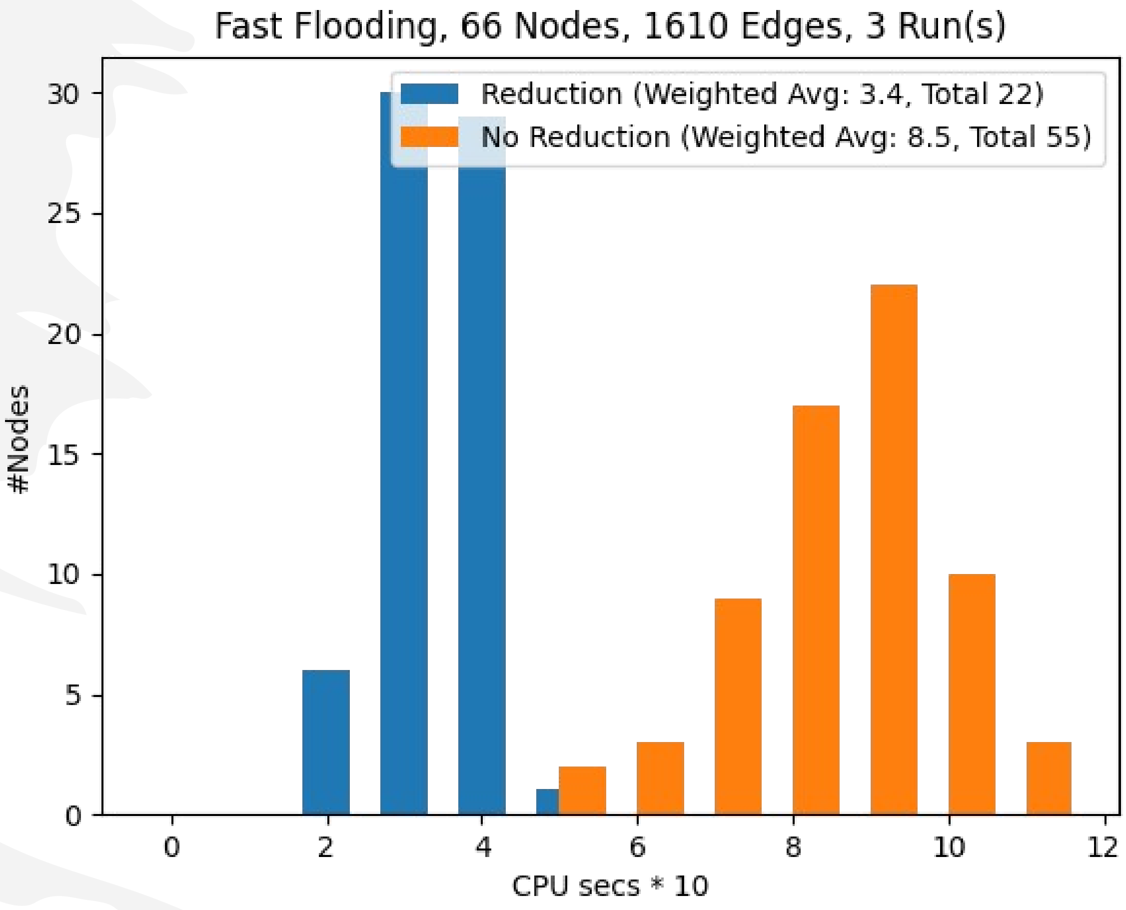
- 0.5K LSPs
- 66 nodes
- Very few (2) sources
- 805 adjacencies
- A bit denser



CPU Consumption

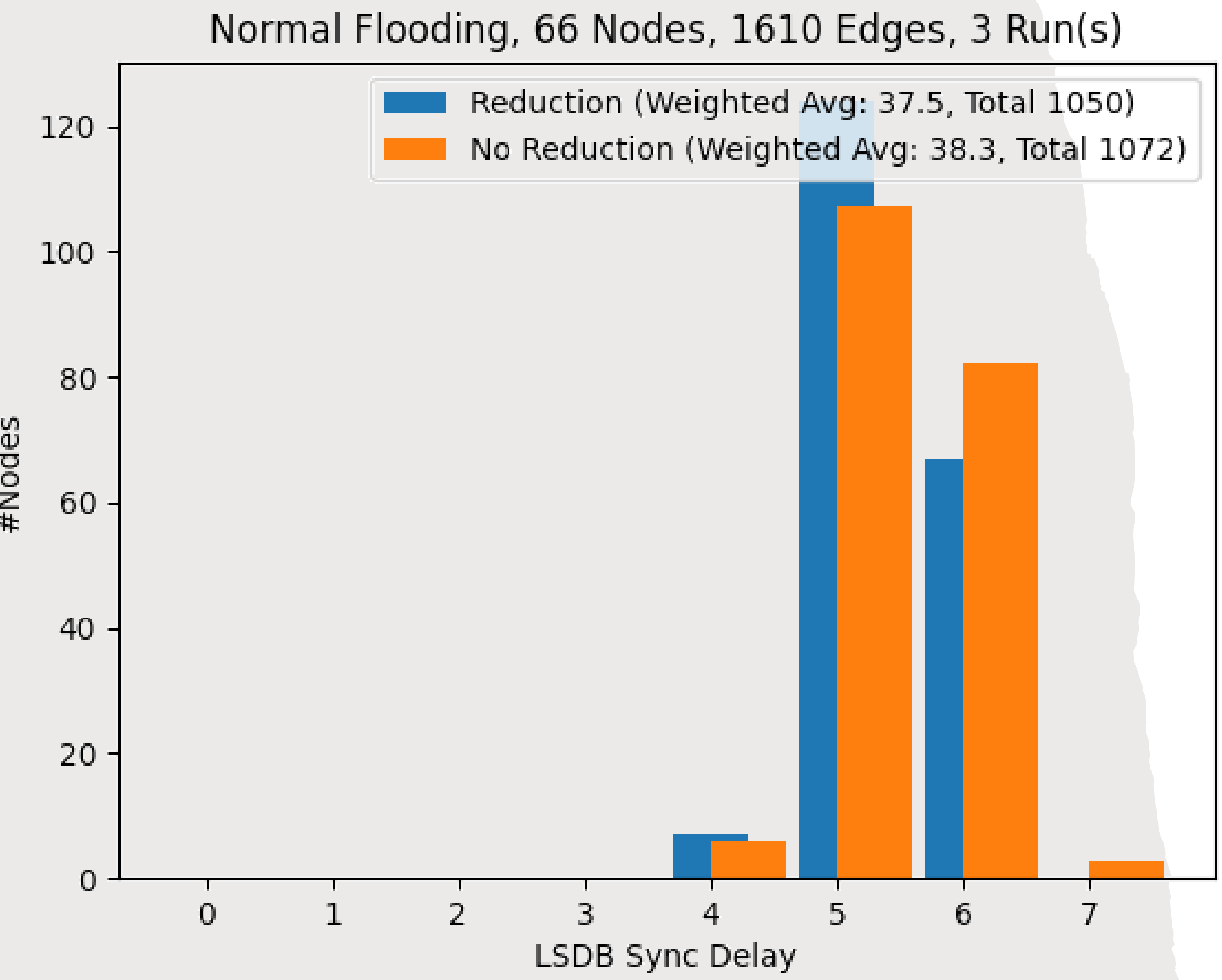
- Reduction: 60%
- Fast Flooding: Same

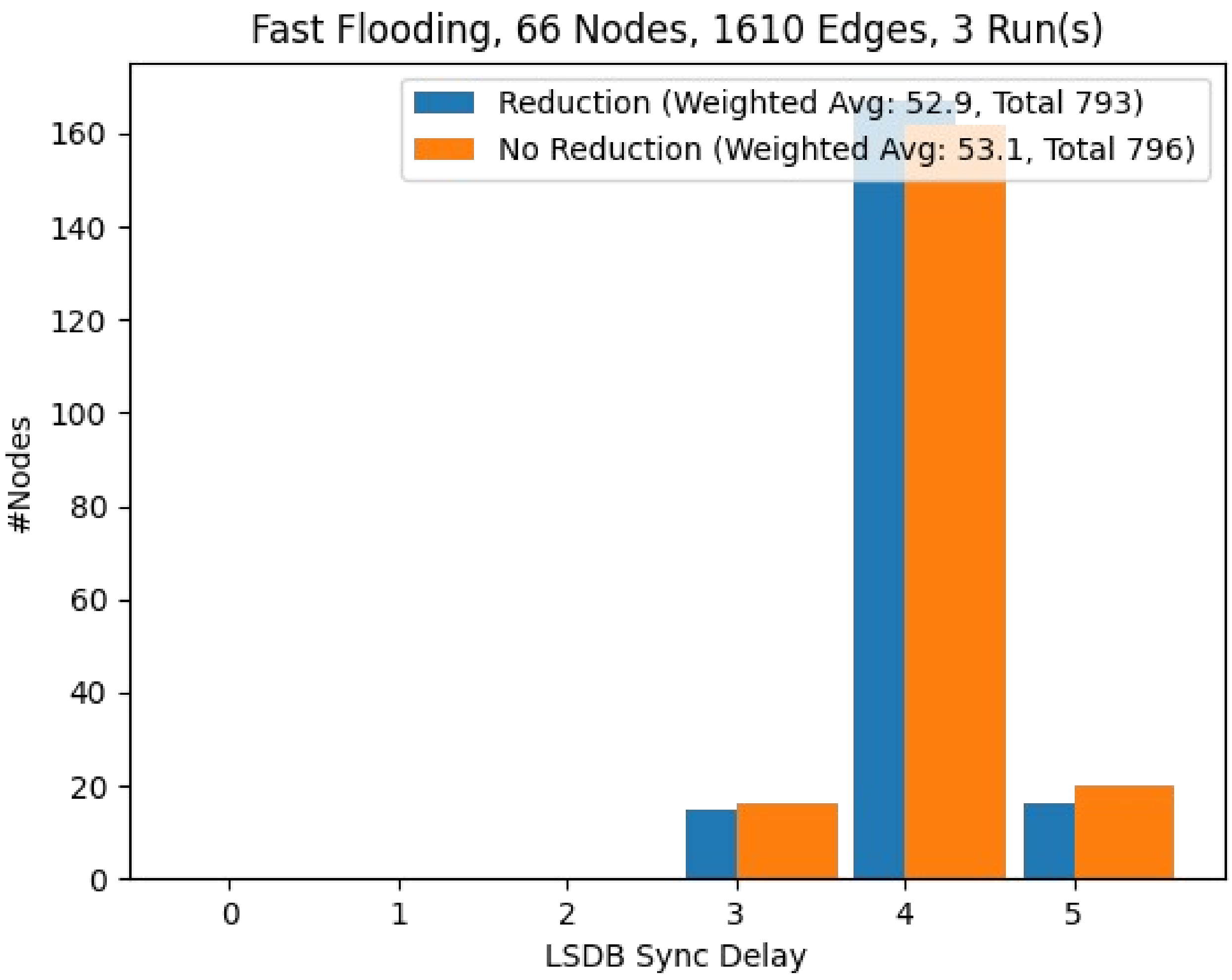




LSDB Sync Delay

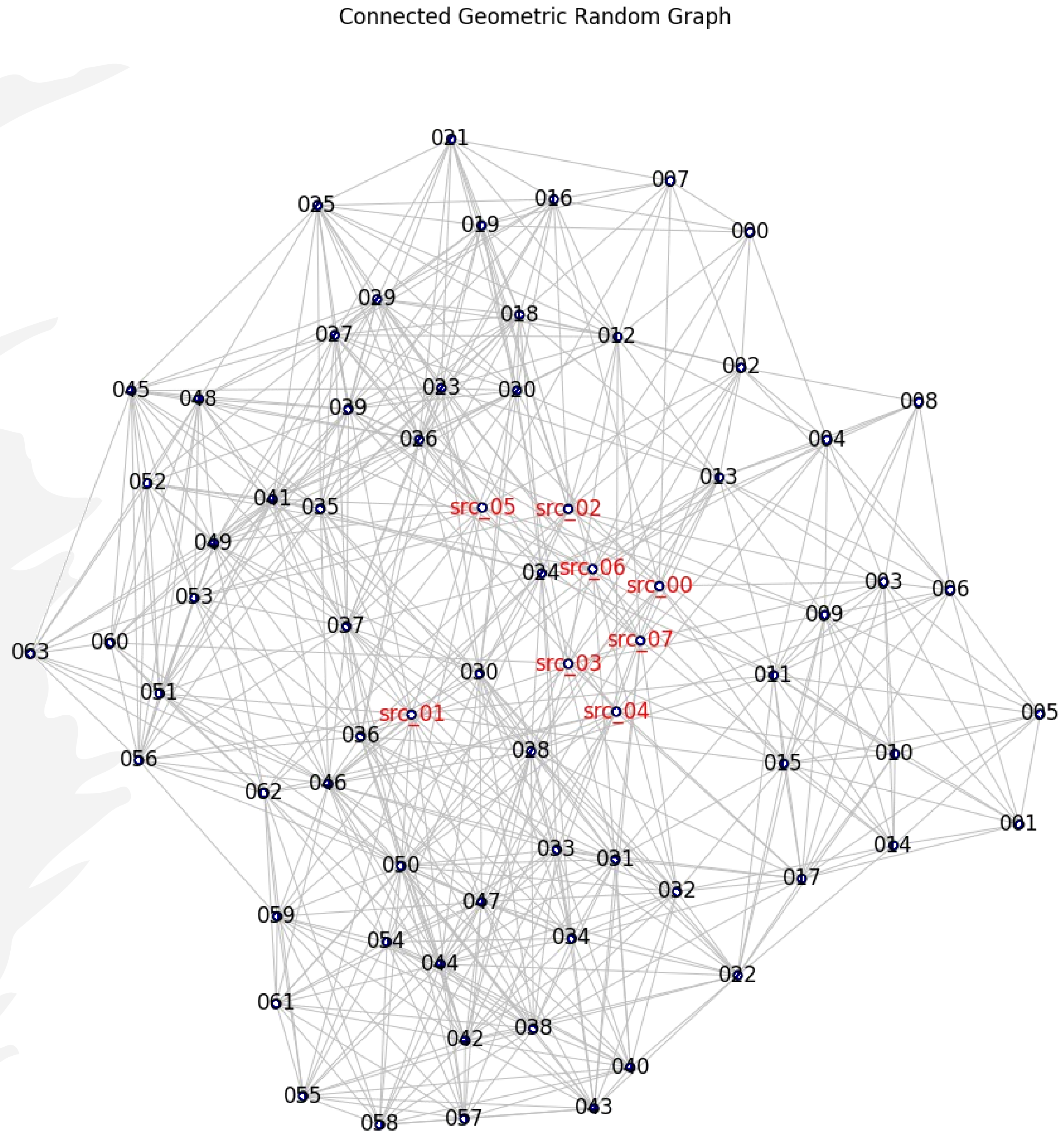
Reduction: No Gain
Fast Flooding: 30% Speedup





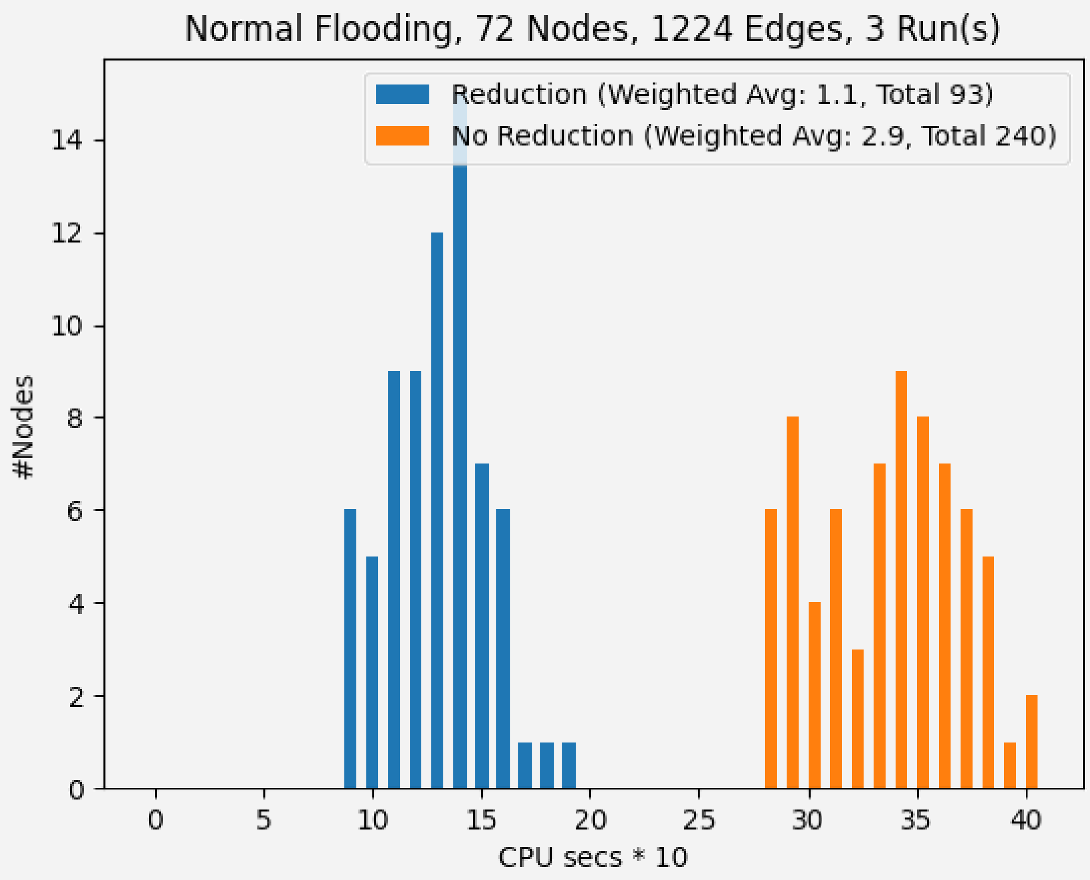
Even Bigger Graph

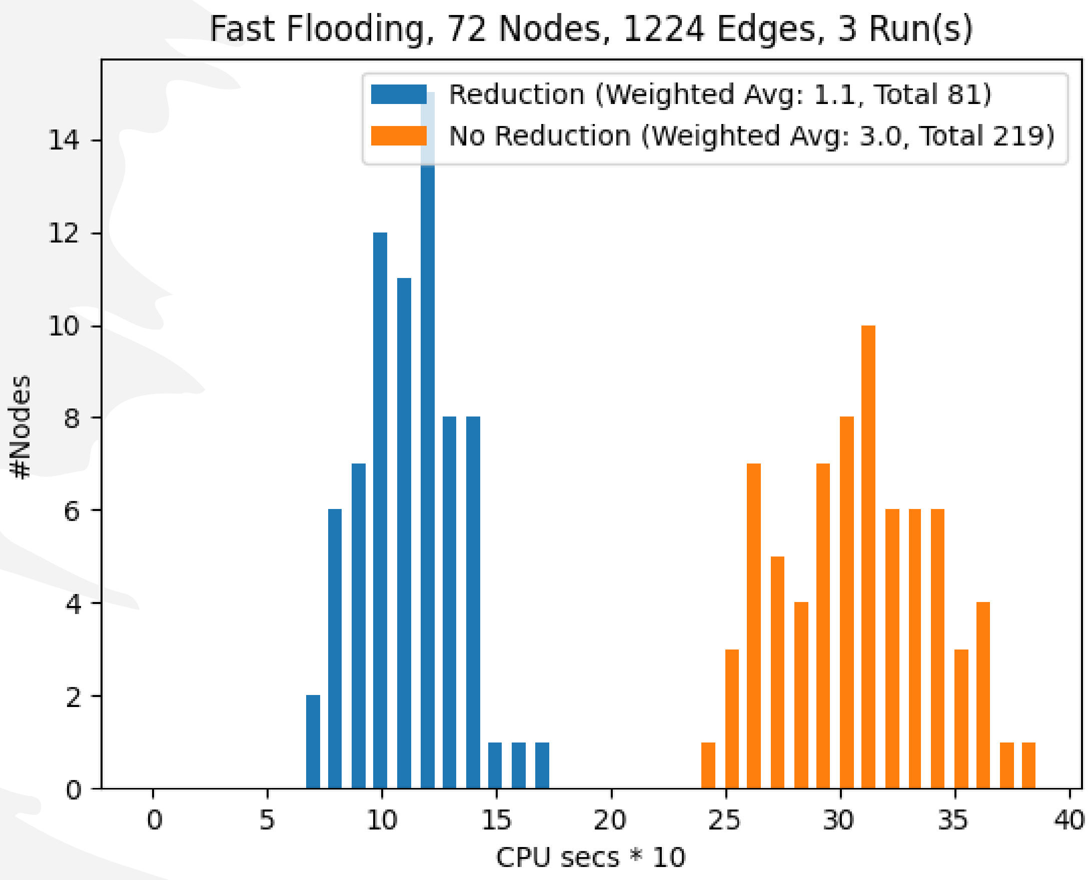
- 1.6K LSPs
- 72 nodes
- 8 sources
- Less dense with 612 adjacencies



CPU Consumption

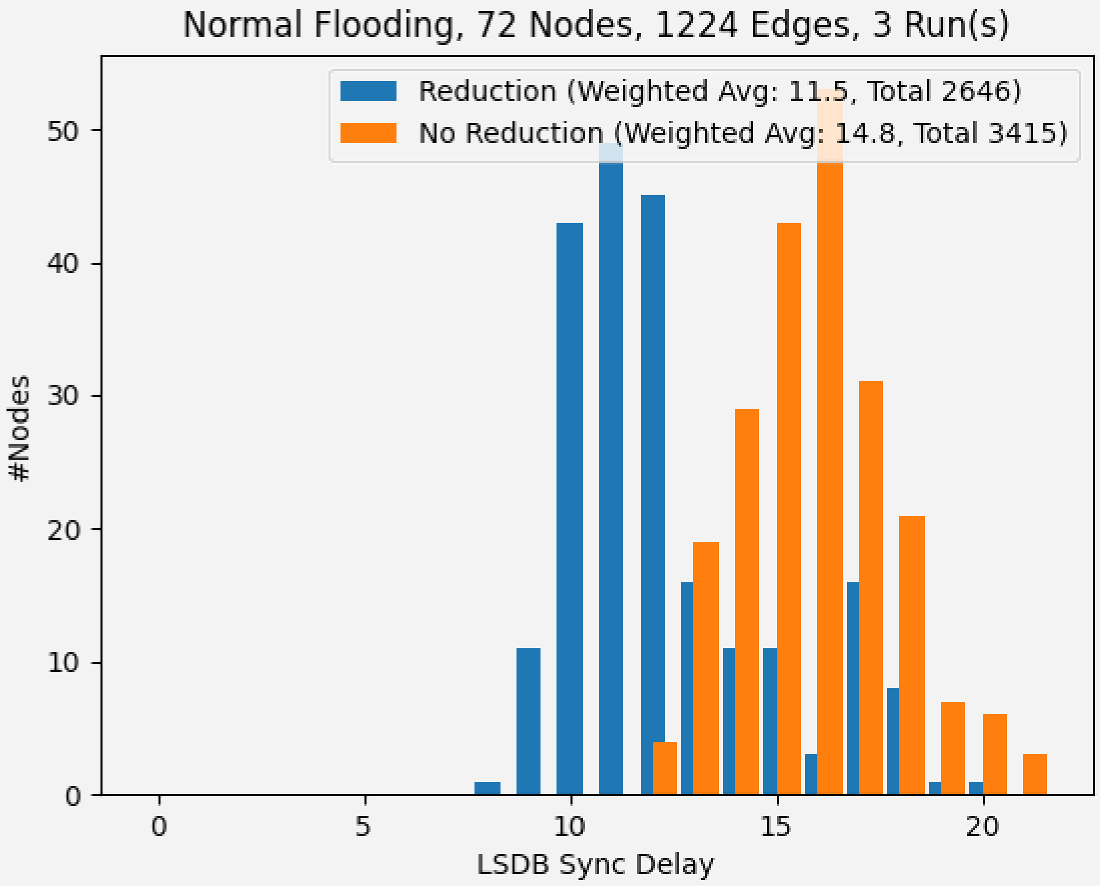
- Reduction: 60%
- Fast Flooding: another 10% less

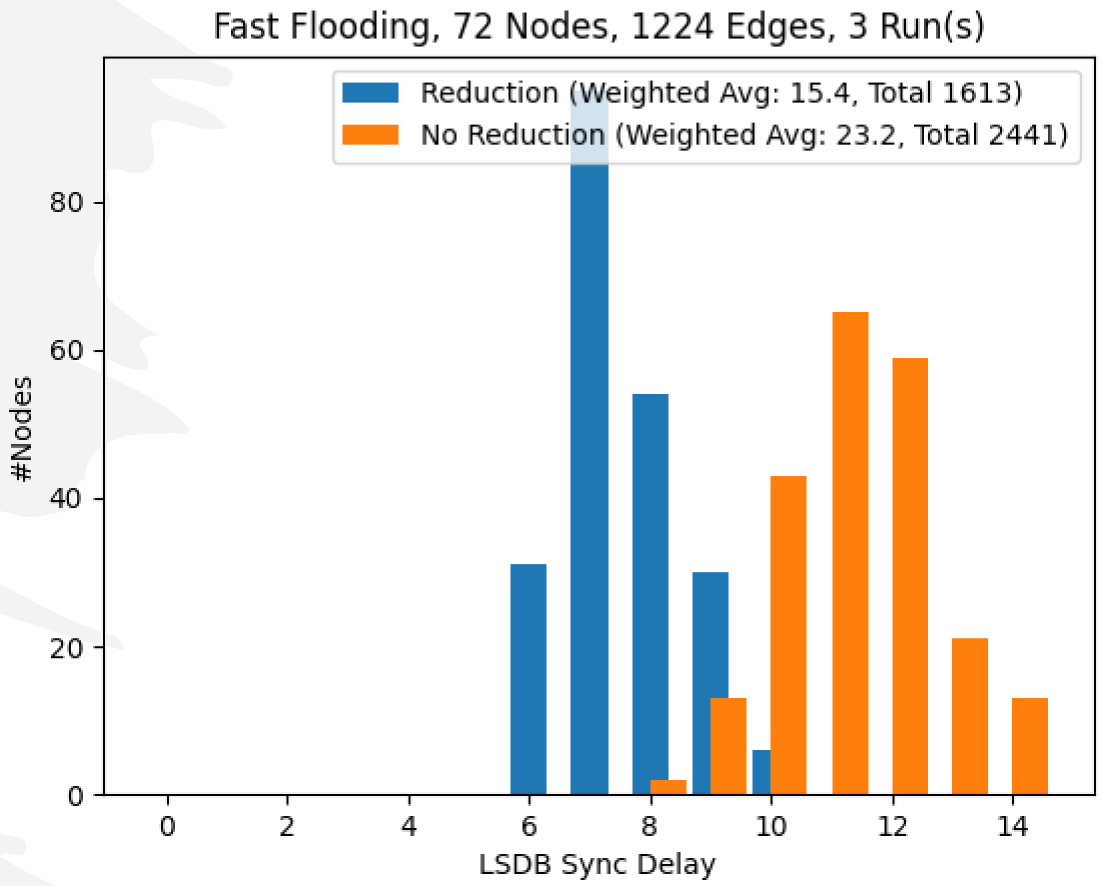




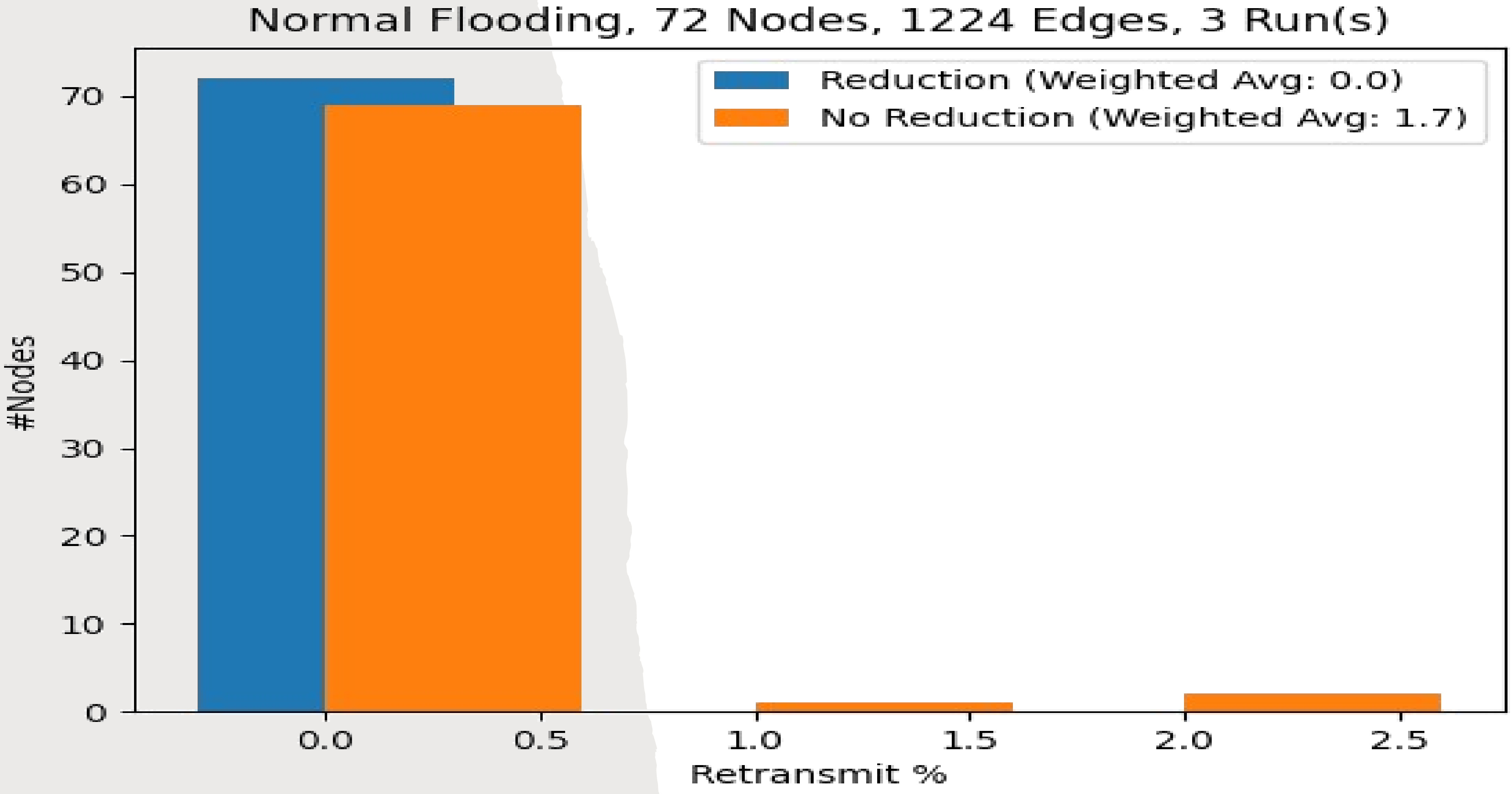
LSDB Sync Delay

- Reduction: 30%
- Fast Flooding: 40% speedup



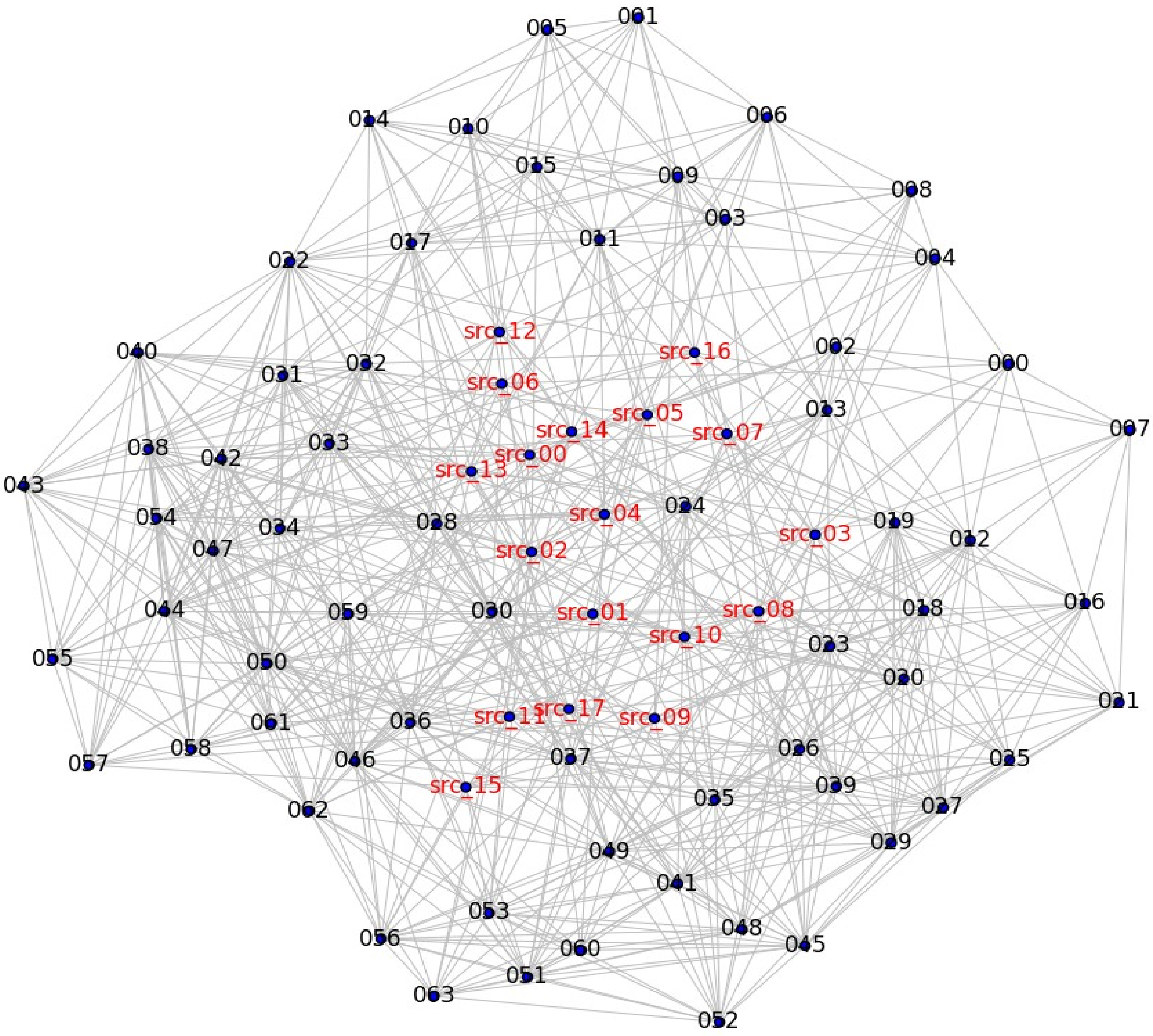


RTX% Shows


Normal Flooding, 72 Nodes, 1224 Edges, 3 Run(s)
Reduction (Weighted Avg: 0.0)
No Reduction (Weighted Avg: 1.7)
#Nodes
0
10
20
30
40
50
60
70
0.0
0.5
1.0
1.5
2.0
2.5
Retransmit %

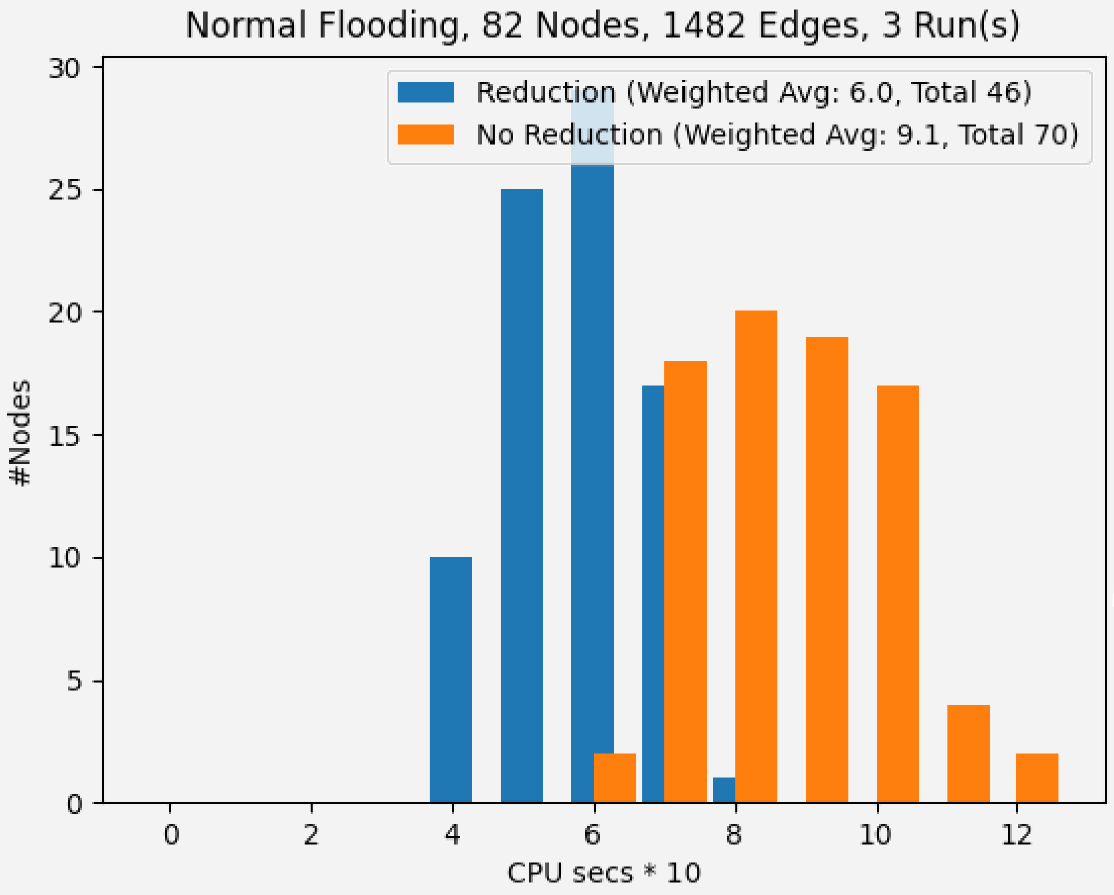
Even Bigger Bigger Graph

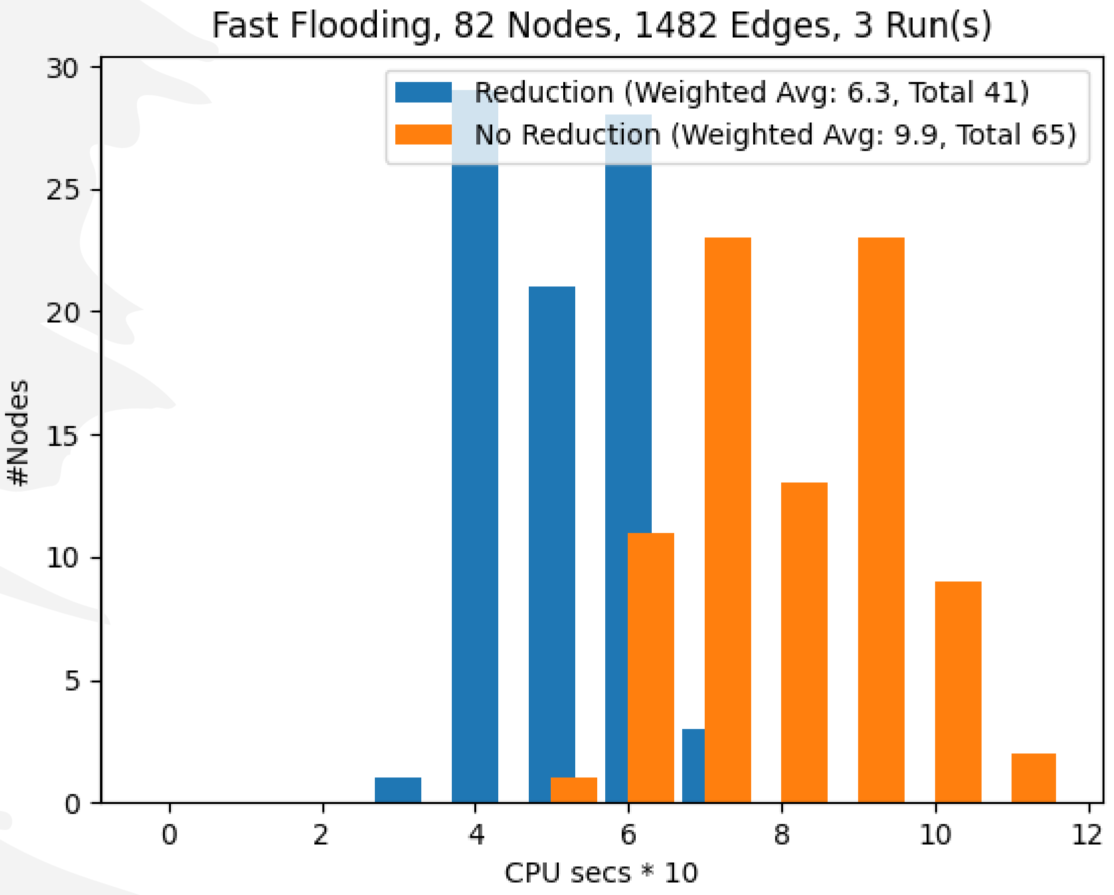
- Fewer (0.7K) LSPs
- 82 nodes
- 18 sources
- 741 adjacencies



CPU Consumption

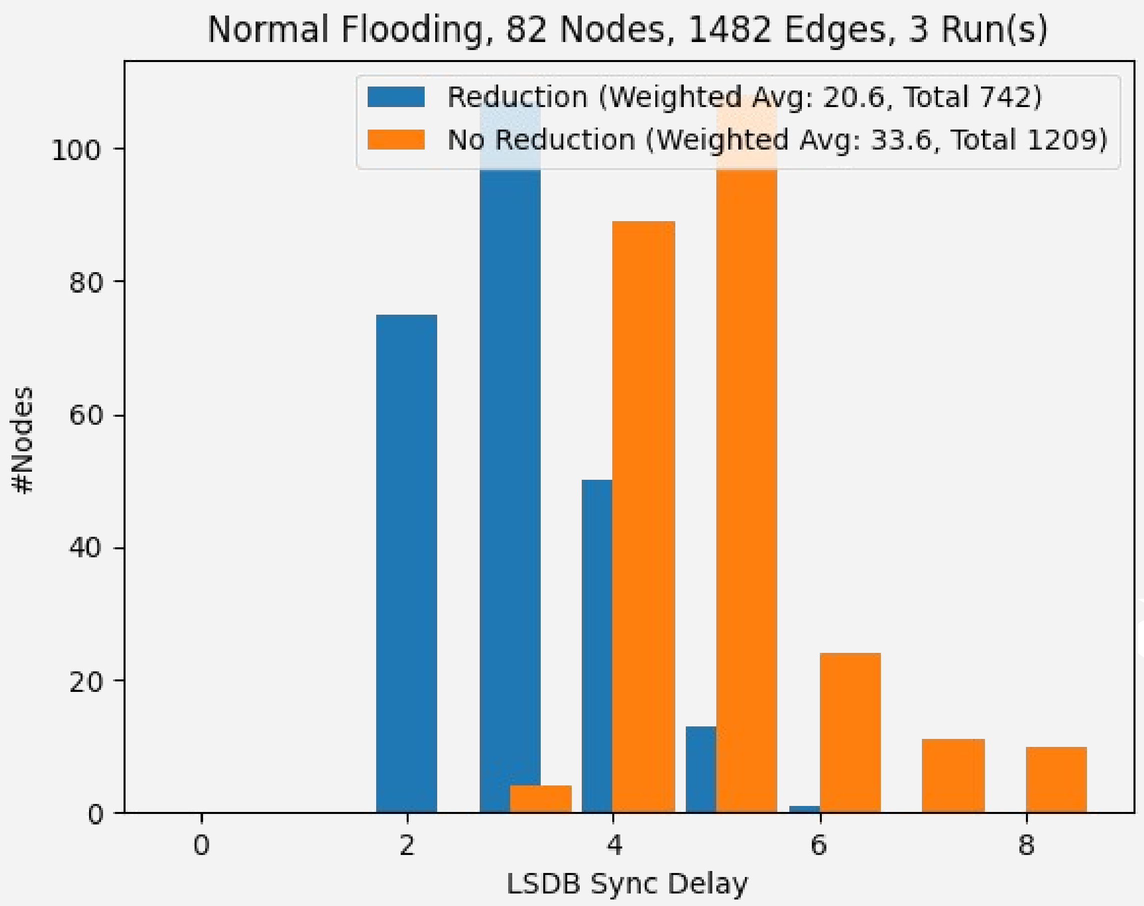
- Reduction: 50%
- Fast Flooding: 10% less

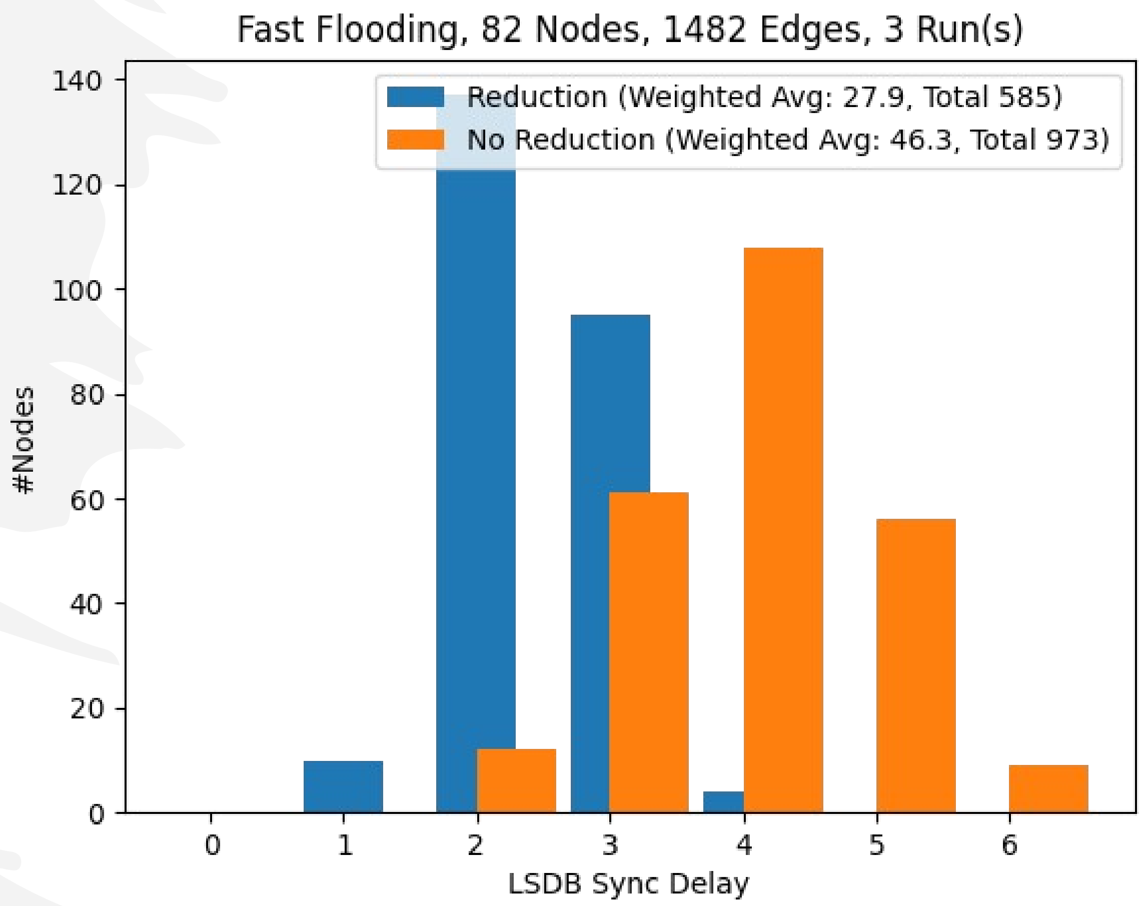




LSDB Sync Delay

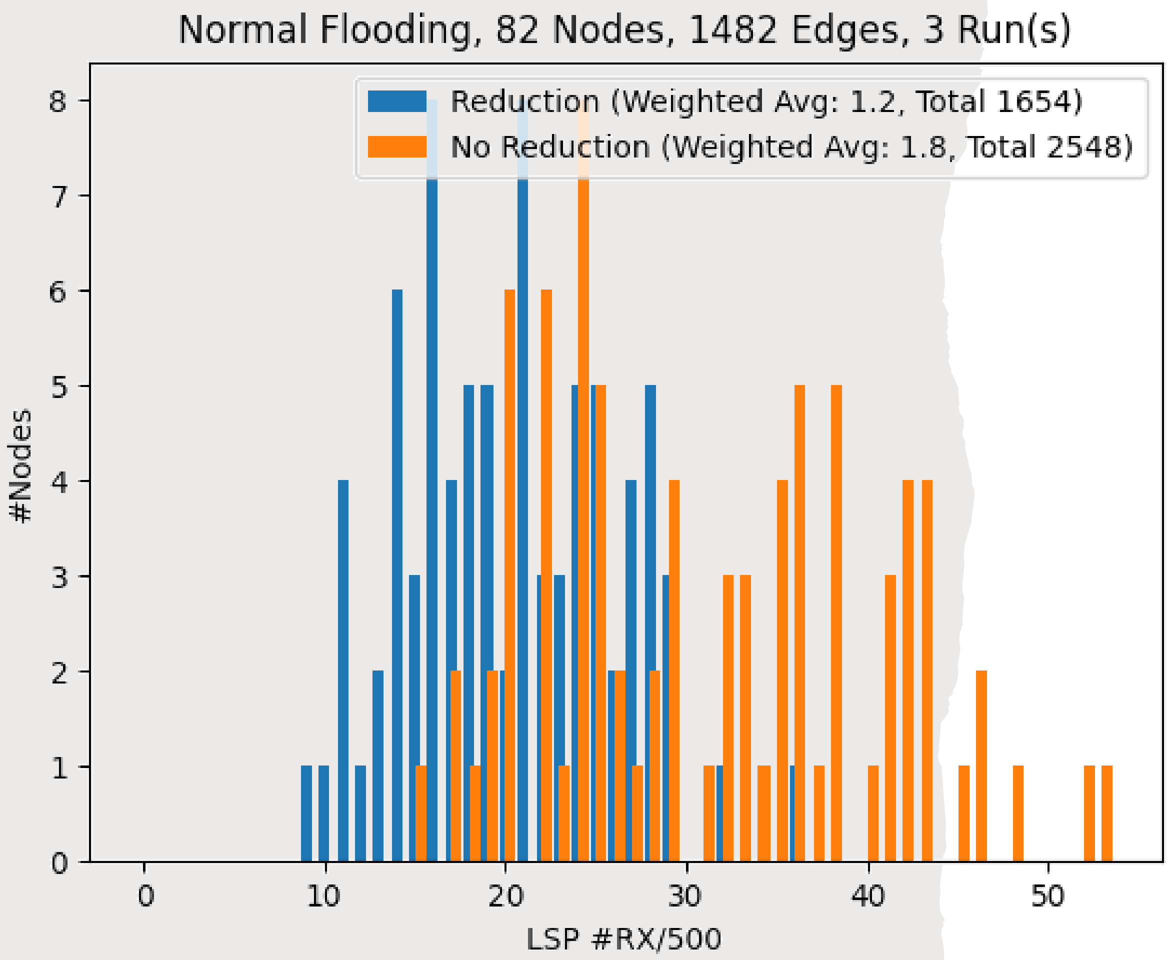
- Reduction: 40% speedup
- Fast Flooding: another 20% speedup

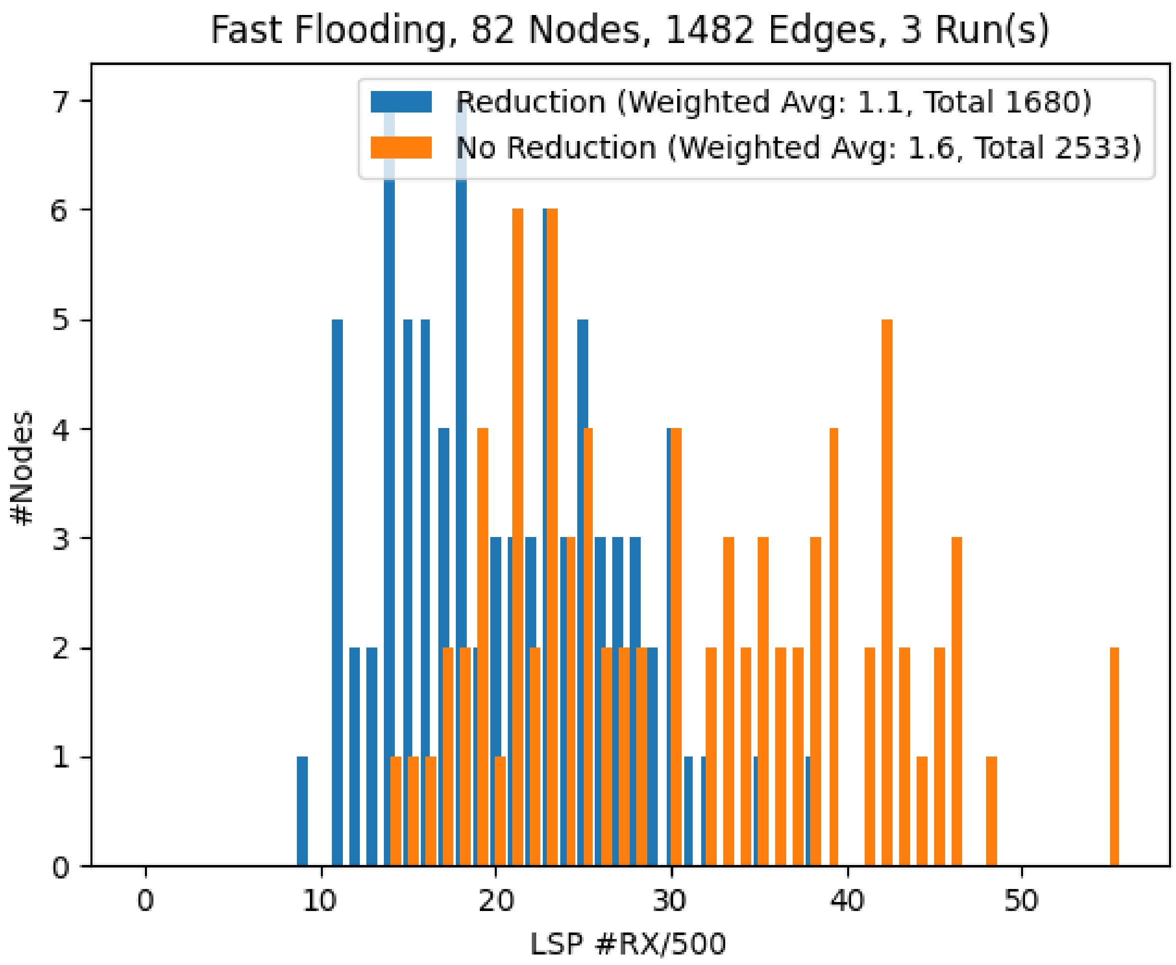




RX LSP (divided by 500)

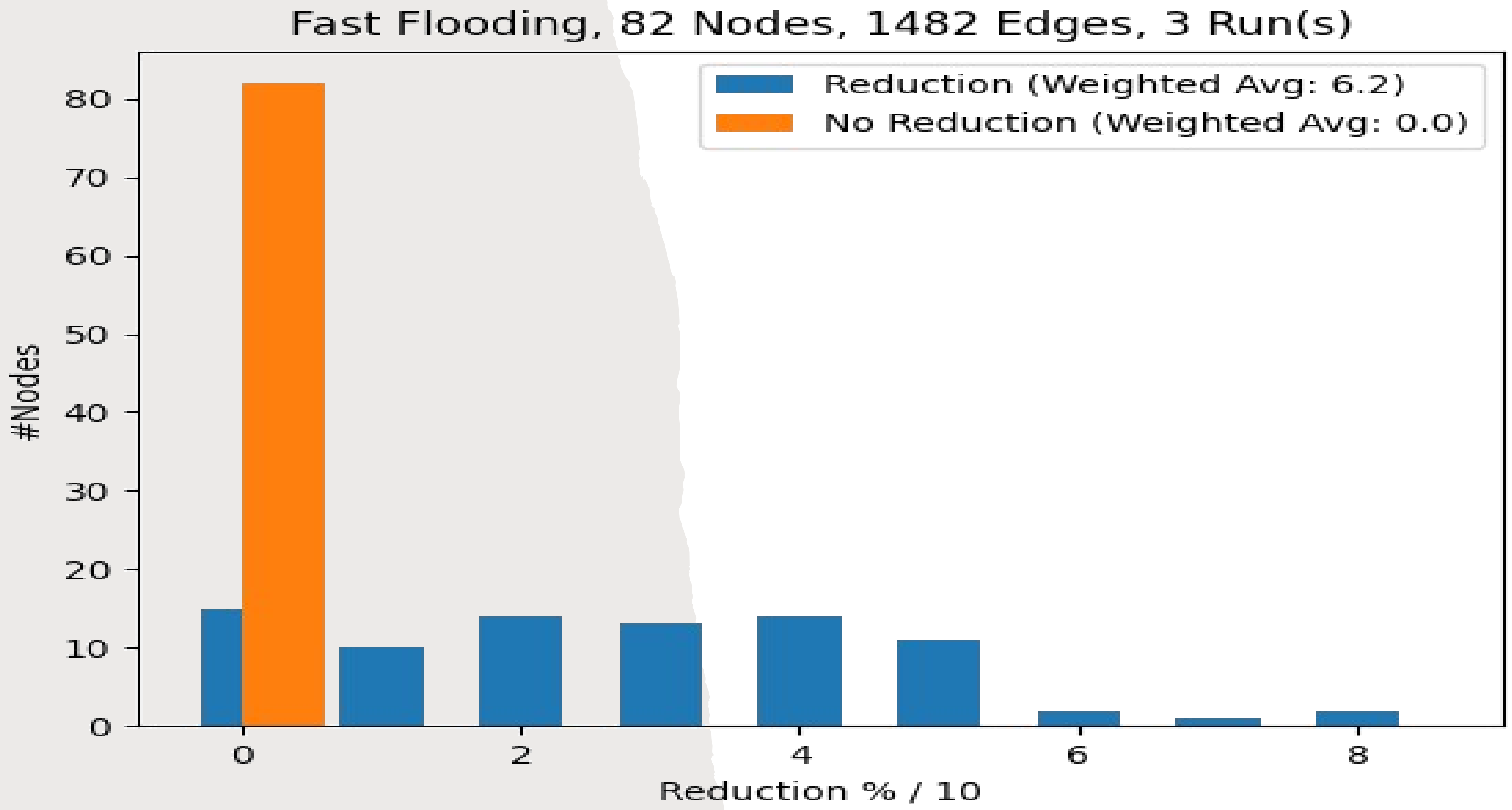
- Reduction: 35%



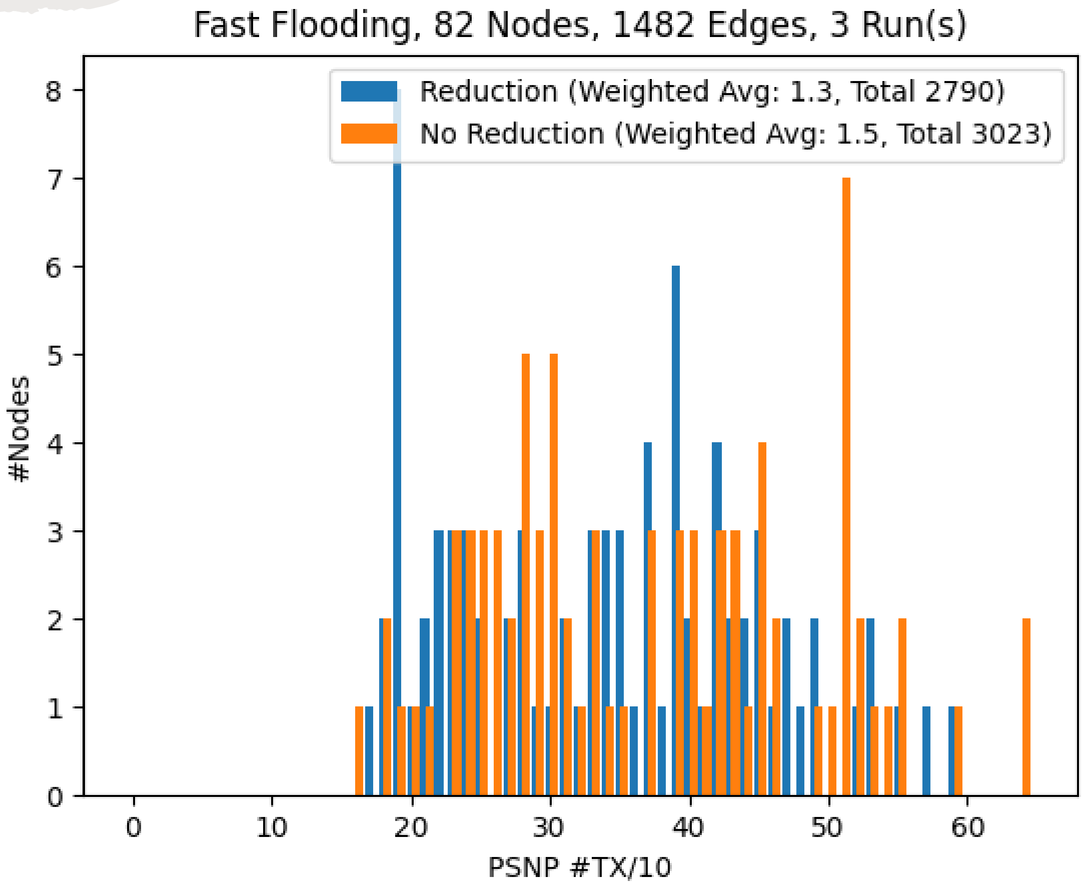


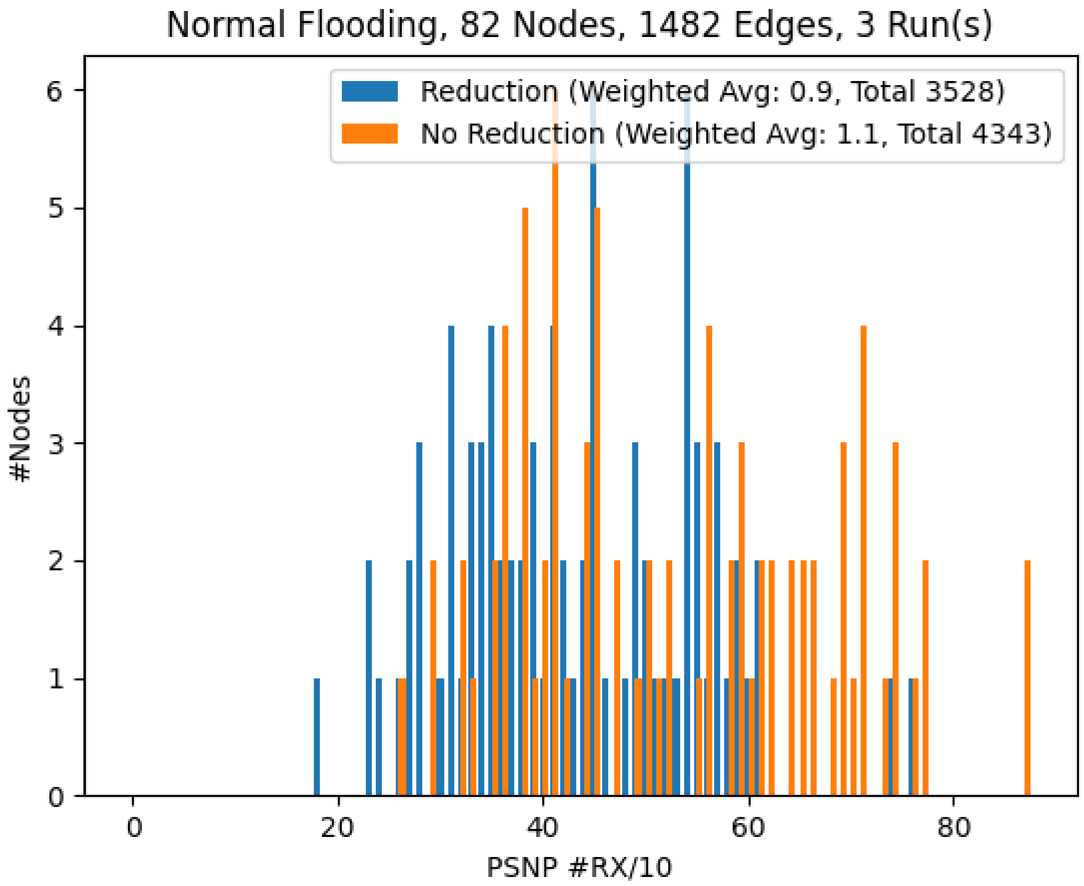
Reduction Distribution

Reduction: Lots of nodes needed since LSPs come from all kind of directions



What about all those extra PSNPs?





Conclusions Once Again

- Massive CPU saving with reduction in all scenarios (~60%)
- Good LSP TX/RX Reduction (~25-70%)
  - Depends on #sources and volume per source
- Decent to good LSDB Synchronization Speedup (~0-40%)
  - 2 cores vs. 1 neglectable differences
  - Difference larger with faster flooding and more sources and more LSPs
- On slow flooding with lots LSPs RTX% with reduction looks different
  - Only about ½ compared to no reduction
  - With reduction RTX% concentrated on fewer nodes
- Who reduces depends lots on direction from which LSPs arrive