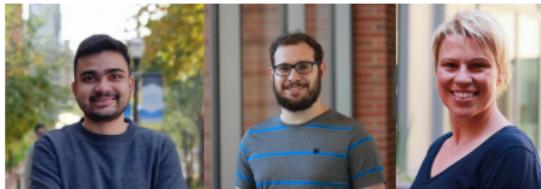


# LDPC Code Design for Data Availability Attacks in Blockchain Systems

Debaranab Mitra, Lev Tauz, and Lara Dolecek

Electrical and Computer Engineering  
University of California, Los Angeles

University of Toronto  
June 28 2021



**Samueli**  
School of Engineering

# Table of Contents

1. Background and Central Problems
2. Data Availability Attacks on Light Nodes
3. Data Availability Attacks in Side Blockchains
4. Conclusion

# Table of Contents

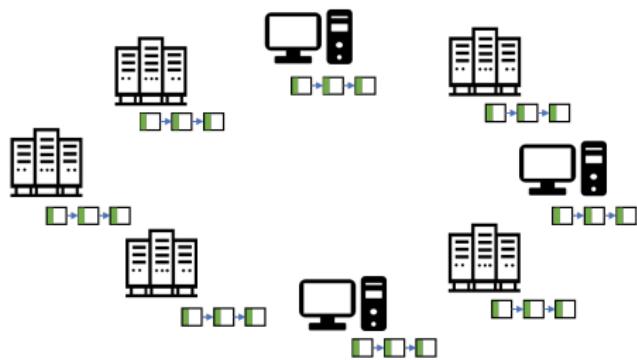
1. Background and Central Problems
2. Data Availability Attacks on Light Nodes
3. Data Availability Attacks in Side Blockchains
4. Conclusion

# Blockchain



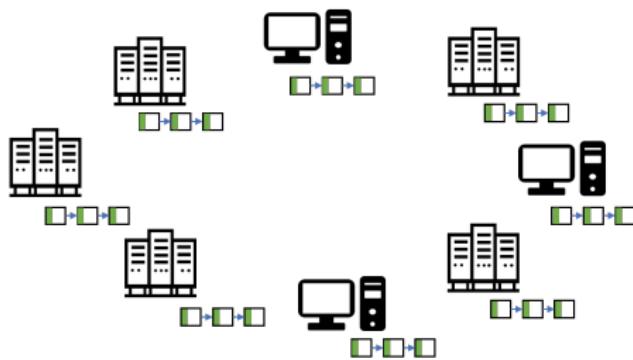
- ▶ Distributed Ledger
- ▶ Decentralized trust platforms
- ▶ Application:
  - Finance and currency
  - Healthcare services
  - Supply chain management
  - Industrial IoT
  - e-voting

# Central Problem 1: Prohibitive Storage Overhead



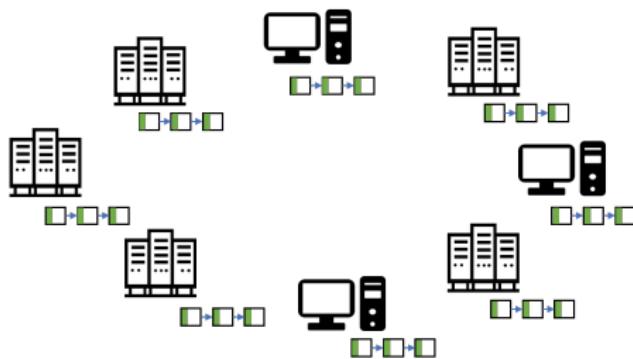
- ▶ Ledger maintained by a network of nodes

# Central Problem 1: Prohibitive Storage Overhead



- ▶ Ledger maintained by a network of nodes
- ▶ Each node maintains a local copy of the ledger

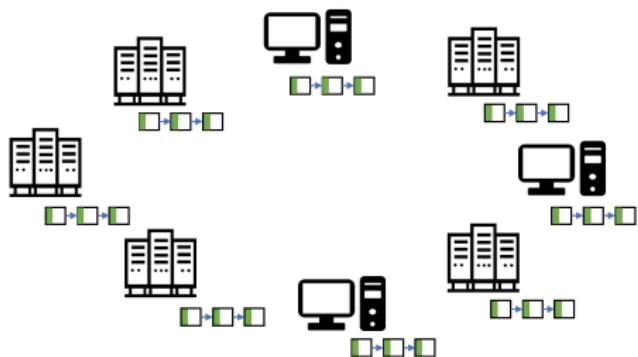
# Central Problem 1: Prohibitive Storage Overhead



- ▶ Ledger maintained by a network of nodes
- ▶ Each node maintains a local copy of the ledger

Significant storage overhead

# Central Problem 1: Prohibitive Storage Overhead



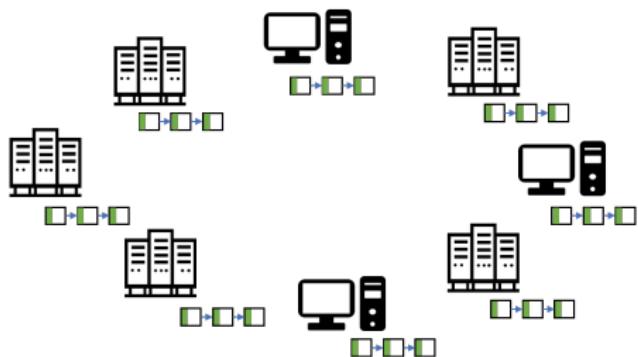
- ▶ Ledger maintained by a network of nodes
- ▶ Each node maintains a local copy of the ledger
- ▶ Bitcoin ledger size  $\sim 350\text{GB}^1$
- ▶ Ethereum ledger size  $\sim 830\text{GB}^2$

Significant storage overhead

As of 6/22/2021, <sup>1</sup><https://www.blockchain.com/charts/blocks-size>

<sup>2</sup><https://etherscan.io/chartsync/chaindefault>

# Central Problem 1: Prohibitive Storage Overhead



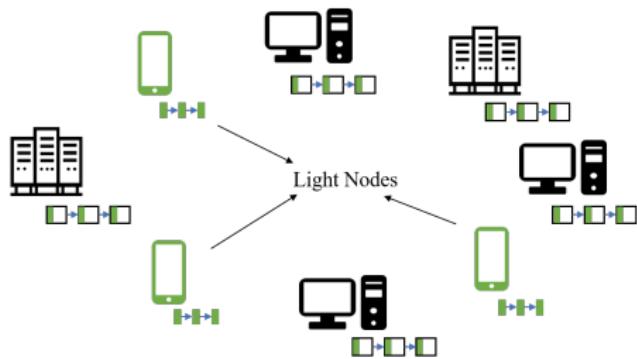
- ▶ Ledger maintained by a network of nodes
- ▶ Each node maintains a local copy of the ledger
- ▶ Prohibitive for resource limited nodes
- ▶ Bitcoin ledger size  $\sim 350\text{GB}^1$
- ▶ Ethereum ledger size  $\sim 830\text{GB}^2$

Significant storage overhead

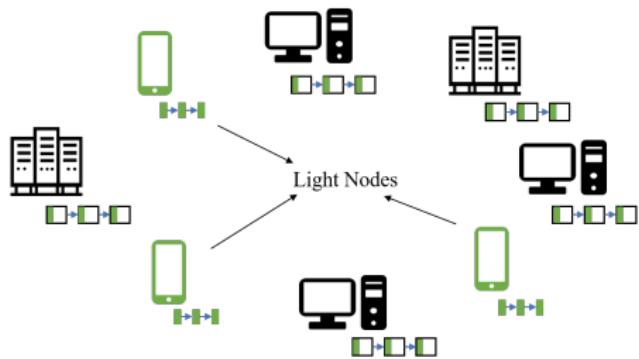
As of 6/22/2021, <sup>1</sup><https://www.blockchain.com/charts/blocks-size>

<sup>2</sup><https://etherscan.io/chartsync/chaindefault>

# Solution: Allowing Light Nodes

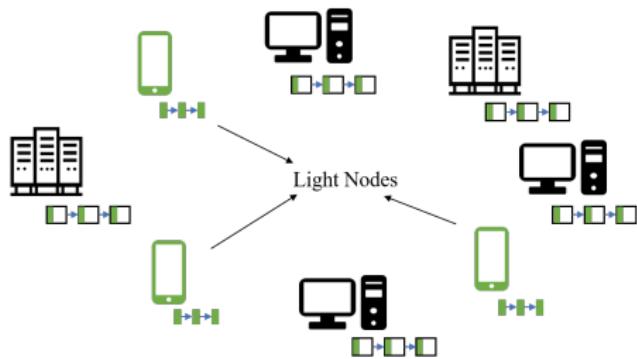


# Solution: Allowing Light Nodes



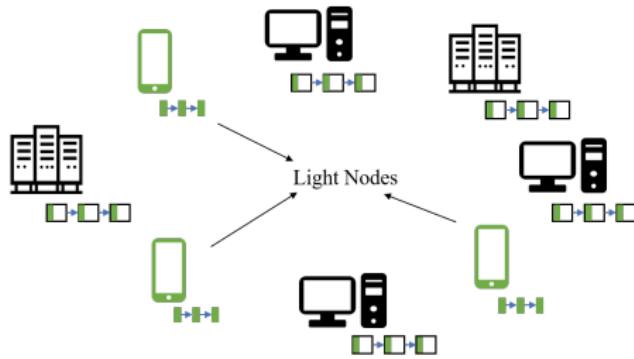
- ▶ Only store block headers  
(total size  $\sim 1\text{GB}$  for Ethereum)

# Solution: Allowing Light Nodes



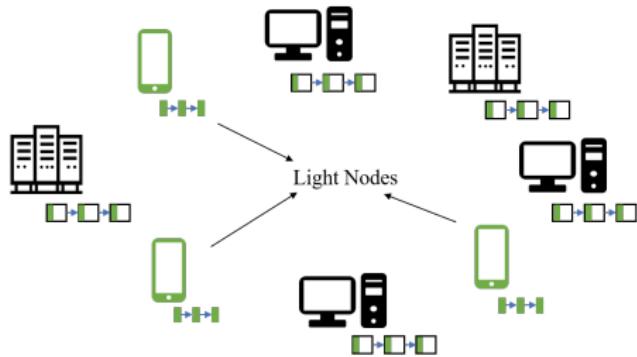
- ▶ Only store block headers (total size  $\sim 1\text{GB}$  for Ethereum)
- ▶ Can verify transaction inclusion in a block

# Solution: Allowing Light Nodes



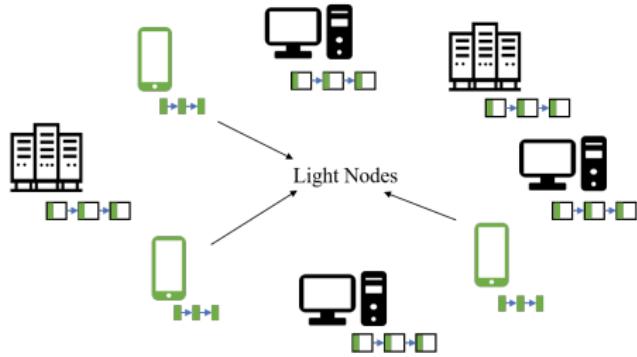
- ▶ Only store block headers (total size  $\sim 1\text{GB}$  for Ethereum)
- ▶ Can verify transaction inclusion in a block
- ▶ Cannot verify transaction correctness

# Solution: Allowing Light Nodes

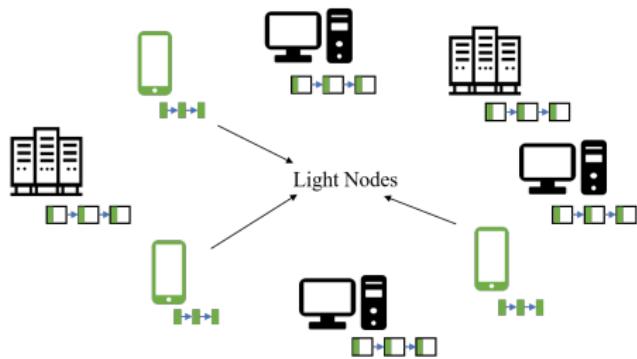


- ▶ Only store block headers (total size  $\sim 1\text{GB}$  for Ethereum)
- ▶ Can verify transaction inclusion in a block
- ▶ Cannot verify transaction correctness  $\rightarrow$  Rely on honest Full nodes for fraud notification

## Central Problem 2: Poor Throughput and Latency



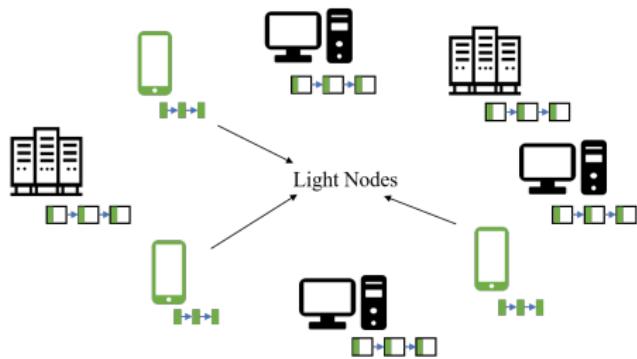
## Central Problem 2: Poor Throughput and Latency



	Transaction throughput	Confirmation Latency
Bitcoin		
Ethereum		

[Li '20]

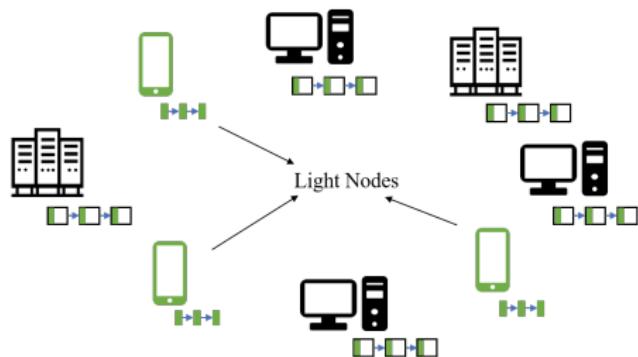
## Central Problem 2: Poor Throughput and Latency



	Transaction throughput	Confirmation Latency
Bitcoin	5-7 transactions/s	
Ethereum		

[Li '20]

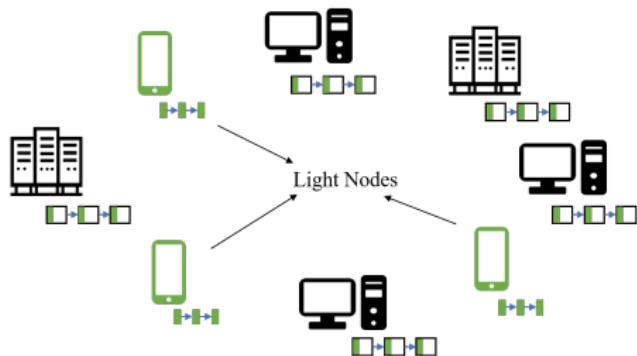
## Central Problem 2: Poor Throughput and Latency



	Transaction throughput	Confirmation Latency
Bitcoin	5-7 transactions/s	hours
Ethereum		

[Li '20]

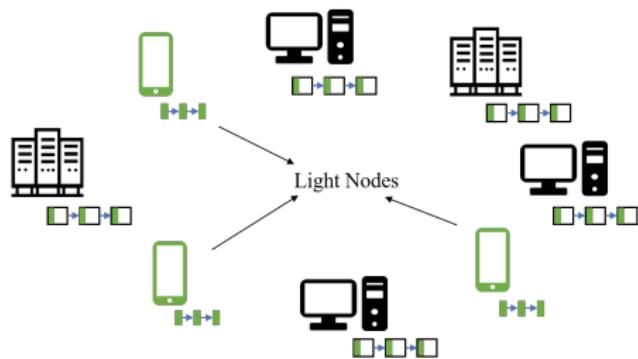
## Central Problem 2: Poor Throughput and Latency



	Transaction throughput	Confirmation Latency
Bitcoin	5-7 transactions/s	hours
Ethereum	30 transactions/s	

[Li '20]

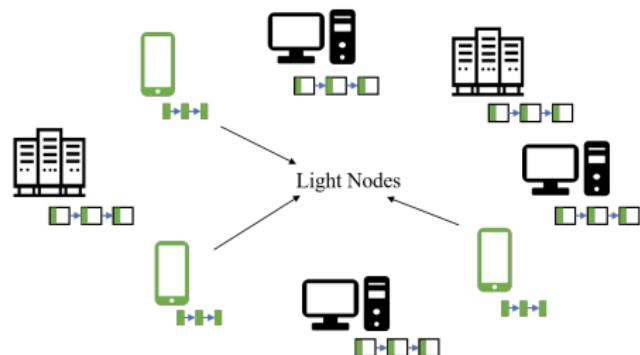
## Central Problem 2: Poor Throughput and Latency



	Transaction throughput	Confirmation Latency
Bitcoin	5-7 transactions/s	hours
Ethereum	30 transactions/s	tens of minutes

[Li '20]

## Central Problem 2: Poor Throughput and Latency



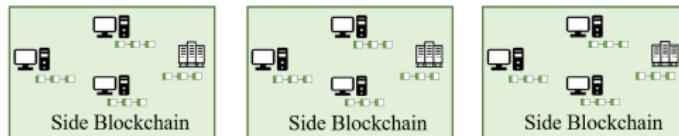
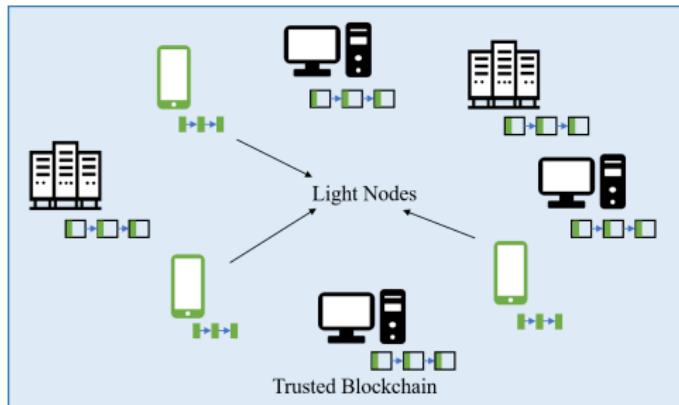
	Transaction throughput	Confirmation Latency
Bitcoin	5-7 transactions/s	hours
Ethereum	30 transactions/s	tens of minutes

[Li '20]

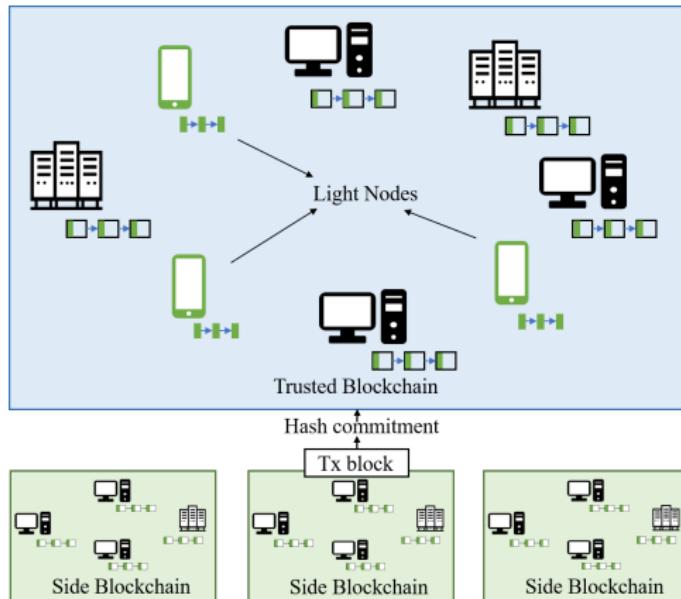
Contrast: Visa processes more than 10,000 transactions/s<sup>3</sup>

<sup>3</sup><https://usa.visa.com>

# Solution: Running Side Blockchains



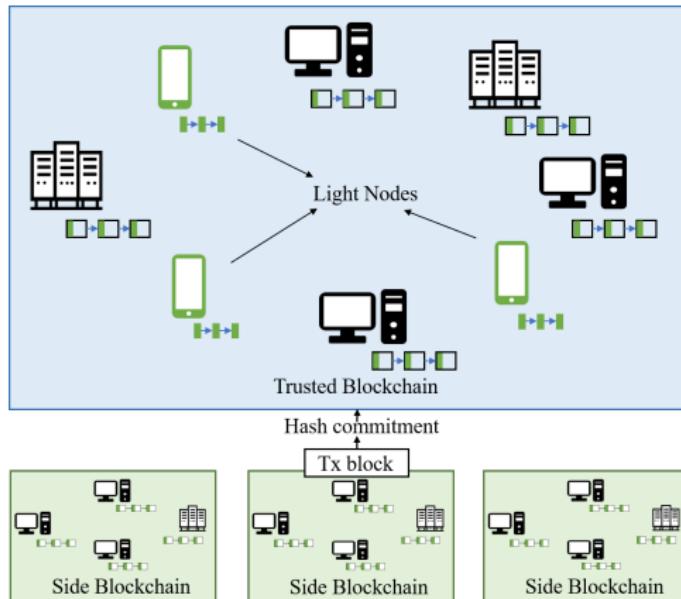
# Solution: Running Side Blockchains



Side Blockchain nodes:

- ▶ Push hash commitment of their block to trusted blockchain

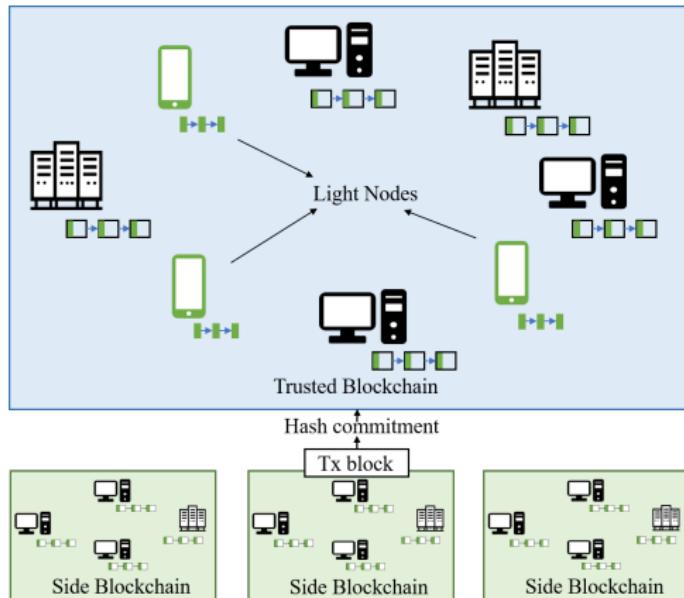
# Solution: Running Side Blockchains



Side Blockchain nodes:

- ▶ Push hash commitment of their block to trusted blockchain
- ▶ Order of transactions same as hash order in trusted blockchain

# Solution: Running Side Blockchains



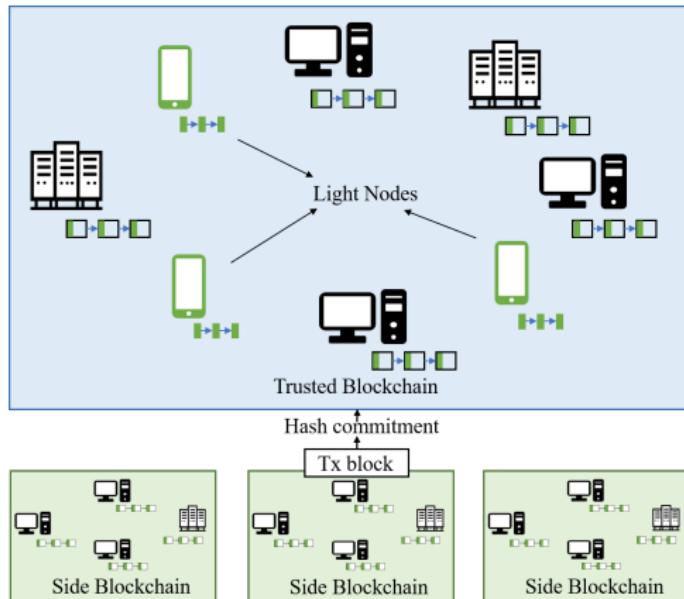
Side Blockchain nodes:

- ▶ Push hash commitment of their block to trusted blockchain
- ▶ Order of transactions same as hash order in trusted blockchain

Trusted Blockchain:

- ▶ Only store the hash of the side blockchain

# Solution: Running Side Blockchains



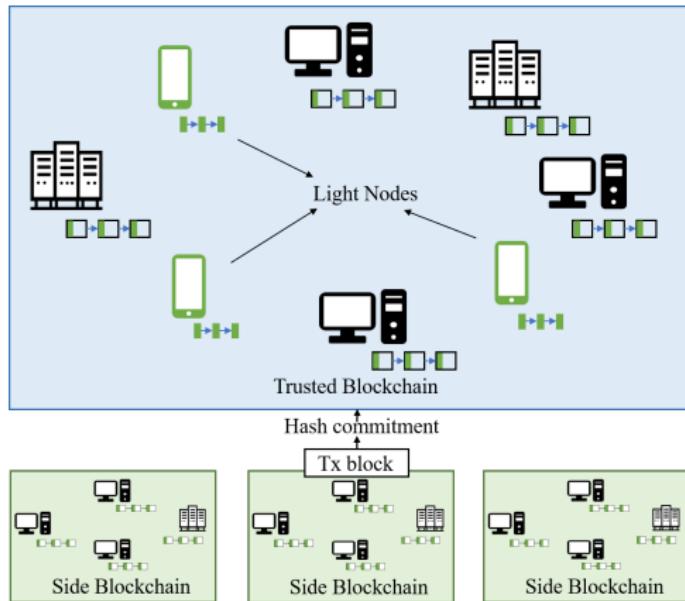
Side Blockchain nodes:

- ▶ Push hash commitment of their block to trusted blockchain
- ▶ Order of transactions same as hash order in trusted blockchain

Trusted Blockchain:

- ▶ Only store the hash of the side blockchain
- ▶ Side blockchains make commitments in parallel

# Solution: Running Side Blockchains



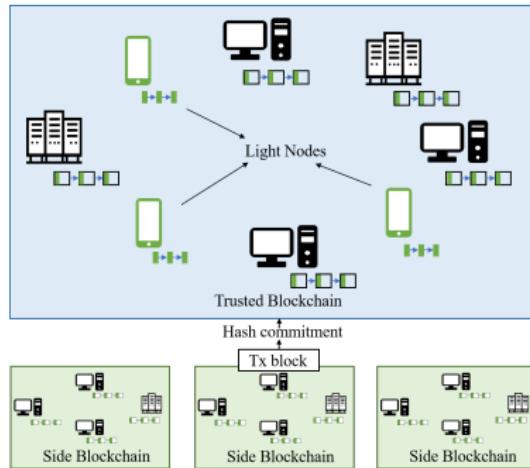
Side Blockchain nodes:

- ▶ Push hash commitment of their block to trusted blockchain
- ▶ Order of transactions same as hash order in trusted blockchain

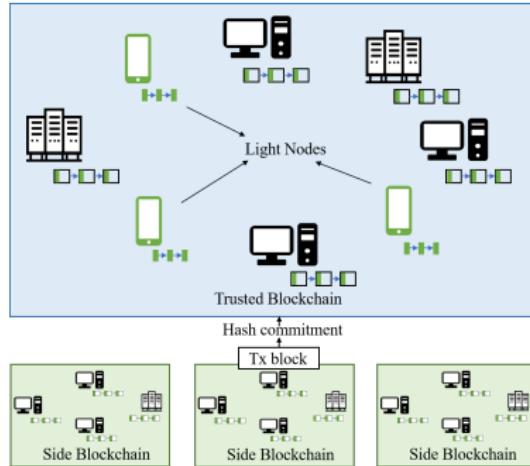
Trusted Blockchain:

- ▶ Only store the hash of the side blockchain
- ▶ Side blockchains make commitments in parallel
- ▶ Leads to higher transaction throughput

# Data Availability (DA) Attack



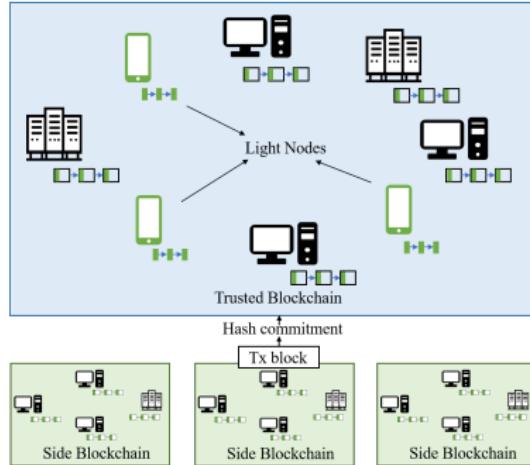
# Data Availability (DA) Attack



Light Nodes:

- ▶ Only store the header of each block

# Data Availability (DA) Attack



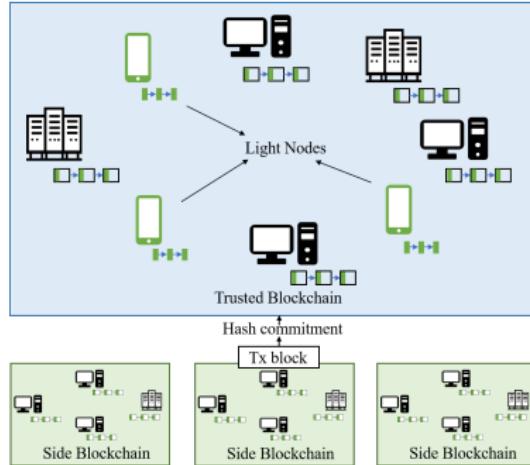
## Light Nodes:

- ▶ Only store the header of each block

## Trusted Blockchain:

- ▶ Only store the hash commitment of the side blockchain blocks

# Data Availability (DA) Attack



**Light Nodes:**

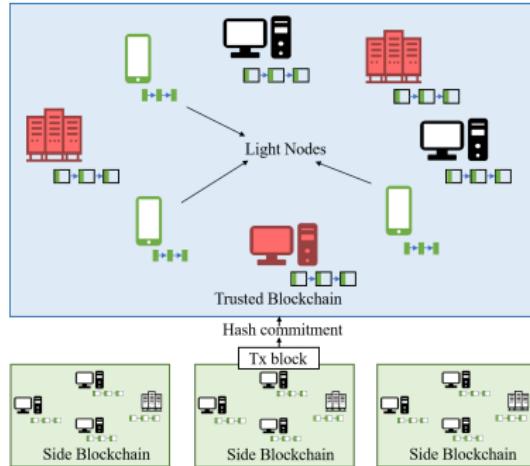
- ▶ Only store the header of each block

**Trusted Blockchain:**

- ▶ Only store the hash commitment of the side blockchain blocks

Both are vulnerable to Data Availability attacks

# Data Availability (DA) Attack



## Light Nodes:

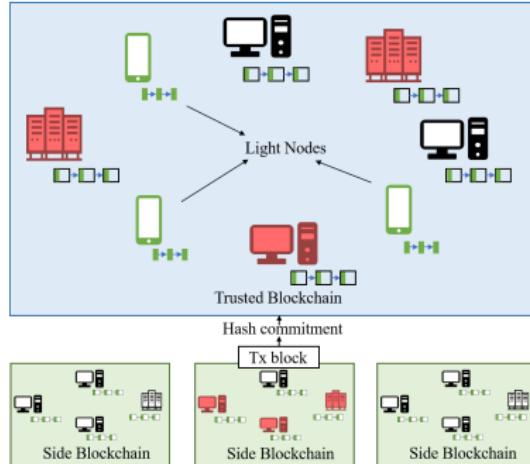
- ▶ Only store the header of each block
- ▶ Occur when there is a dishonest majority of full nodes

## Trusted Blockchain:

- ▶ Only store the hash commitment of the side blockchain blocks

Both are vulnerable to Data Availability attacks

# Data Availability (DA) Attack



## Light Nodes:

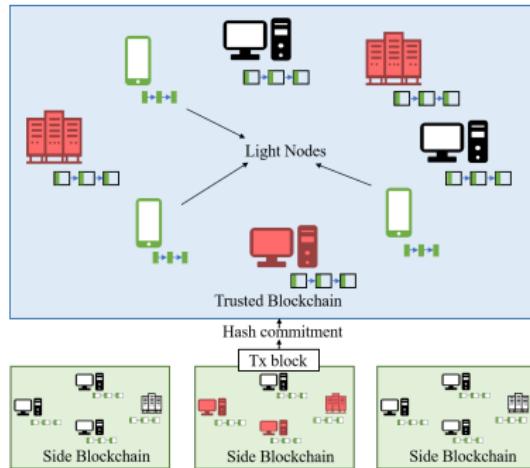
- ▶ Only store the header of each block
- ▶ Occur when there is a dishonest majority of full nodes

## Trusted Blockchain:

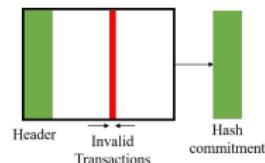
- ▶ Only store the hash commitment of the side blockchain blocks
- ▶ Occur when there is a dishonest majority of side blockchain nodes

Both are vulnerable to Data Availability attacks

# Data Availability (DA) Attack



Adversary generates an invalid block



## Light Nodes:

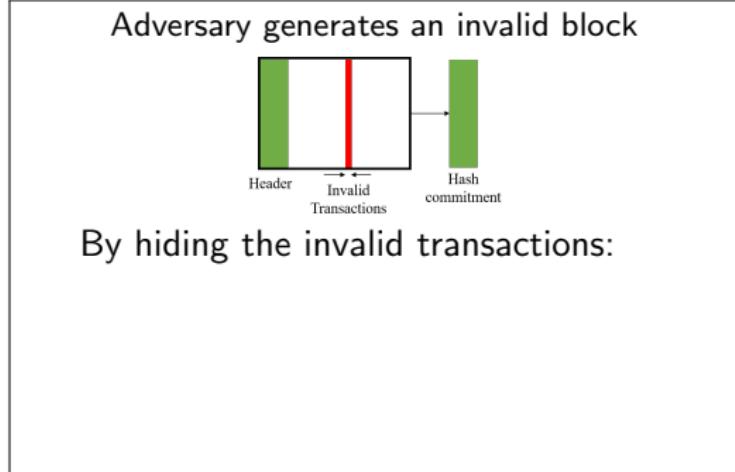
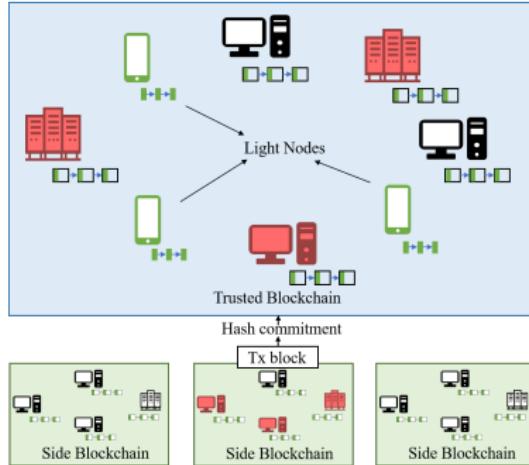
- ▶ Only store the header of each block
- ▶ Occur when there is a dishonest majority of full nodes

## Trusted Blockchain:

- ▶ Only store the hash commitment of the side blockchain blocks
- ▶ Occur when there is a dishonest majority of side blockchain nodes

Both are vulnerable to Data Availability attacks

# Data Availability (DA) Attack



## Light Nodes:

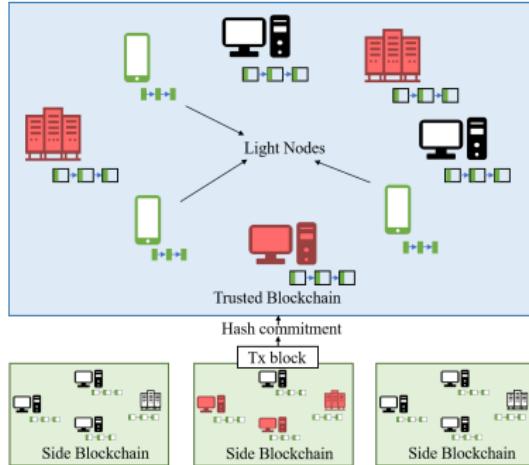
- ▶ Only store the header of each block
- ▶ Occur when there is a dishonest majority of full nodes

## Trusted Blockchain:

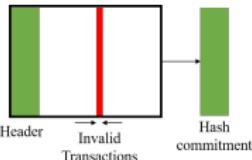
- ▶ Only store the hash commitment of the side blockchain blocks
- ▶ Occur when there is a dishonest majority of side blockchain nodes

Both are vulnerable to Data Availability attacks

# Data Availability (DA) Attack



Adversary generates an invalid block



By hiding the invalid transactions:

- ▶ Adversarial full nodes trick light nodes to accept invalid header

**Light Nodes:**

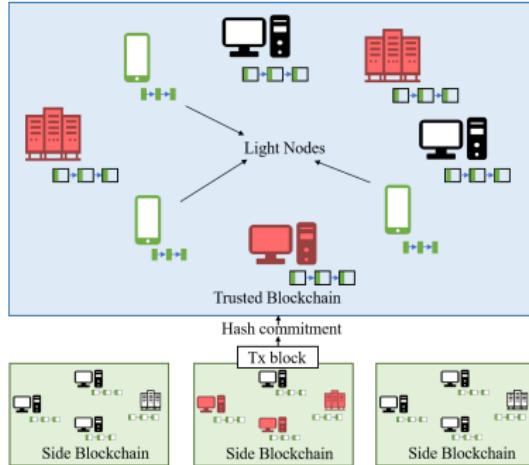
- ▶ Only store the header of each block
- ▶ Occur when there is a dishonest majority of full nodes

**Trusted Blockchain:**

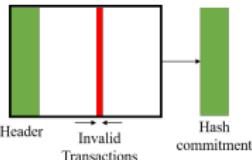
- ▶ Only store the hash commitment of the side blockchain blocks
- ▶ Occur when there is a dishonest majority of side blockchain nodes

Both are vulnerable to Data Availability attacks

# Data Availability (DA) Attack



Adversary generates an invalid block



By hiding the invalid transactions:

- ▶ Adversarial full nodes trick light nodes to accept invalid header
- ▶ Adversarial side blockchain nodes commit invalid hash to trusted blockchain

Light Nodes:

- ▶ Only store the header of each block
- ▶ Occur when there is a dishonest majority of full nodes

Trusted Blockchain:

- ▶ Only store the hash commitment of the side blockchain blocks
- ▶ Occur when there is a dishonest majority of side blockchain nodes

Both are vulnerable to Data Availability attacks

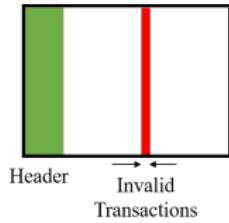
# Table of Contents

1. Background and Central Problems
2. Data Availability Attacks on Light Nodes
3. Data Availability Attacks in Side Blockchains
4. Conclusion

# Data Availability(DA) Attack on Light Nodes

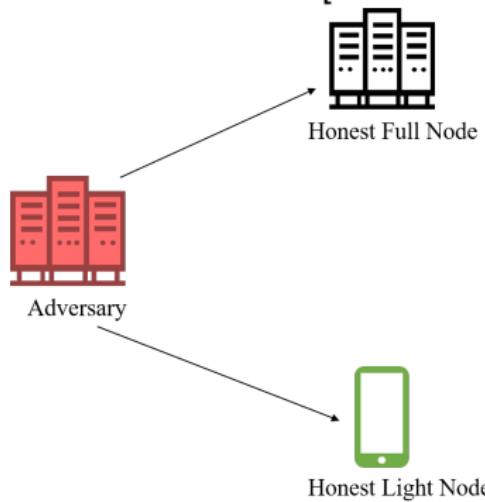
Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]

Adversary creates an invalid block

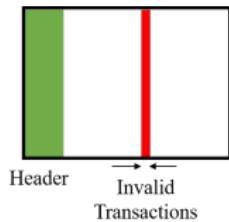


# Data Availability(DA) Attack on Light Nodes

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]

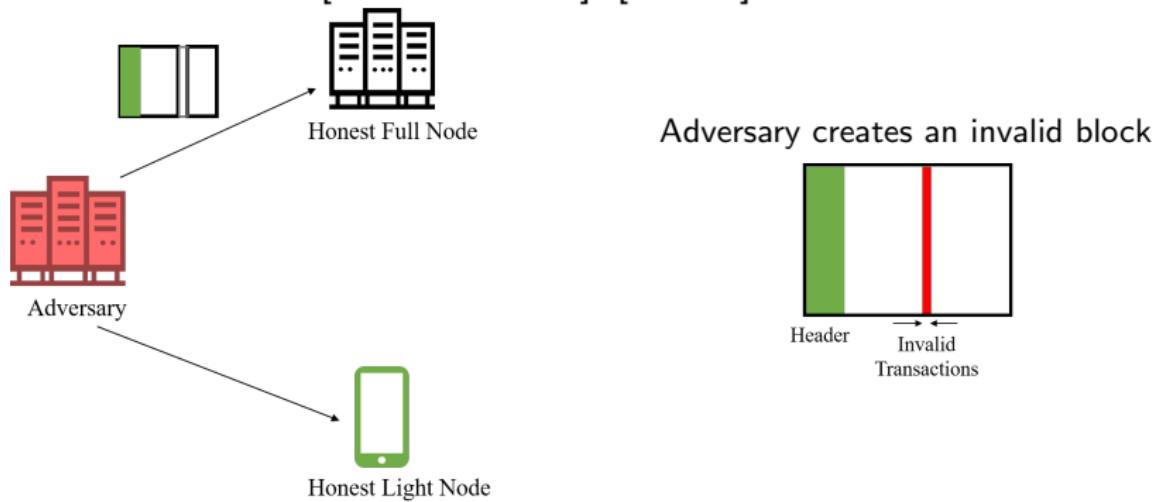


Adversary creates an invalid block



# Data Availability(DA) Attack on Light Nodes

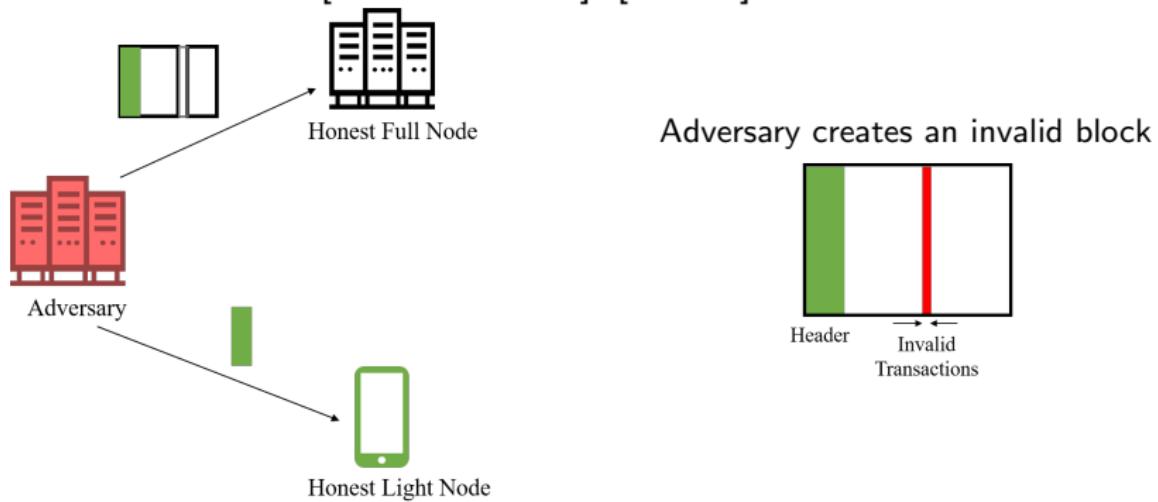
Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]



- ▶ **Adversary:** Provides block to Full node but hides invalid portion

# Data Availability(DA) Attack on Light Nodes

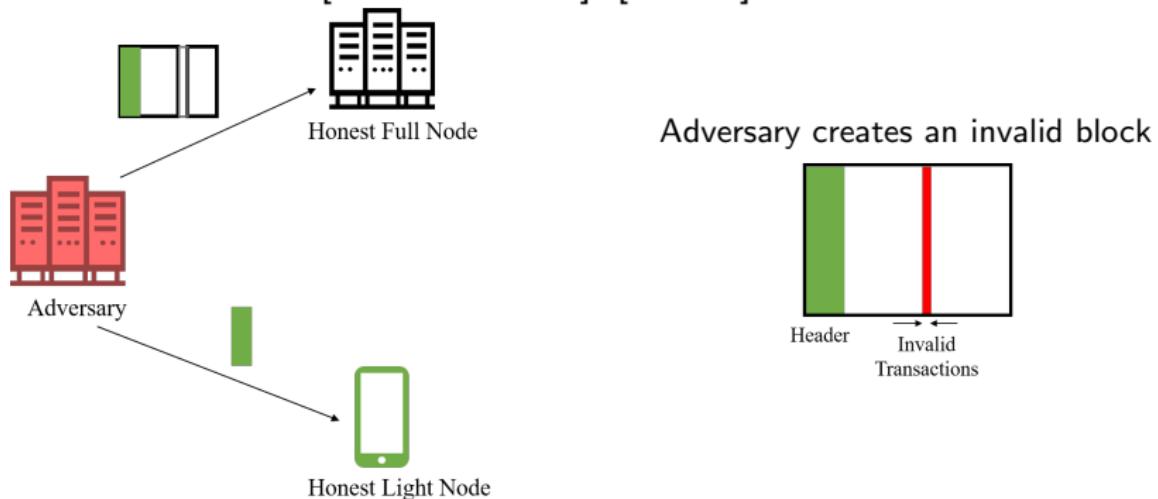
Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]



- ▶ **Adversary:** Provides block to Full node but hides invalid portion  
Provides header to Light node

# Data Availability(DA) Attack on Light Nodes

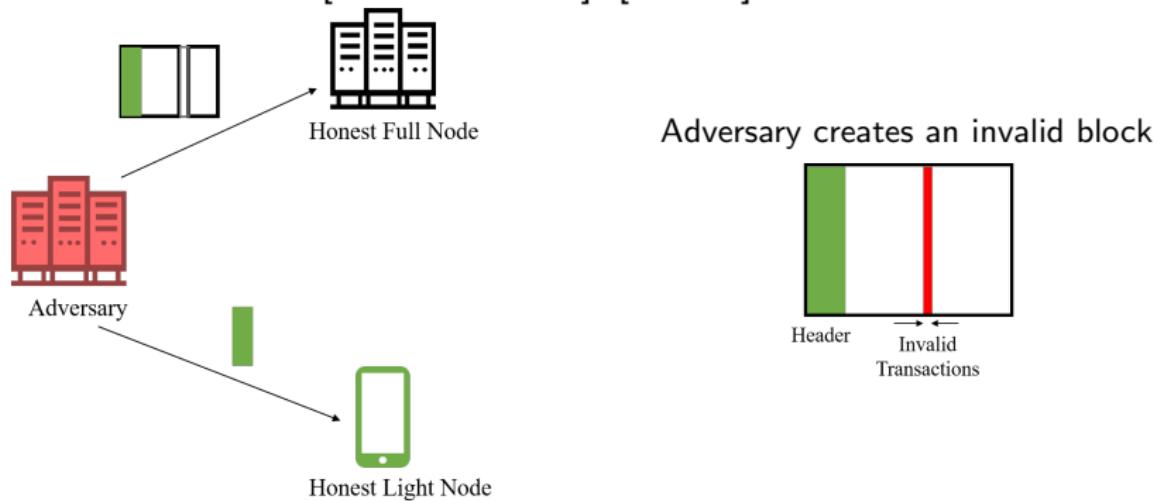
Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]



- ▶ **Adversary:** Provides block to Full node but hides invalid portion  
Provides header to Light node
- ▶ **Honest Nodes:** Cannot verify missing transactions

# Data Availability(DA) Attack on Light Nodes

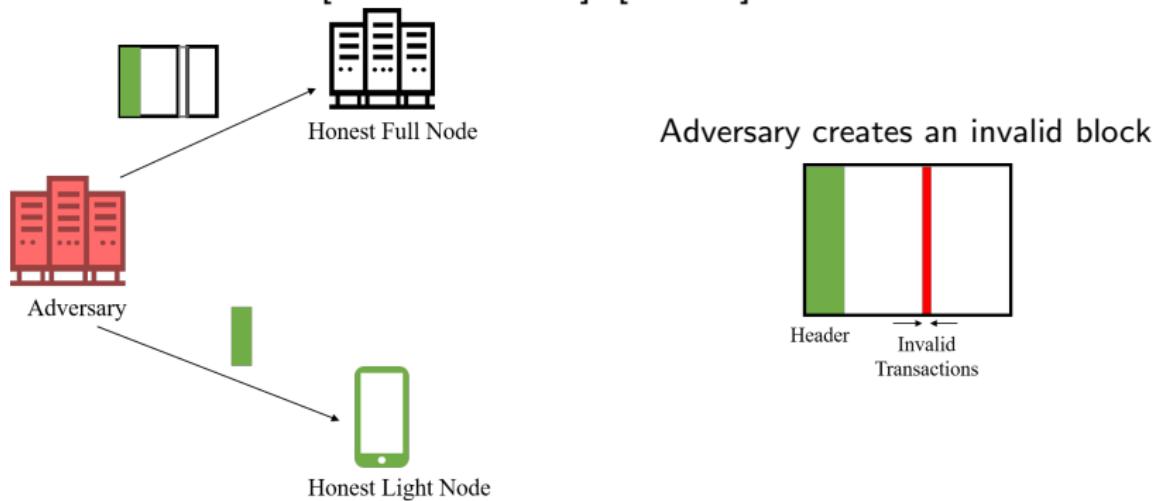
Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]



- ▶ **Adversary:** Provides block to Full node but hides invalid portion  
Provides header to Light node
- ▶ **Honest Nodes:** Cannot verify missing transactions → No fraud proof

# Data Availability(DA) Attack on Light Nodes

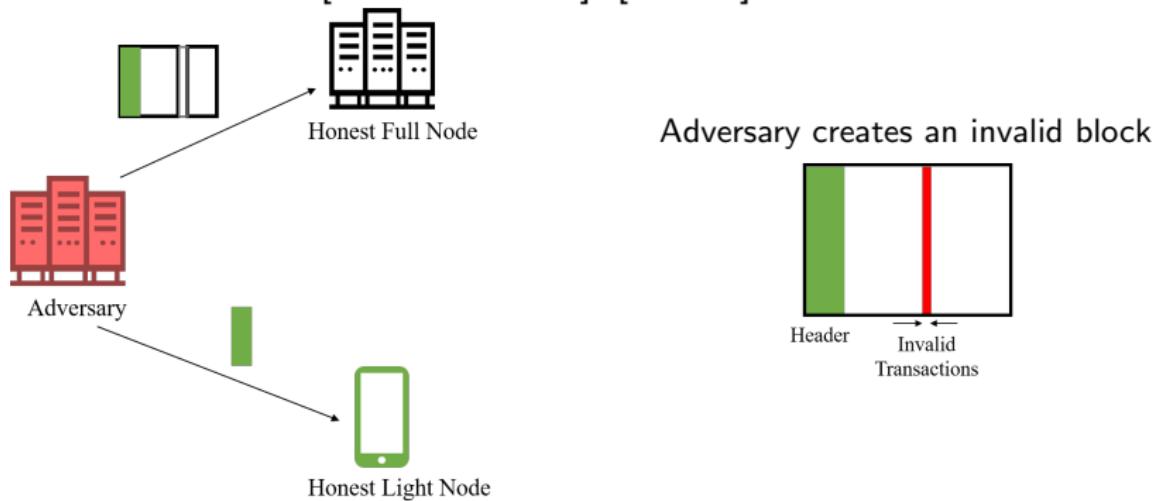
Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]



- ▶ Adversary: Provides block to Full node but hides invalid portion  
Provides header to Light node
- ▶ Honest Nodes: Cannot verify missing transactions → No fraud proof
- ▶ Light Nodes: No fraud proof

# Data Availability(DA) Attack on Light Nodes

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam '18], [Yu '19]

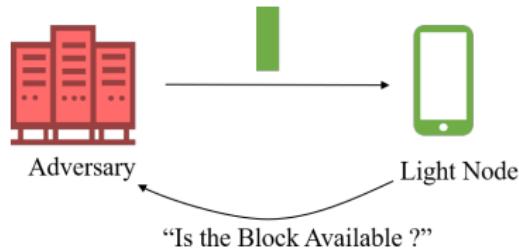


- ▶ **Adversary:** Provides block to Full node but hides invalid portion  
Provides header to Light node
- ▶ **Honest Nodes:** Cannot verify missing transactions → No fraud proof
- ▶ **Light Nodes:** No fraud proof → accept the header.

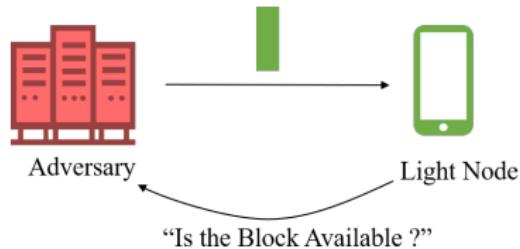
# Ensuring Data Availability



# Ensuring Data Availability

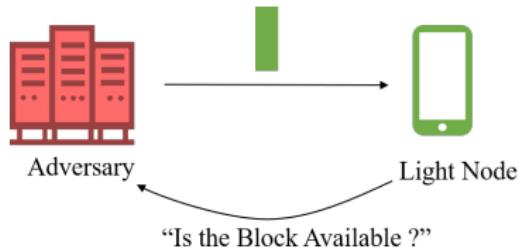


# Ensuring Data Availability



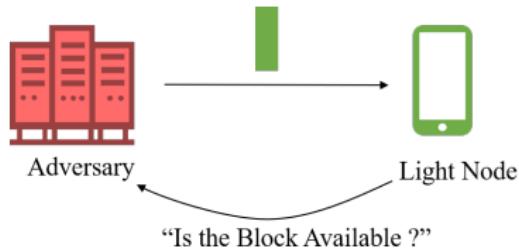
- ▶ Anonymously request/sample few random chunks of the block

# Ensuring Data Availability

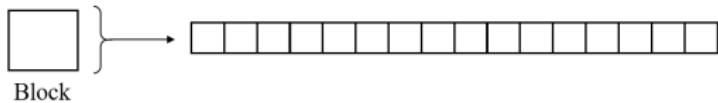


- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion

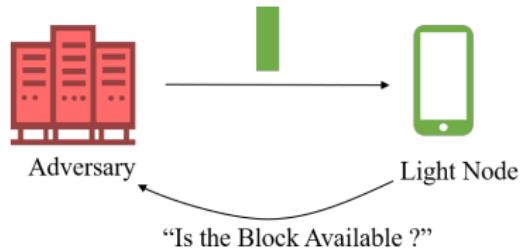
# Ensuring Data Availability



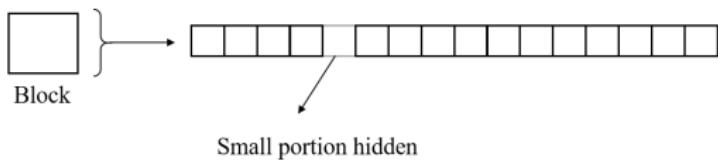
- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion



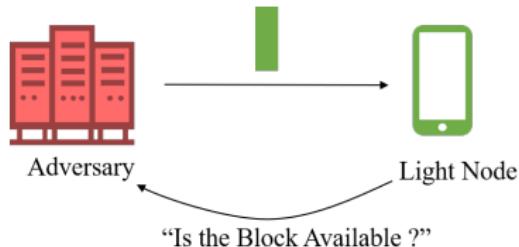
# Ensuring Data Availability



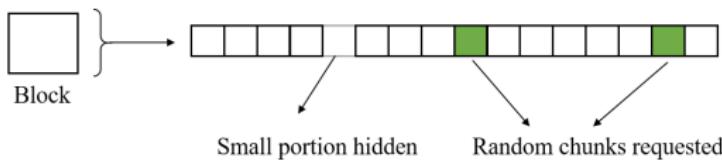
- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion



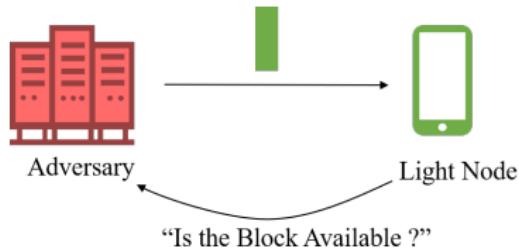
# Ensuring Data Availability



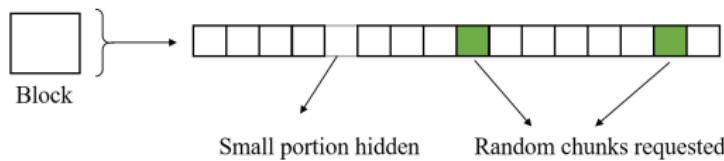
- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion



# Ensuring Data Availability

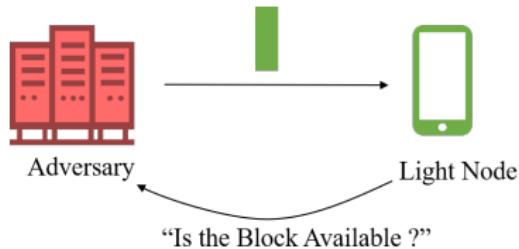


- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion

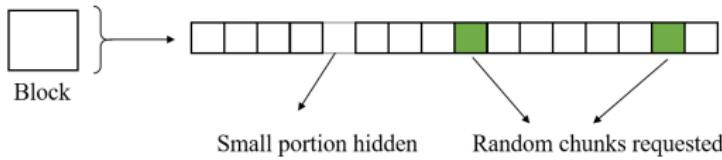


Probability of failure  
using 2 random samples:

# Ensuring Data Availability

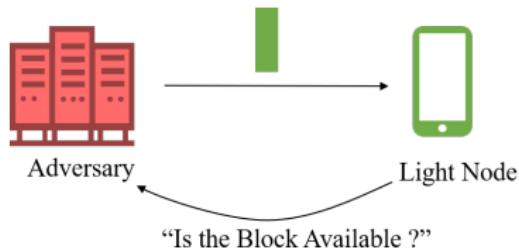


- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion



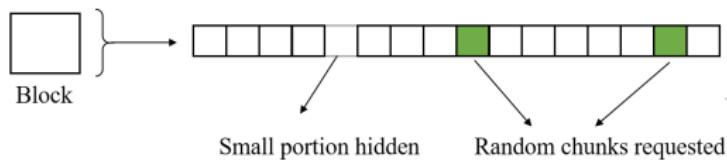
Probability of failure  
using 2 random samples:  
$$\left(1 - \frac{1}{16}\right) \left(1 - \frac{1}{15}\right) = 0.87$$

# Ensuring Data Availability



- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion

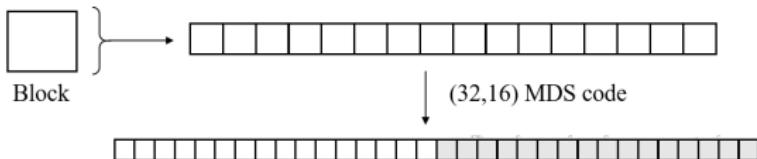
No coding:



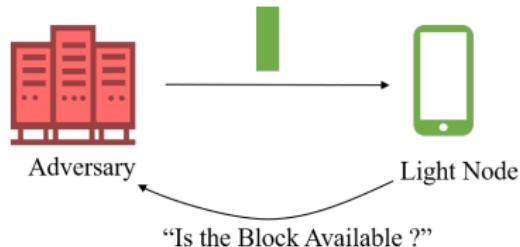
Probability of failure using 2 random samples:

$$\left(1 - \frac{1}{16}\right) \left(1 - \frac{1}{15}\right) = 0.87$$

Erasure coding:

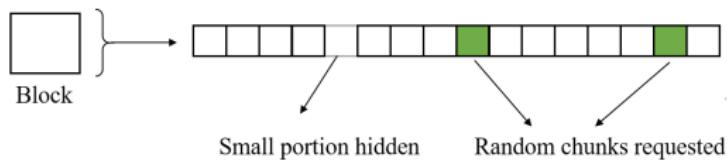


# Ensuring Data Availability



- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion

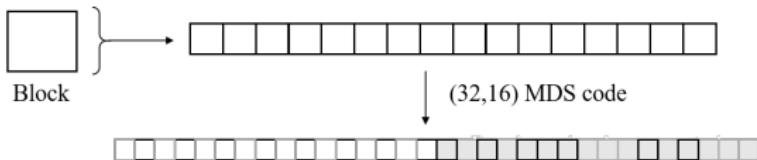
No coding:



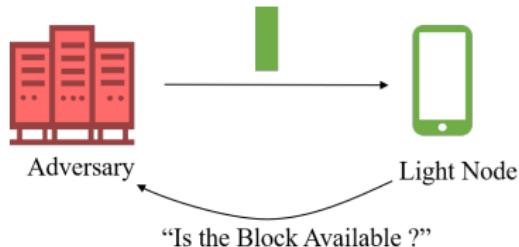
Probability of failure using 2 random samples:

$$\left(1 - \frac{1}{16}\right) \left(1 - \frac{1}{15}\right) = 0.87$$

Erasure coding:

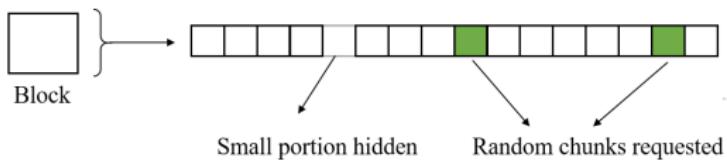


# Ensuring Data Availability



- ▶ Anonymously request/sample few random chunks of the block
- ▶ Adversary can hide a small portion

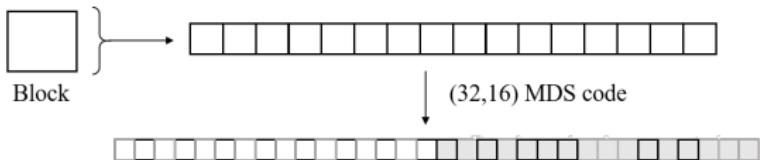
No coding:



Probability of failure using 2 random samples:

$$\left(1 - \frac{1}{16}\right) \left(1 - \frac{1}{15}\right) = 0.87$$

Erasure coding:



Probability of failure using 2 random samples:

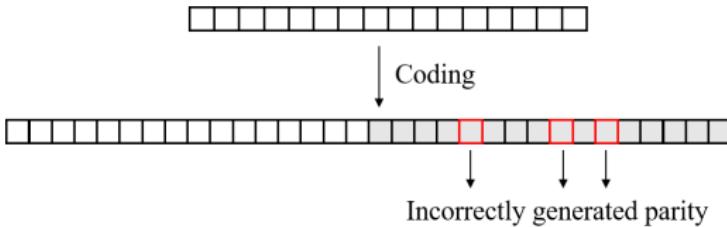
$$\left(1 - \frac{17}{32}\right) \left(1 - \frac{17}{31}\right) = 0.21$$

# Choice of Code Matters

# Choice of Code Matters

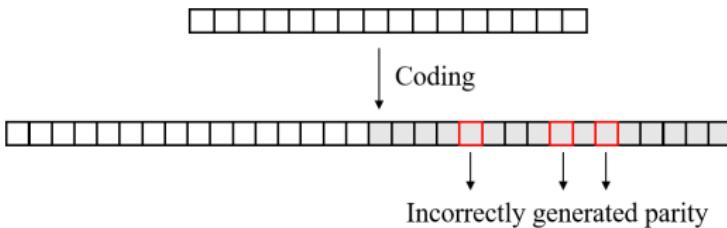
- ▶ Incorrect coding attack:

# Choice of Code Matters



- ▶ Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes

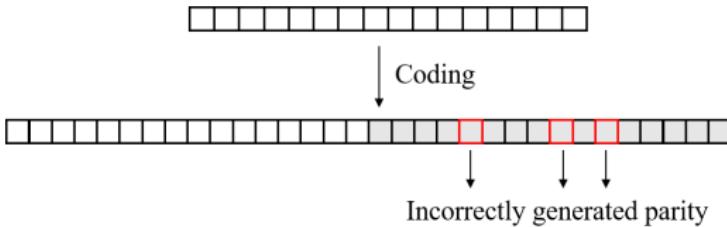
# Choice of Code Matters



## ► Incorrect coding attack:

- Adversary sends incorrectly coded block to Full Nodes
- Honest Full nodes can detect and send incorrect coding proof
- Incorrect coding proof size:  $\mathcal{O}(\text{sparsity of parity check equations})$

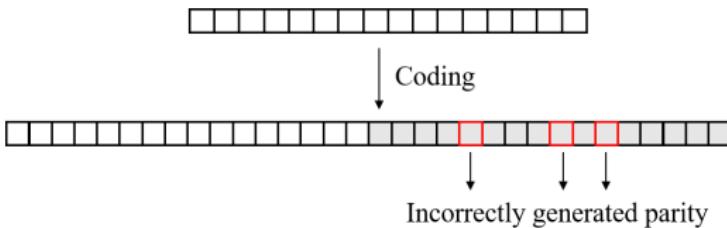
# Choice of Code Matters



## ► Incorrect coding attack:

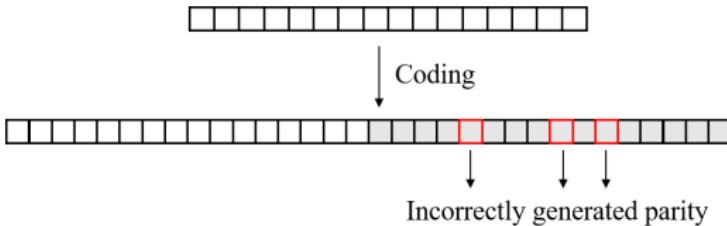
- Adversary sends incorrectly coded block to Full Nodes
- Honest Full nodes can detect and send incorrect coding proof
- Incorrect coding proof size:  $\mathcal{O}(\text{sparsity of parity check equations})$
- MDS codes: proof size =  $\mathcal{O}(\text{block size})$

# Choice of Code Matters



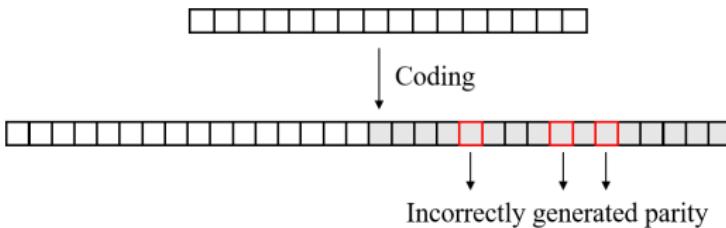
- ▶ Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size:  $\mathcal{O}(\text{sparsity of parity check equations})$
  - **MDS codes: proof size =  $\mathcal{O}(\text{block size})$**
- ▶ Decoding complexity

# Choice of Code Matters



- ▶ Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size:  $\mathcal{O}(\text{sparsity of parity check equations})$
  - **MDS codes: proof size =  $\mathcal{O}(\text{block size})$**
- ▶ Decoding complexity
- ▶ Undecodable ratio  $\delta$

# Choice of Code Matters



- ▶ Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size:  $\mathcal{O}(\text{sparsity of parity check equations})$
  - **MDS codes: proof size =  $\mathcal{O}(\text{block size})$**
- ▶ Decoding complexity
- ▶ Undecodable ratio  $\delta$ 
  - Probability of Light node failure using  $s$  random samples =  $(1 - \delta)^s$

# LDPC Codes: A Strong Contender

LDPC codes:

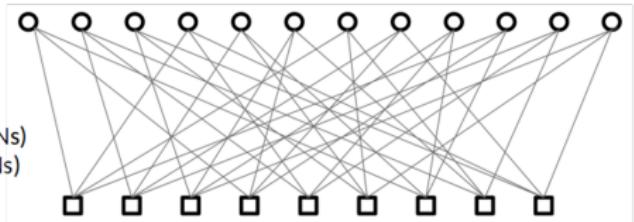
- ▶ Characterized by a sparse parity check matrix

# LDPC Codes: A Strong Contender

LDPC codes:

- ▶ Characterized by a sparse parity check matrix
- ▶ Tanner Graph

circles: variable nodes (VNs)  
squares: check nodes (CNs)



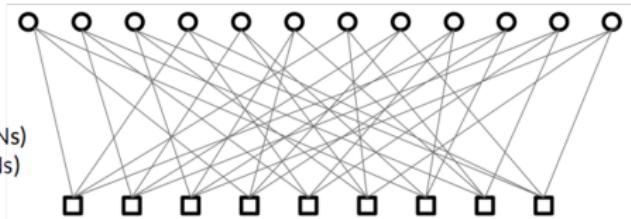
# LDPC Codes: A Strong Contender

LDPC codes:

- ▶ Characterized by a sparse parity check matrix

- ▶ Tanner Graph

circles: variable nodes (VNs)  
squares: check nodes (CNs)

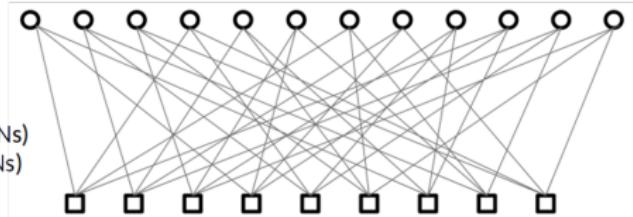


LDPC codes have been shown to be suitable for this application [Yu' 19]

# LDPC Codes: A Strong Contender

LDPC codes:

- ▶ Characterized by a sparse parity check matrix



- ▶ Tanner Graph

LDPC codes have been shown to be suitable for this application [Yu' 19]

- ▶ Small incorrect coding proof size due to small check node degree

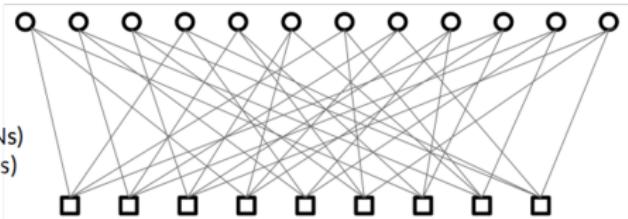
# LDPC Codes: A Strong Contender

LDPC codes:

- ▶ Characterized by a sparse parity check matrix

- ▶ Tanner Graph

circles: variable nodes (VNs)  
squares: check nodes (CNs)



LDPC codes have been shown to be suitable for this application [Yu' 19]

- ▶ Small incorrect coding proof size due to small check node degree
- ▶ Linear decoding in terms of the block size using peeling decoder

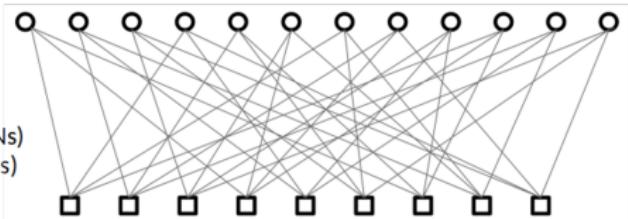
# LDPC Codes: A Strong Contender

LDPC codes:

- ▶ Characterized by a sparse parity check matrix

- ▶ Tanner Graph

circles: variable nodes (VNs)  
squares: check nodes (CNs)

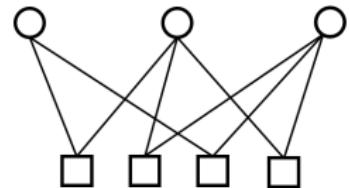


LDPC codes have been shown to be suitable for this application [Yu' 19]

- ▶ Small incorrect coding proof size due to small check node degree
- ▶ Linear decoding in terms of the block size using peeling decoder
- What about the undecodable ratio?

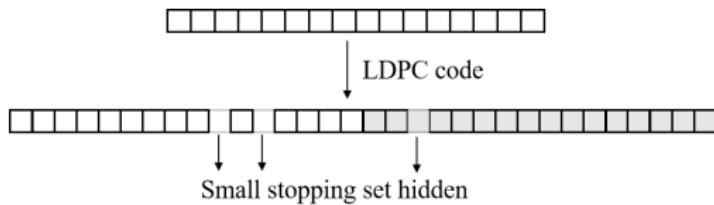
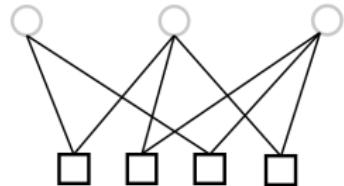
# Challenge with LDPC Codes: Small Stopping Sets

- ▶ Substructure in the Tanner Graph



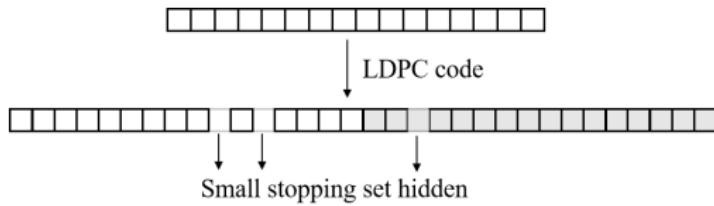
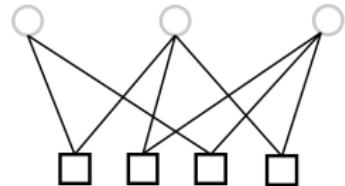
# Challenge with LDPC Codes: Small Stopping Sets

- ▶ Substructure in the Tanner Graph
- ▶ If hidden, prevents peeling decoder from decoding the block → No fraud proof



# Challenge with LDPC Codes: Small Stopping Sets

- ▶ Substructure in the Tanner Graph
- ▶ If hidden, prevents peeling decoder from decoding the block → No fraud proof

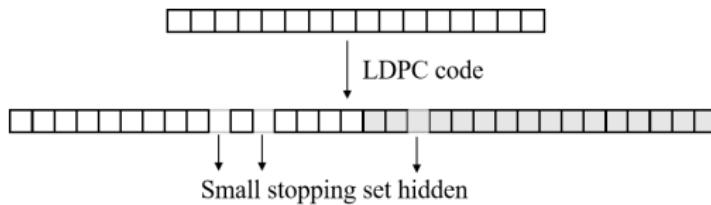
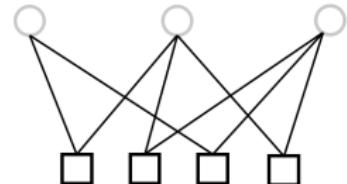


Probability of failure  
using 2 random samples:

$$\left(1 - \frac{3}{32}\right) \left(1 - \frac{3}{31}\right) = 0.81$$

# Challenge with LDPC Codes: Small Stopping Sets

- ▶ Substructure in the Tanner Graph
- ▶ If hidden, prevents peeling decoder from decoding the block → No fraud proof



Probability of failure  
using 2 random samples:

$$\left(1 - \frac{3}{32}\right) \left(1 - \frac{3}{31}\right) = 0.81$$

Our work: Design of specialized LDPC codes with a coupled sampling strategy to achieve a significantly lower probability of failure.

## Motivation: Not all VNs are equal

In this work, we considered an adversary which randomly hides a stopping set of a particular size.

## Motivation: Not all VNs are equal

In this work, we considered an adversary which randomly hides a stopping set of a particular size.

### Lemma

*Of all stopping sets (SSs) of size  $\mu$ , when an adversary randomly hides one of them, and light nodes sample all VNs in the set  $\mathcal{L}$ , then*

# Motivation: Not all VNs are equal

In this work, we considered an adversary which randomly hides a stopping set of a particular size.

## Lemma

*Of all stopping sets (SSs) of size  $\mu$ , when an adversary randomly hides one of them, and light nodes sample all VNs in the set  $\mathcal{L}$ , then*

$$\text{Probability of failure} = 1 - \frac{\text{fraction of SSs}}{\text{of size } \mu \text{ touched by } \mathcal{L}}$$

# Motivation: Not all VNs are equal

In this work, we considered an adversary which randomly hides a stopping set of a particular size.

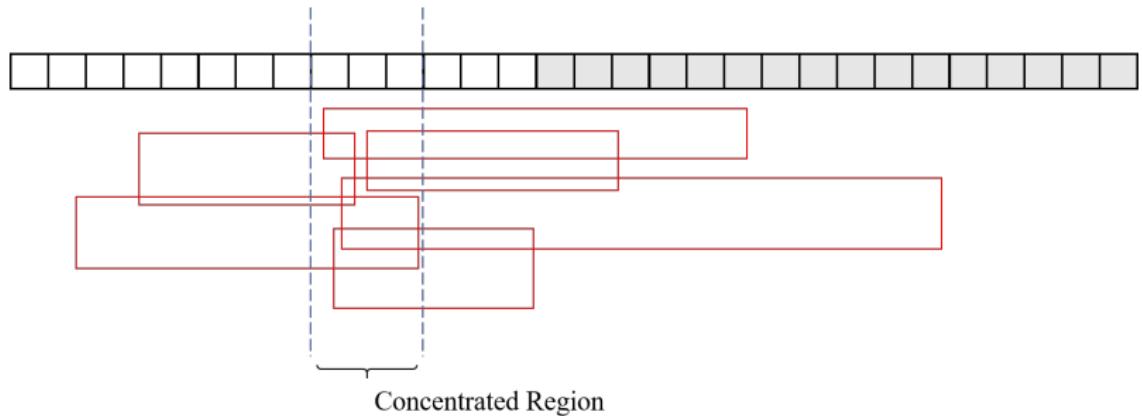
## Lemma

*Of all stopping sets (SSs) of size  $\mu$ , when an adversary randomly hides one of them, and light nodes sample all VNs in the set  $\mathcal{L}$ , then*

$$\text{Probability of failure} = 1 - \frac{\text{fraction of SSs}}{\text{of size } \mu \text{ touched by } \mathcal{L}}$$

- ▶ Selecting a set  $\mathcal{L}$  of VNs which touches large no. of SSs  
→ Prob. of failure ↓

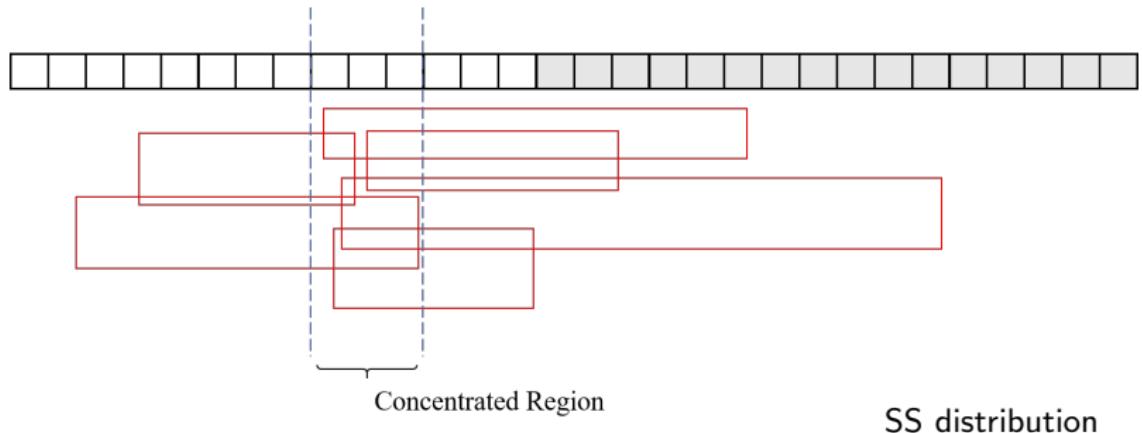
# Concentrated Stopping Set Design



Code Design Idea:

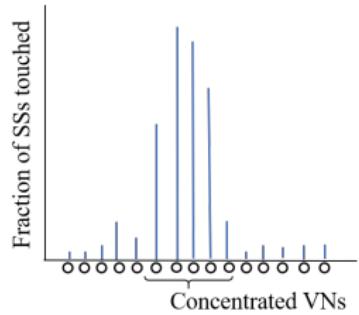
- ▶ Concentrate stopping sets to a small section of VNs

# Concentrated Stopping Set Design

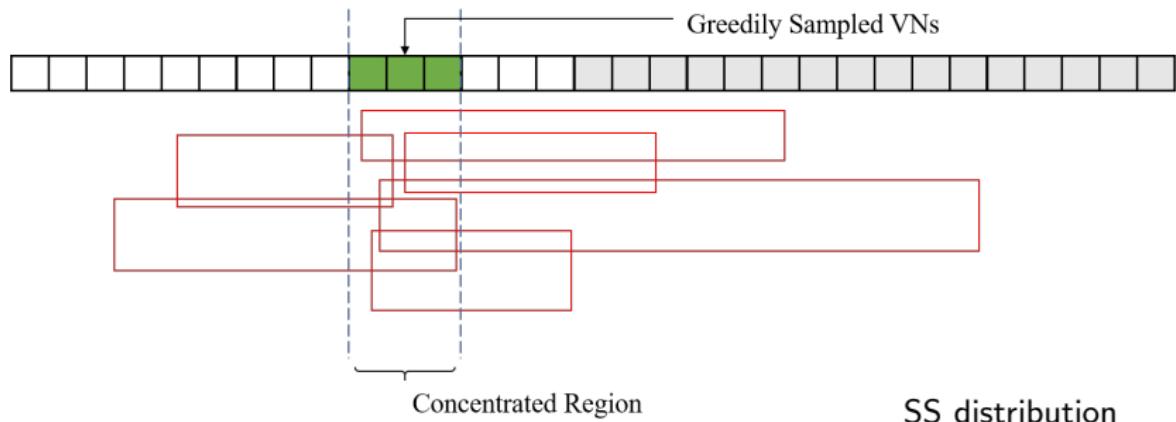


Code Design Idea:

- ▶ Concentrate stopping sets to a small section of VNs

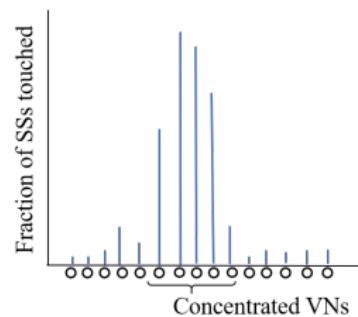


# Concentrated Stopping Set Design

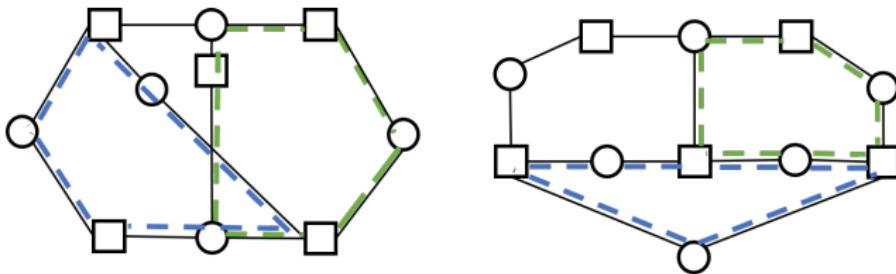


Code Design Idea:

- ▶ Concentrate stopping sets to a small section of VNs
- ▶ Greedily Sample this small section of VNs

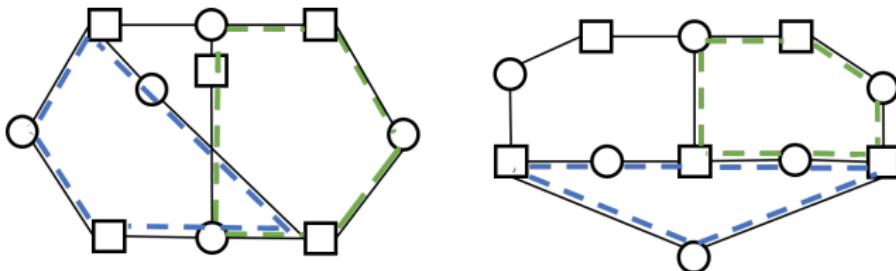


# How to Concentrate Stopping Sets?



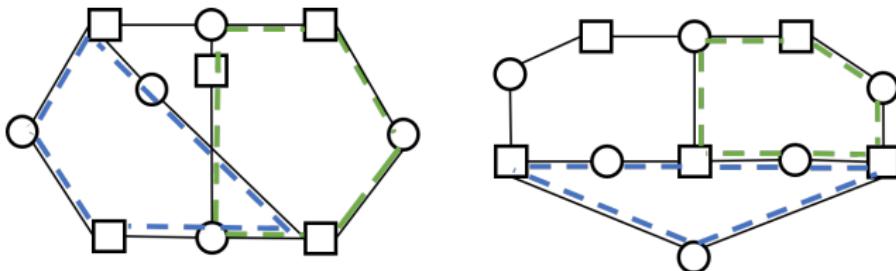
- ▶ When there are no degree 1 VNs, stopping sets are either cycles or interconnection of cycles [Tian '03]

# How to Concentrate Stopping Sets?



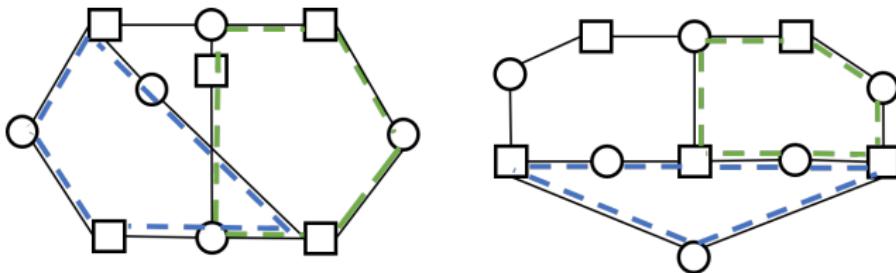
- ▶ When there are no degree 1 VNs, stopping sets are either cycles or interconnection of cycles [Tian '03]
- ▶ Concentrating cycles  $\implies$  Concentrating stopping sets

# How to Concentrate Stopping Sets?



- ▶ When there are no degree 1 VNs, stopping sets are either cycles or interconnection of cycles [Tian '03]
- ▶ Concentrating cycles  $\implies$  Concentrating stopping sets
- How to design codes with concentrated cycles?

# How to Concentrate Stopping Sets?

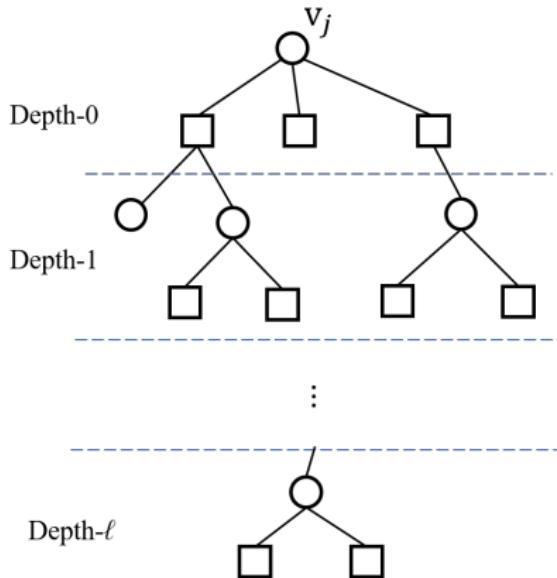


- ▶ When there are no degree 1 VNs, stopping sets are either cycles or interconnection of cycles [Tian '03]
- ▶ Concentrating cycles  $\implies$  Concentrating stopping sets
- How to design codes with concentrated cycles?  
We do so by modifying the well-known Progressive Edge Growth (PEG) algorithm

# PEG Algorithm

- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

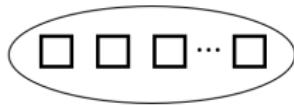
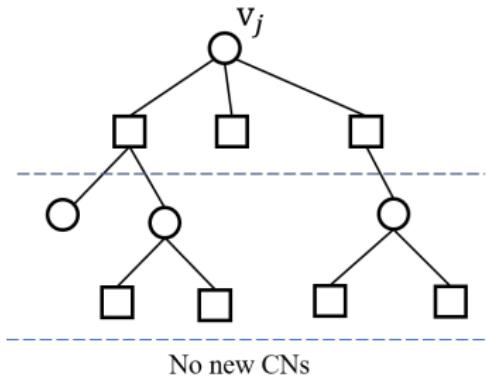
# PEG Algorithm



- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

**For** each VN  $v_j$   
Expand Tanner Graph in a BFS fashion

# PEG Algorithm

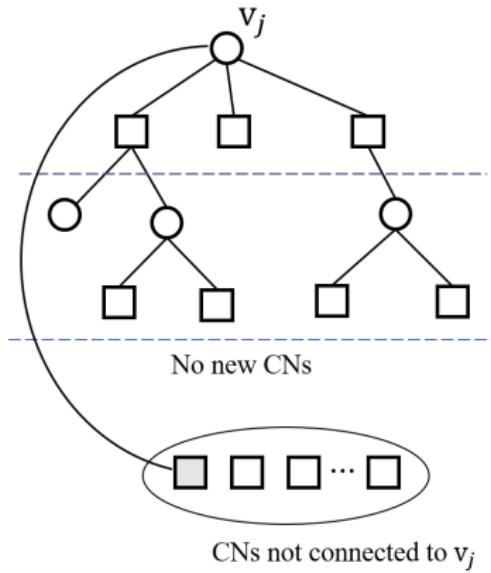


CNs not connected to  $v_j$

- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

**For** each VN  $v_j$   
Expand Tanner Graph in a BFS fashion  
**If**  $\exists$  CNs not connected to  $v_j$

# PEG Algorithm

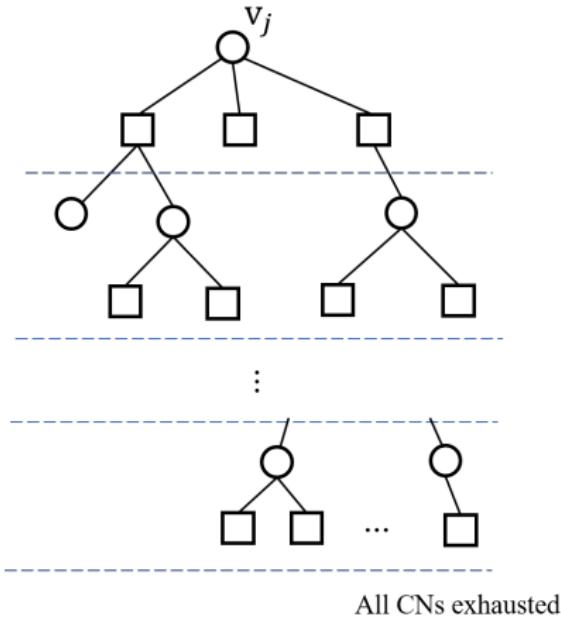


- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

**For** each VN  $v_j$   
Expand Tanner Graph in a BFS fashion  
**If**  $\exists$  CNs not connected to  $v_j$ 

- Select a CN with min degree not connected to  $v_j$

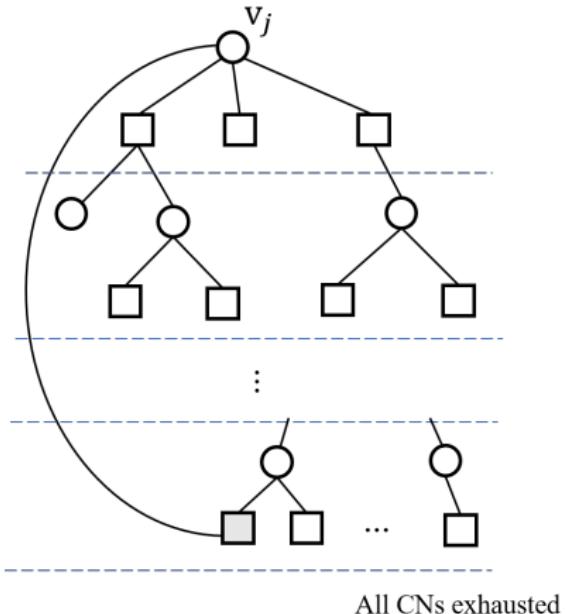
# PEG Algorithm



- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

```
For each VN  $v_j$ 
  Expand Tanner Graph in a BFS fashion
  If  $\exists$  CNs not connected to  $v_j$ 
    • Select a CN with min degree not
      connected to  $v_j$ 
  Else
```

# PEG Algorithm

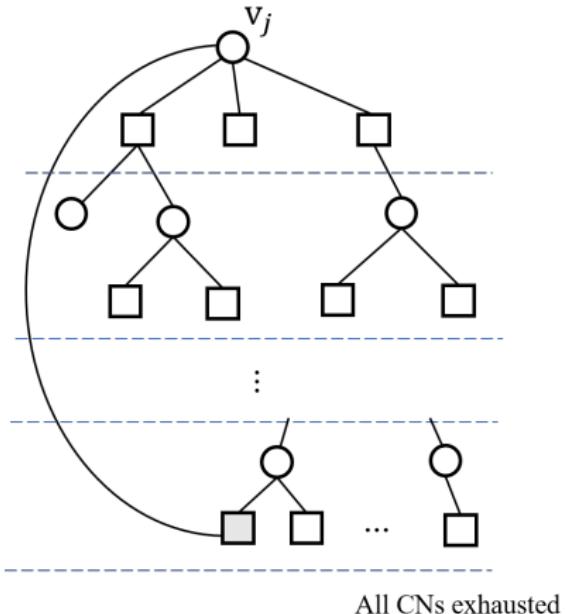


- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

```

For each VN  $v_j$ 
    Expand Tanner Graph in a BFS fashion
    If  $\exists$  CNs not connected to  $v_j$ 
        • Select a CN with min degree not
          connected to  $v_j$ 
    Else
        • Find CNs most distant to  $v_j$ 
        • Select one with minimum degree
  
```

# PEG Algorithm



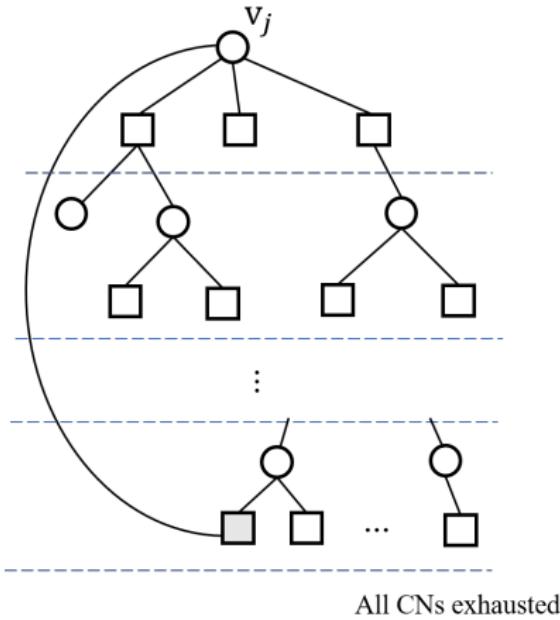
- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

```

For each VN  $v_j$ 
    Expand Tanner Graph in a BFS fashion
    If  $\exists$  CNs not connected to  $v_j$ 
        • Select a CN with min degree not
          connected to  $v_j$ 
    Else
        • Find CNs most distant to  $v_j$ 
        • Select one with minimum degree
    New cycles created

```

# PEG Algorithm



- ▶ Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

```

For each VN  $v_j$ 
  Expand Tanner Graph in a BFS fashion
  If  $\exists$  CNs not connected to  $v_j$ 
    • Select a CN with min degree not
      connected to  $v_j$ 
  Else
    • Find CNs most distant to  $v_j$ 
    • Select one with minimum degree
      New cycles created

```

We modify the CN selection criteria in green to concentrate cycles

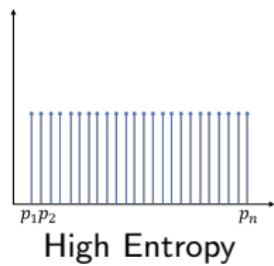
## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

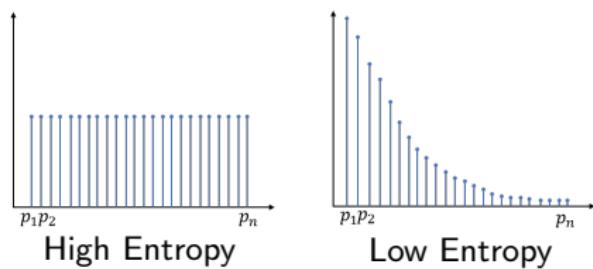
- ▶ Uniform distributions have high entropy



## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

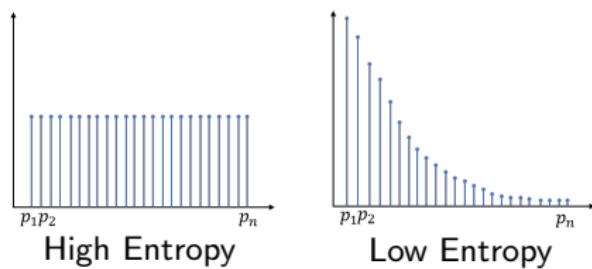
- ▶ Uniform distributions have high entropy
- ▶ Concentrated distributions have low entropy



## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

- ▶ Uniform distributions have high entropy
- ▶ Concentrated distributions have low entropy

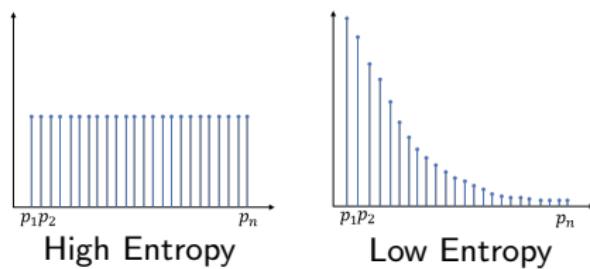


We want the cycle distributions to be concentrated

## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

- ▶ Uniform distributions have high entropy
- ▶ Concentrated distributions have low entropy

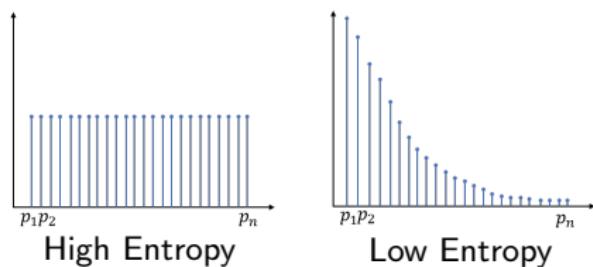


We want the cycle distributions to be concentrated  
→ Select CNs such that the entropy of the cycle distribution is minimized

## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

- ▶ Uniform distributions have high entropy
- ▶ Concentrated distributions have low entropy



### EC (Entropy Constrained)-PEG Algorithm

**For each VN  $v_j$**

Expand Tanner Graph in a BFS fashion

**If**  $\exists$  CNs not connected to  $v_j$

- select a CN with min degree not connected to  $v_j$

**Else** New cycles created

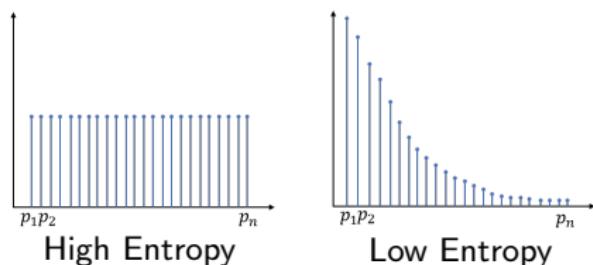
We want the cycle distributions to be concentrated

→ Select CNs such that the entropy of the cycle distribution is minimized

## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

- ▶ Uniform distributions have high entropy
- ▶ Concentrated distributions have low entropy



### EC (Entropy Constrained)-PEG Algorithm

**For each VN  $v_j$**

Expand Tanner Graph in a BFS fashion

**If**  $\exists$  CNs not connected to  $v_j$

- select a CN with min degree not connected to  $v_j$

**Else New cycles created**

- Find CNs most distant to  $v_j$

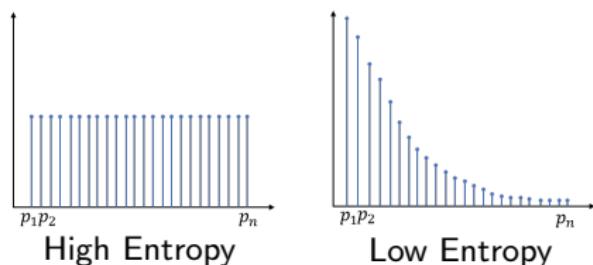
We want the cycle distributions to be concentrated

→ Select CNs such that the entropy of the cycle distribution is minimized

# Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

- ▶ Uniform distributions have high entropy
- ▶ Concentrated distributions have low entropy



## EC (Entropy Constrained)-PEG Algorithm

**For each VN  $v_j$**

Expand Tanner Graph in a BFS fashion

**If**  $\exists$  CNs not connected to  $v_j$

- select a CN with min degree not connected to  $v_j$

**Else New cycles created**

- Find CNs most distant to  $v_j$
- Select CN that results in minimum entropy of resultant cycle distribution

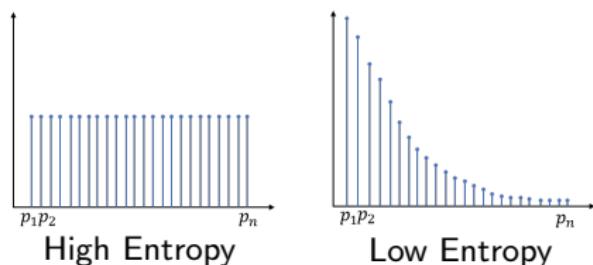
We want the cycle distributions to be concentrated

→ Select CNs such that the entropy of the cycle distribution is minimized

## Using Entropy to Concentrate Cycles

For distribution  $p = (p_1, p_2, \dots, p_n)$ , Entropy  $\mathcal{H}(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i}$

- ▶ Uniform distributions have high entropy
- ▶ Concentrated distributions have low entropy



### EC (Entropy Constrained)-PEG Algorithm

**For each VN  $v_j$**

Expand Tanner Graph in a BFS fashion

**If**  $\exists$  CNs not connected to  $v_j$

- select a CN with min degree not connected to  $v_j$

**Else New cycles created**

- Find CNs most distant to  $v_j$
- Select CN that results in minimum entropy of resultant cycle distribution
- Update cycle distribution

We want the cycle distributions to be concentrated

→ Select CNs such that the entropy of the cycle distribution is minimized

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

VNs  $(v_1, v_2, \dots, v_n)$

# EC-PEG Algorithm

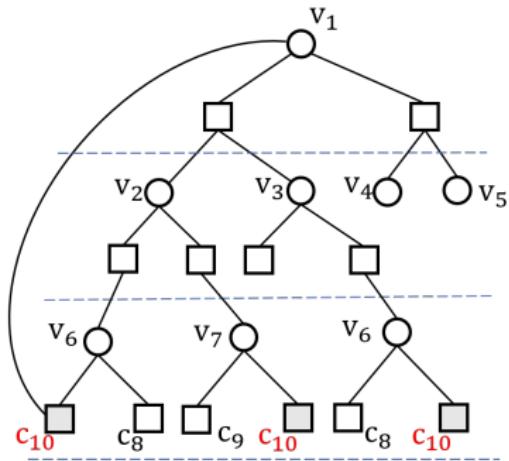
- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

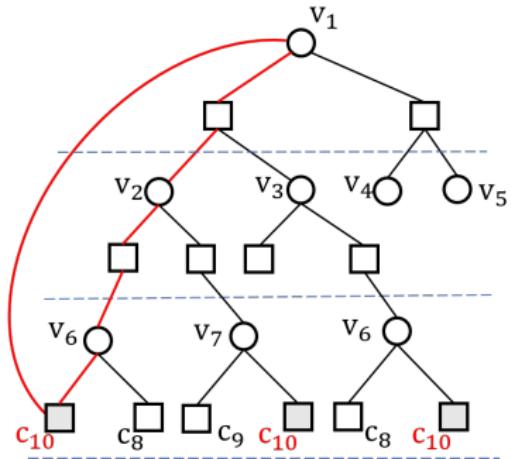


VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

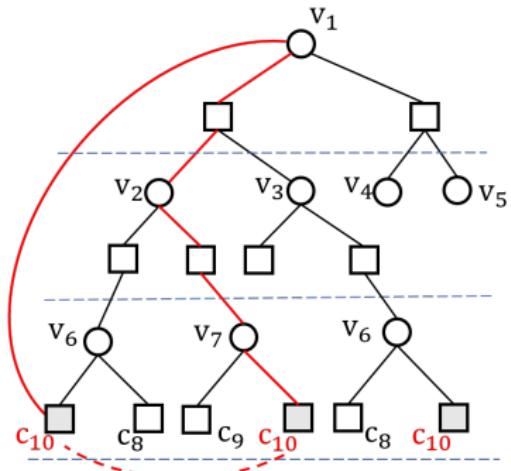


VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

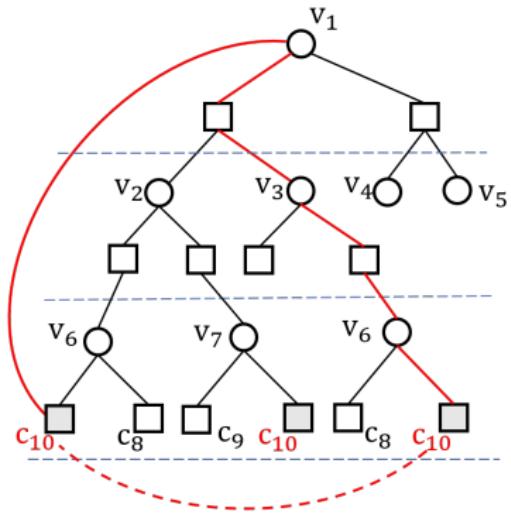


VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

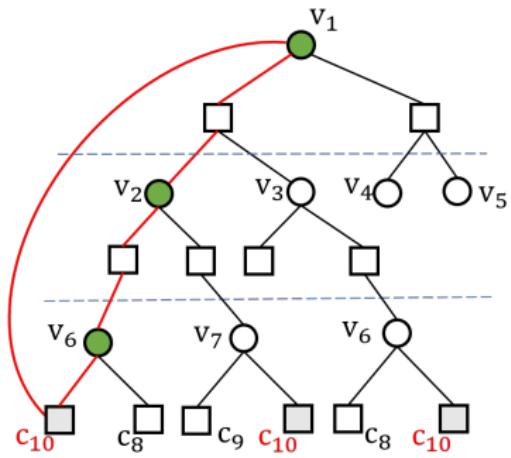


VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN



VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$



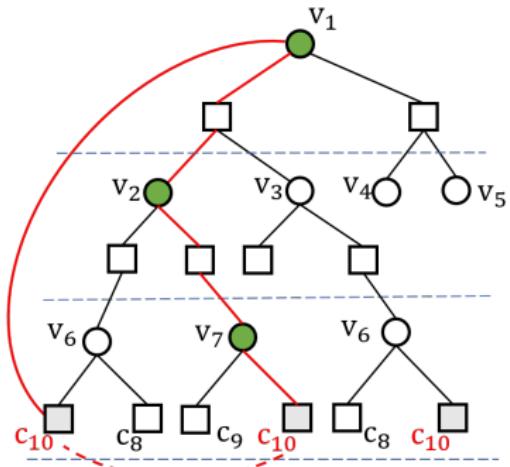
$$\lambda_1^{(6)} = \lambda_1^{(6)} + 1$$

$$\lambda_2^{(6)} = \lambda_2^{(6)} + 1$$

$$\lambda_6^{(6)} = \lambda_6^{(6)} + 1$$

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN



VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$



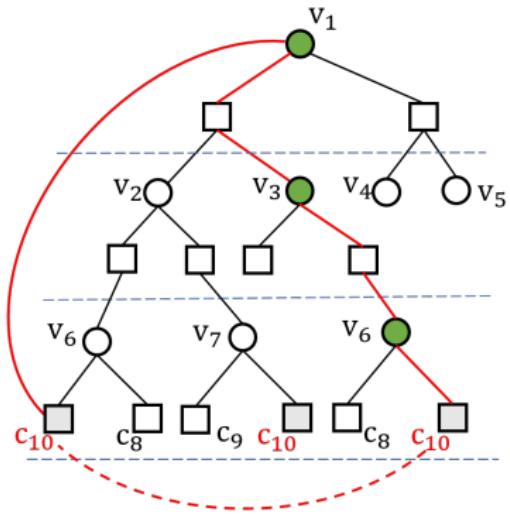
$$\lambda_1^{(6)} = \lambda_1^{(6)} + 1$$

$$\lambda_2^{(6)} = \lambda_2^{(6)} + 1$$

$$\lambda_7^{(6)} = \lambda_7^{(6)} + 1$$

# EC-PEG Algorithm

- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN



VNs  $(v_1, v_2, \dots, v_n)$

- ▶  $\lambda_i^{(g)} :=$  No. of cycles of length  $g$  that  $v_i$  is a part of,  $g = 4, 6, 8$

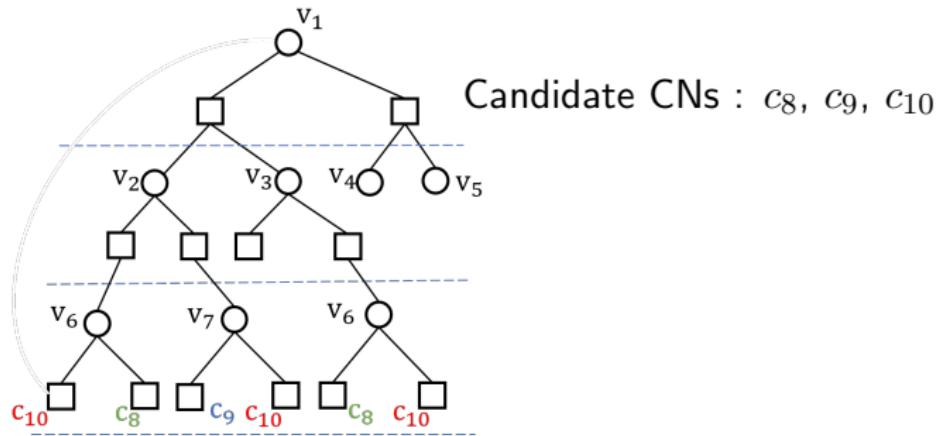


$$\lambda_1^{(6)} = \lambda_1^{(6)} + 1$$

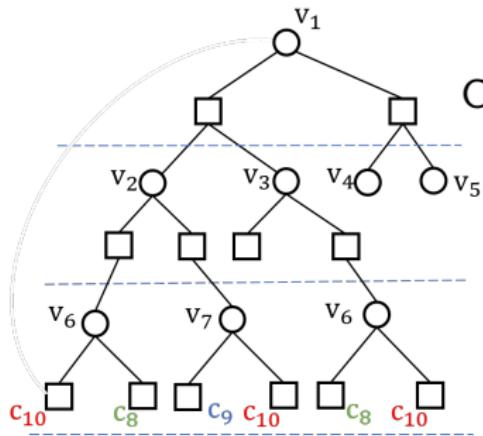
$$\lambda_3^{(6)} = \lambda_3^{(6)} + 1$$

$$\lambda_6^{(6)} = \lambda_6^{(6)} + 1$$

# EC-PEG Algorithm: CN Selection Procedure



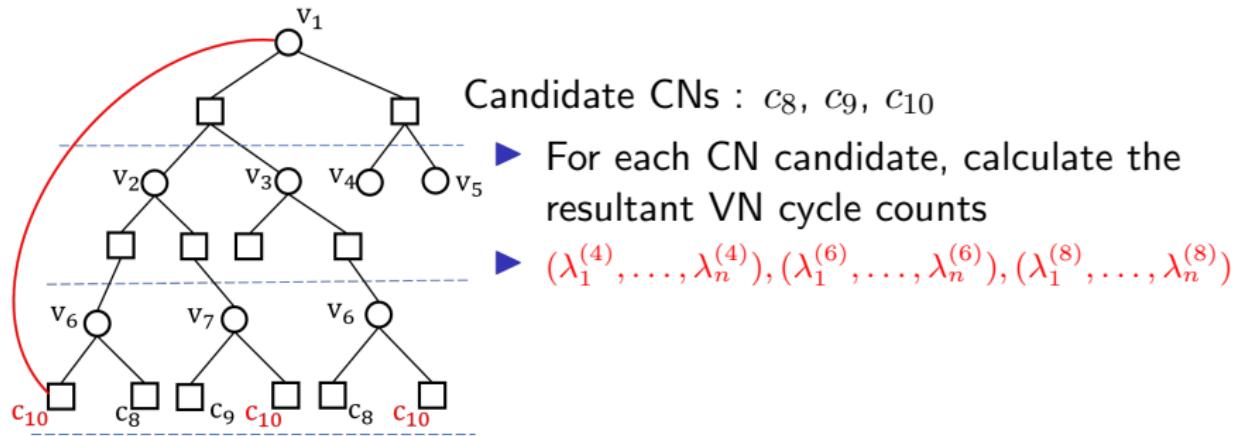
# EC-PEG Algorithm: CN Selection Procedure



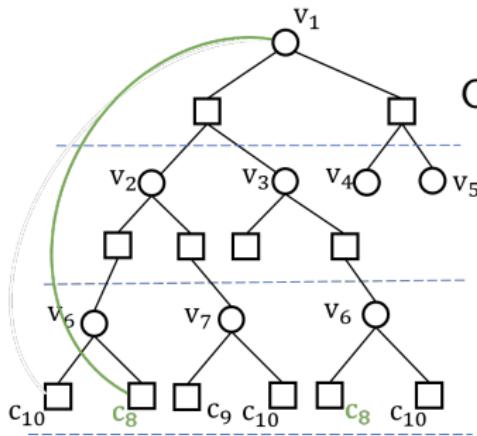
Candidate CNs :  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, calculate the resultant VN cycle counts

# EC-PEG Algorithm: CN Selection Procedure



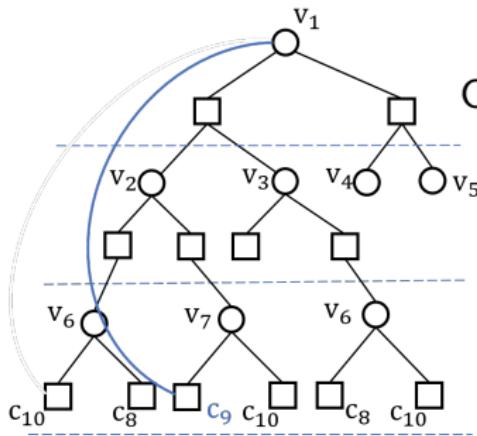
# EC-PEG Algorithm: CN Selection Procedure



Candidate CNs :  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, calculate the resultant VN cycle counts
- ▶  $(\lambda_1^{(4)}, \dots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \dots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \dots, \lambda_n^{(8)})$
- ▶  $(\lambda_1^{(4)}, \dots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \dots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \dots, \lambda_n^{(8)})$

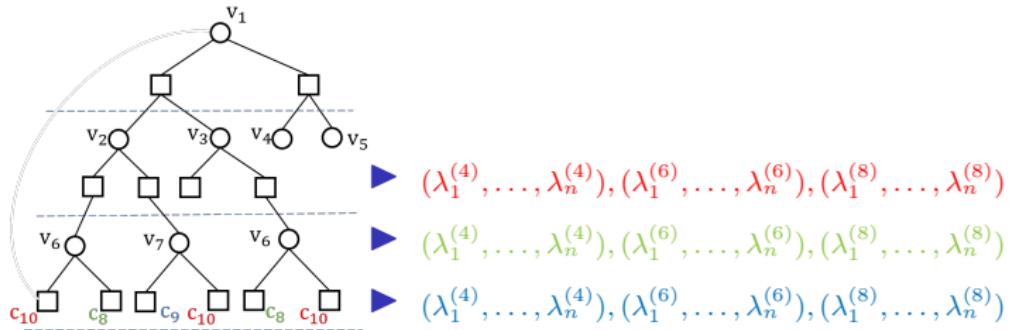
# EC-PEG Algorithm: CN Selection Procedure



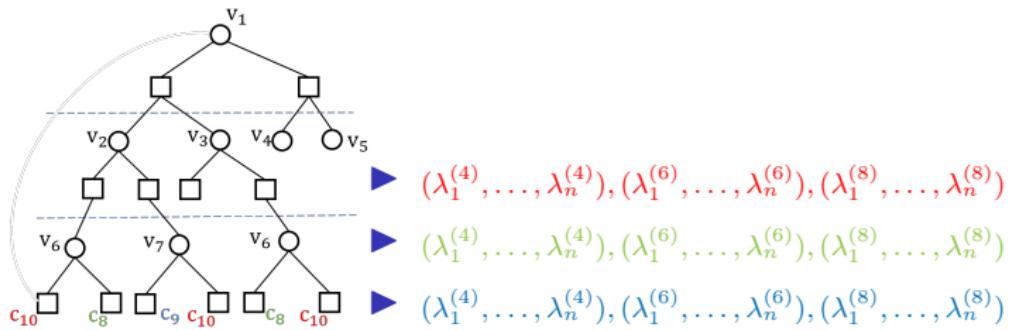
Candidate CNs :  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, calculate the resultant VN cycle counts
- ▶  $(\lambda_1^{(4)}, \dots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \dots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \dots, \lambda_n^{(8)})$
- ▶  $(\lambda_1^{(4)}, \dots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \dots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \dots, \lambda_n^{(8)})$
- ▶  $(\lambda_1^{(4)}, \dots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \dots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \dots, \lambda_n^{(8)})$

# EC-PEG algorithm: CN selection Procedure

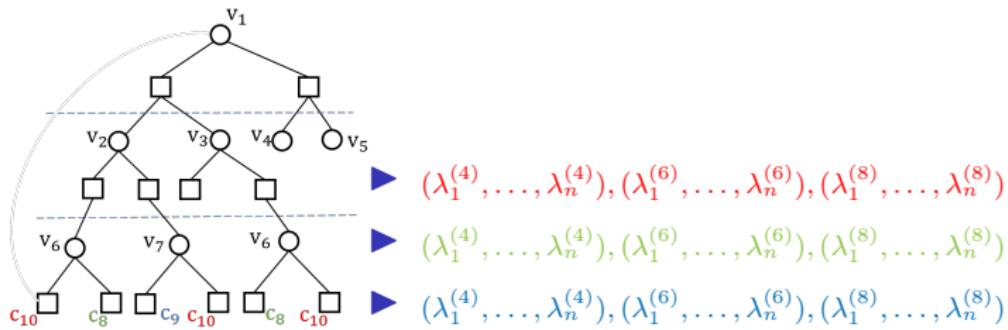


# EC-PEG algorithm: CN selection Procedure



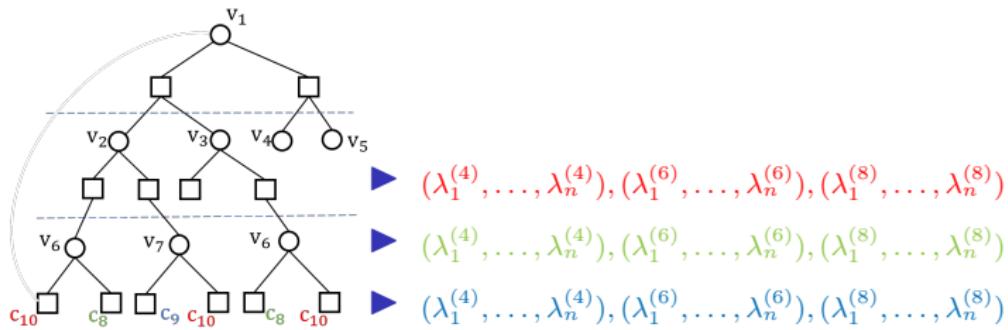
$$\underbrace{(\lambda_1^{(g)}, \dots, \lambda_n^{(g)})}_{\text{cycle counts}}$$

# EC-PEG algorithm: CN selection Procedure



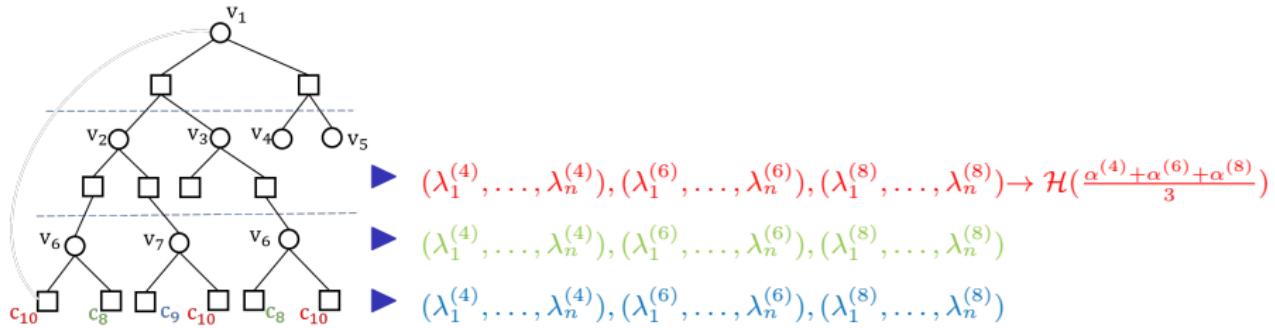
$$\underbrace{(\lambda_1^{(g)}, \dots, \lambda_n^{(g)})}_{\text{cycle counts}} \rightarrow \underbrace{\left( \frac{\lambda_1^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}}, \dots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}} \right)}_{\text{normalized counts}} := \alpha^{(g)}$$

# EC-PEG algorithm: CN selection Procedure



$$\underbrace{(\lambda_1^{(g)}, \dots, \lambda_n^{(g)})}_{\text{cycle counts}} \rightarrow \underbrace{\left( \frac{\lambda_1^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}}, \dots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}} \right)}_{\text{normalized counts}} := \alpha^{(g)} \rightarrow \mathcal{H}\left(\underbrace{\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}}_{\text{entropy of combined counts}}\right)$$

# EC-PEG algorithm: CN selection Procedure



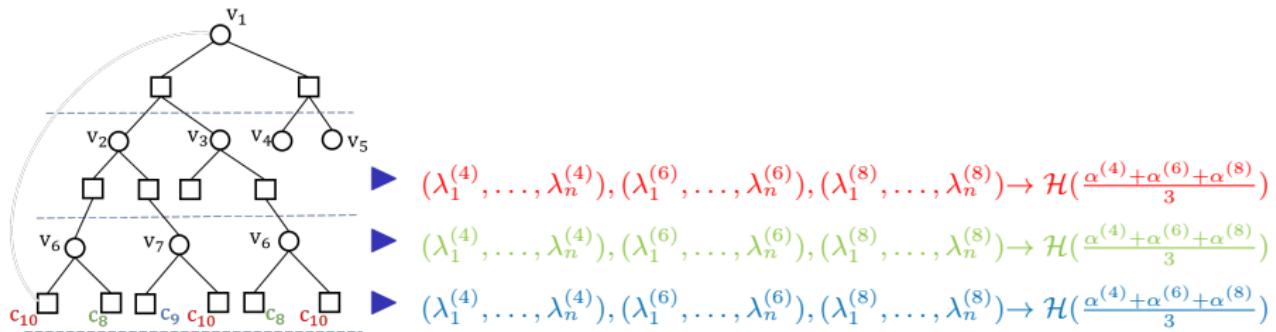
$$\underbrace{(\lambda_1^{(g)}, \dots, \lambda_n^{(g)})}_{\text{cycle counts}} \rightarrow \underbrace{\left( \frac{\lambda_1^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}}, \dots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}} \right)}_{\text{normalized counts}} := \alpha^{(g)} \rightarrow \underbrace{\mathcal{H}\left(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}\right)}_{\text{entropy of combined counts}}$$

# EC-PEG algorithm: CN selection Procedure



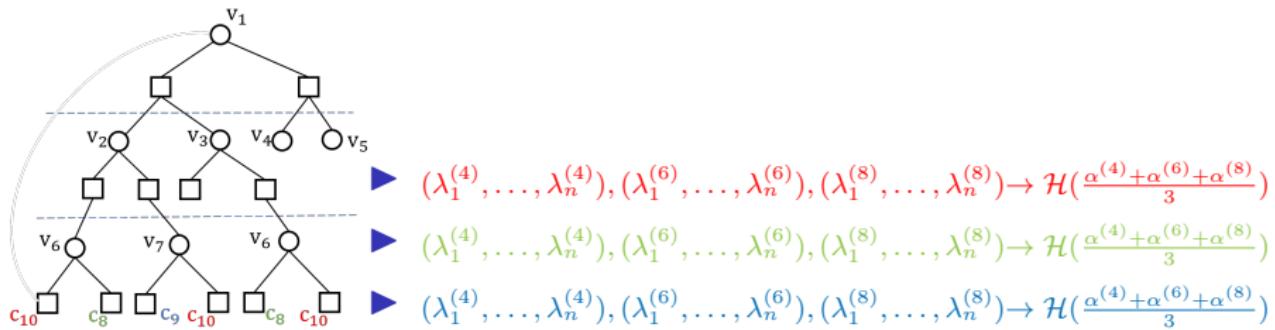
$$\underbrace{(\lambda_1^{(g)}, \dots, \lambda_n^{(g)})}_{\text{cycle counts}} \rightarrow \underbrace{\left( \frac{\lambda_1^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}}, \dots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}} \right)}_{\text{normalized counts}} := \alpha^{(g)} \rightarrow \underbrace{\mathcal{H}\left(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}\right)}_{\text{entropy of combined counts}}$$

# EC-PEG algorithm: CN selection Procedure



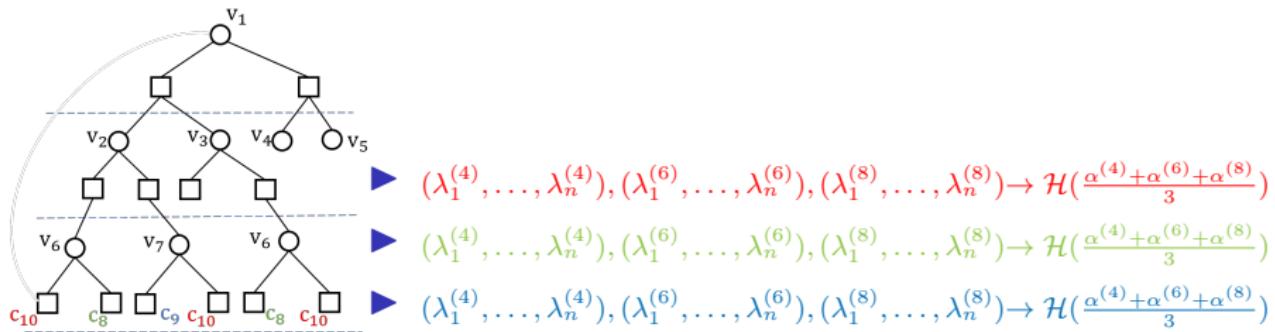
$$\underbrace{(\lambda_1^{(g)}, \dots, \lambda_n^{(g)})}_{\text{cycle counts}} \rightarrow \underbrace{\left( \frac{\lambda_1^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}}, \dots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^n \lambda_i^{(g)}} \right)}_{\text{normalized counts}} := \alpha^{(g)} \rightarrow \underbrace{\mathcal{H}\left(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}\right)}_{\text{entropy of combined counts}}$$

# EC-PEG algorithm: CN selection Procedure



CN selection procedure:

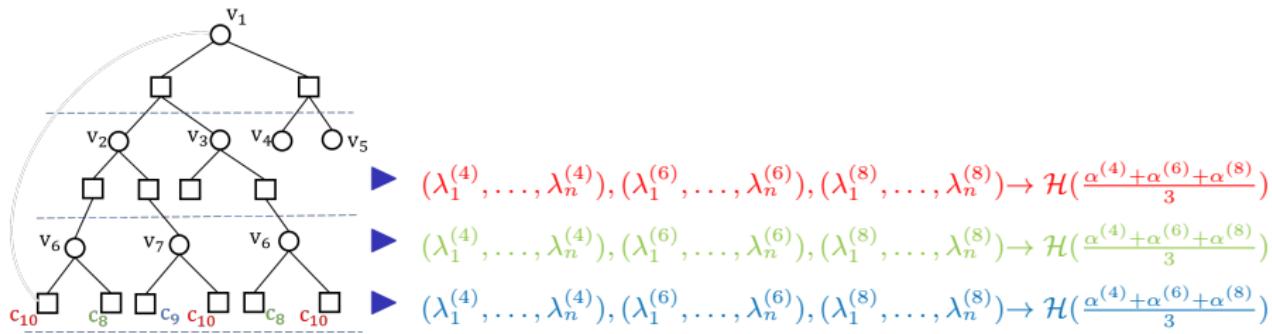
# EC-PEG algorithm: CN selection Procedure



CN selection procedure:

Select CN that results in minimum  $\mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3})$

# EC-PEG algorithm: CN selection Procedure



CN selection procedure:

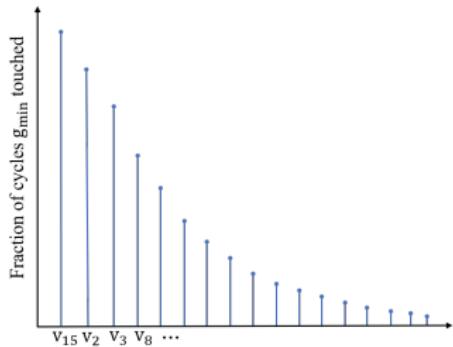
Select CN that results in minimum  $H(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3})$

Note:

- Minimizing the entropy of joint cycle counts ensures that all cycle distributions are concentrated towards the same set of VNs

# Sampling Strategy

- ▶ Our sampling strategy greedily samples VNs that are part of a large number of cycles



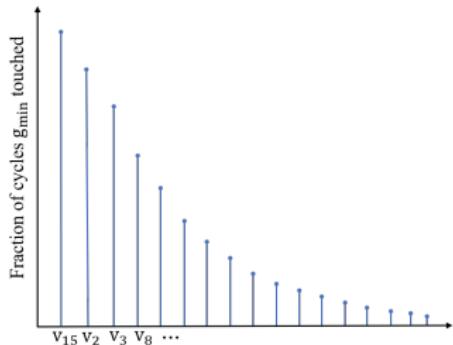
$g = \text{smallest cycle length in Tanner Graph } \mathcal{G}$

**While** sample set size  $< s$

- $v = \text{VN that is part of largest no. of cycles of length } g \text{ in } \mathcal{G}$
- sample set = sample set  $\cup v$
- remove  $v$  and all incident edges from  $\mathcal{G}$

# Sampling Strategy

- ▶ Our sampling strategy greedily samples VNs that are part of a large number of cycles



$g = \text{smallest cycle length in Tanner Graph } \mathcal{G}$

**While** sample set size  $< s$

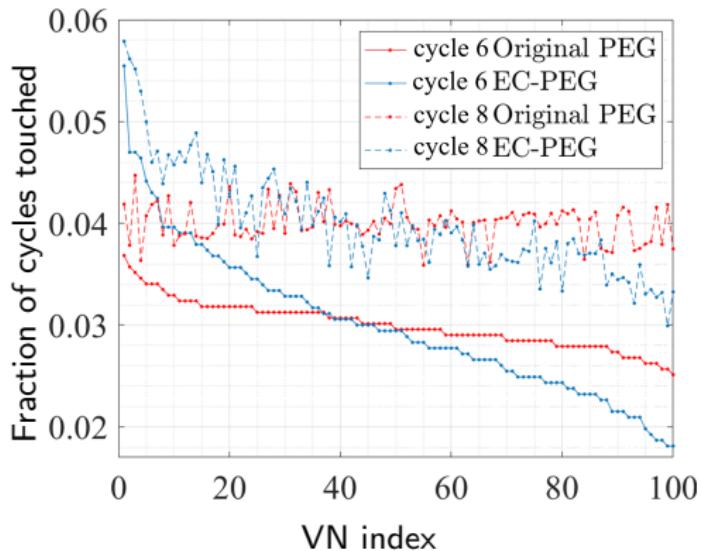
- $v = \text{VN that is part of largest no. of cycles of length } g \text{ in } \mathcal{G}$
- sample set = sample set  $\cup v$
- remove  $v$  and all incident edges from  $\mathcal{G}$
- If  $\nexists$  cycles of length  $g$  in  $\mathcal{G}$ 
  - $g = g + 2$

## Simulation Results

- ▶ Code parameters: Code length = 100, VN degree = 4, Rate =  $\frac{1}{2}$ , girth = 6.

## Simulation Results

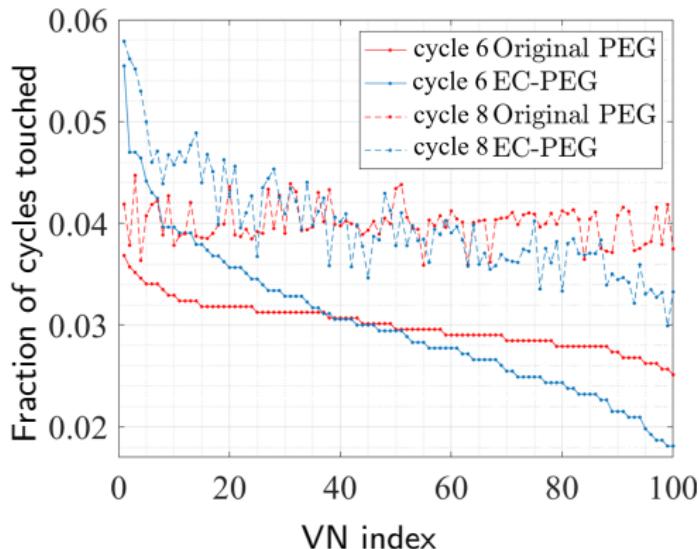
- ▶ Code parameters: Code length = 100, VN degree = 4, Rate =  $\frac{1}{2}$ , girth = 6.



- ▶ VN indices arranged in decreasing order of cycle 6 fractions

## Simulation Results

- ▶ Code parameters: Code length = 100, VN degree = 4, Rate =  $\frac{1}{2}$ , girth = 6.



- ▶ VN indices arranged in decreasing order of cycle 6 fractions
- ▶ Cycle 6 and cycle 8 concentrated towards same set of VNs

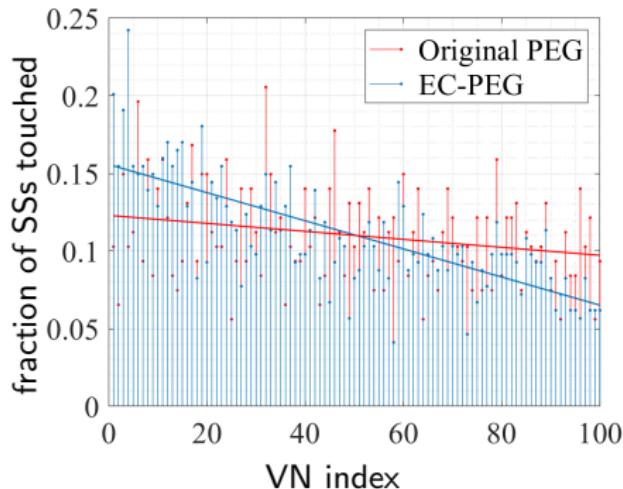
# Simulation Results

Fraction of SSs of size 11, 12 touched by different VNs

# Simulation Results

Fraction of SSs of size 11, 12 touched by different VNs

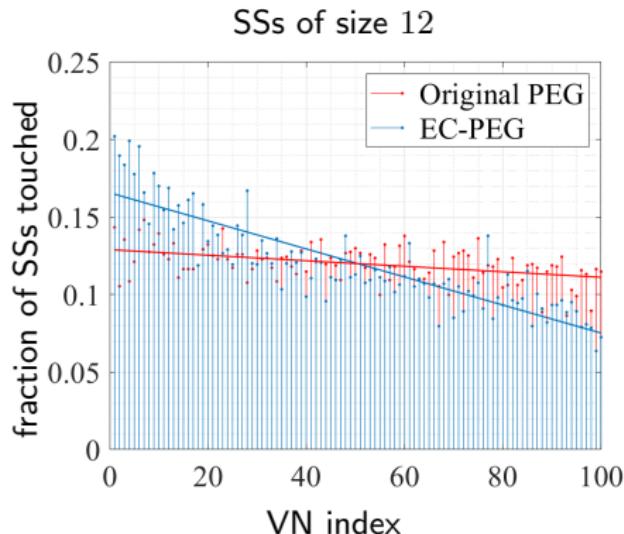
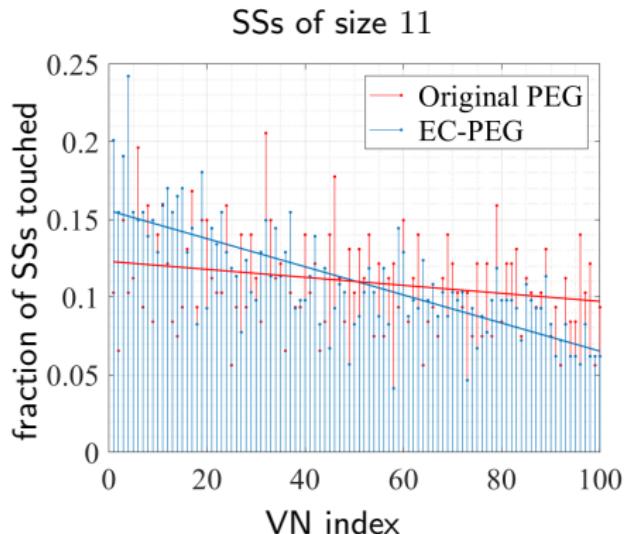
SSs of size 11



- ▶ VN indices arranged in decreasing order of cycle 6 fractions

# Simulation Results

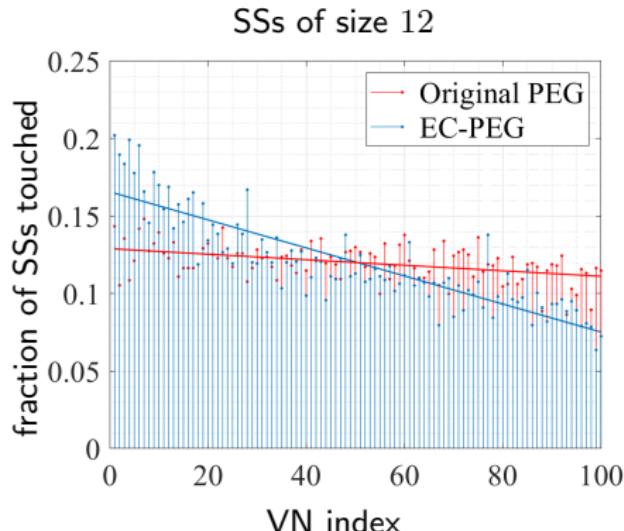
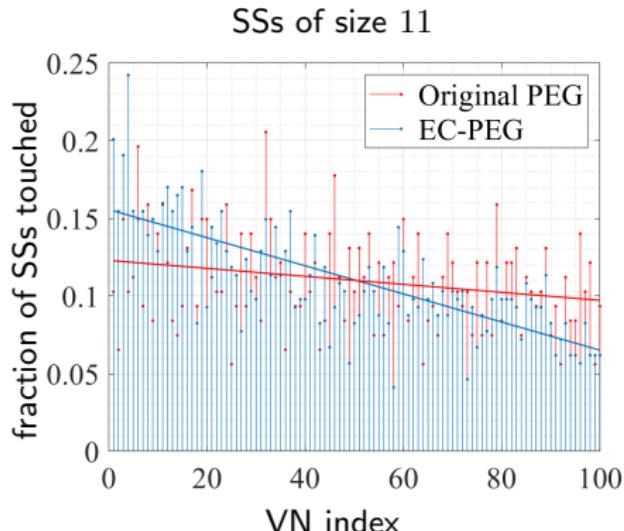
Fraction of SSs of size 11, 12 touched by different VNs



- VN indices arranged in decreasing order of cycle 6 fractions

# Simulation Results

Fraction of SSs of size 11, 12 touched by different VNs



- ▶ VN indices arranged in decreasing order of cycle 6 fractions
- ▶ SSs are concentrated towards the same set of VNs as the cycles

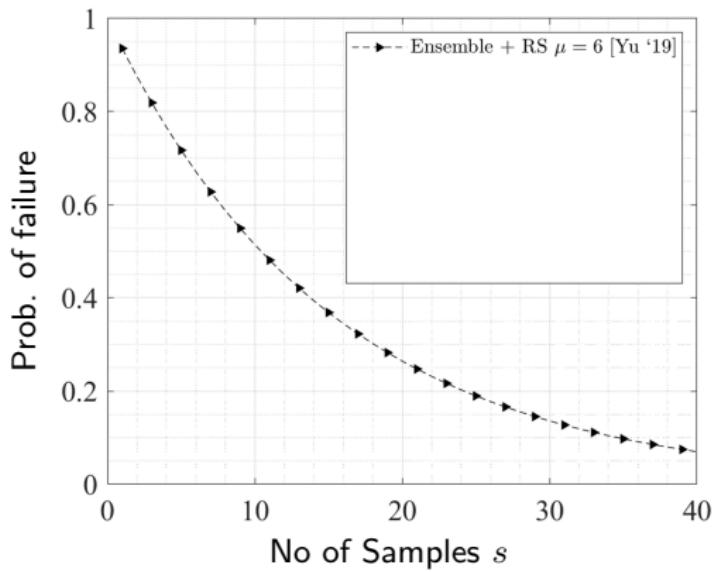
# Simulation Results

Probability of failure for a stopping set of size  $\mu$

# Simulation Results

Probability of failure for a stopping set of size  $\mu$

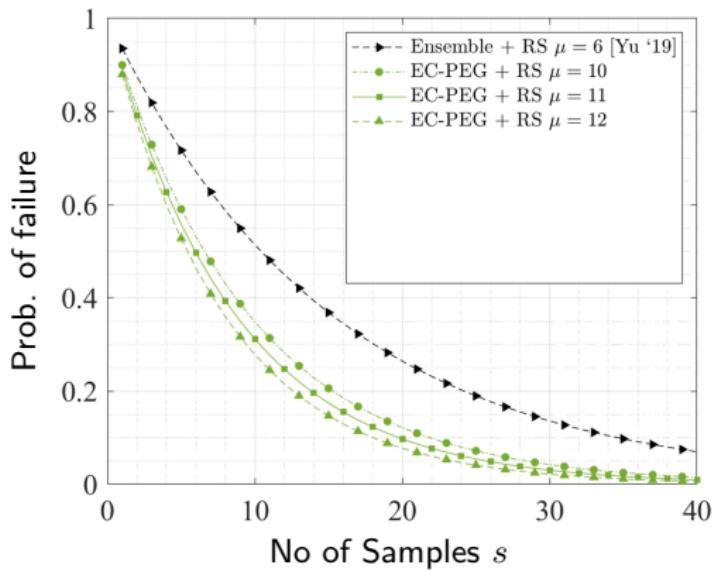
RS: Random Sampling



# Simulation Results

Probability of failure for a stopping set of size  $\mu$

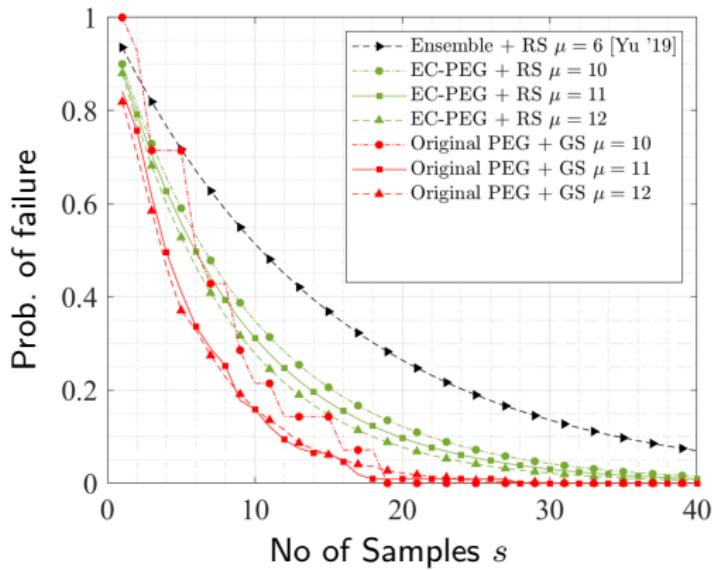
RS: Random Sampling



# Simulation Results

Probability of failure for a stopping set of size  $\mu$

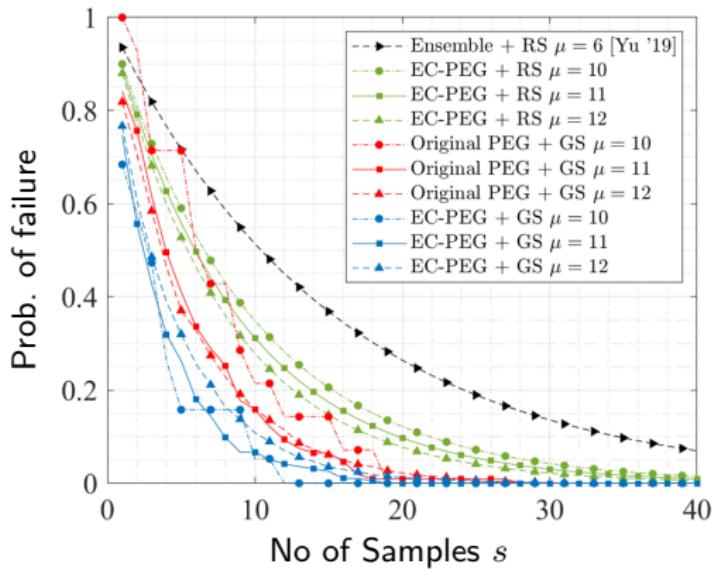
RS: Random Sampling  
GS: Greedy Sampling



# Simulation Results

Probability of failure for a stopping set of size  $\mu$

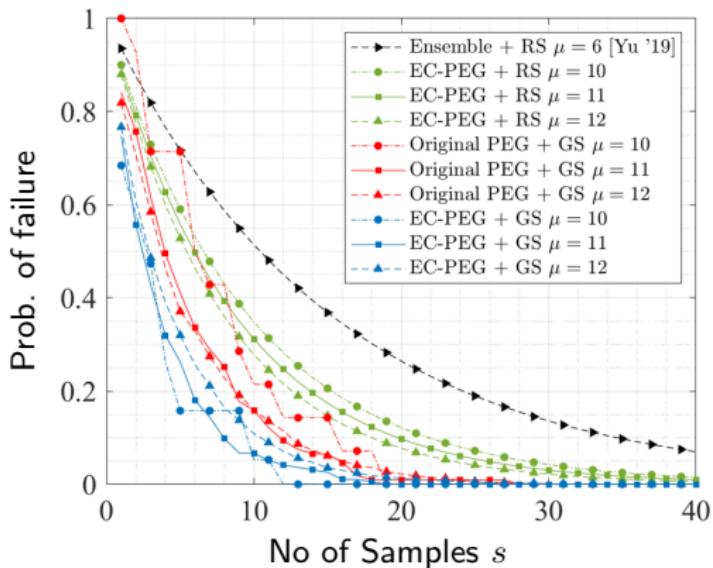
RS: Random Sampling  
GS: Greedy Sampling



# Simulation Results

Probability of failure for a stopping set of size  $\mu$

RS: Random Sampling  
GS: Greedy Sampling



- ▶ Concentrated LDPC codes with Greedy sampling improve the probability of failure

# Incorrect Coding Proof Size

- ▶ Depends on the maximum check node degree

Rate	Code length	VN degree	Ensemble [Yu '19]	PEG	EC-PEG
$\frac{1}{2}$	100	4	16	9	11
	200	4	16	9	15
$\frac{1}{4}$	100	4	8	7	10
	200	4	8	6	9

Table: Maximum CN degree for different codes.

# Incorrect Coding Proof Size

- Depends on the maximum check node degree

Rate	Code length	VN degree	Ensemble [Yu '19]	PEG	EC-PEG
$\frac{1}{2}$	100	4	16	9	11
	200	4	16	9	15
$\frac{1}{4}$	100	4	8	7	10
	200	4	8	6	9

Table: Maximum CN degree for different codes.

- Concentrated LDPC codes do not sacrifice on the incorrect coding proof size

# Table of Contents

1. Background and Central Problems
2. Data Availability Attacks on Light Nodes
3. Data Availability Attacks in Side Blockchains
4. Conclusion

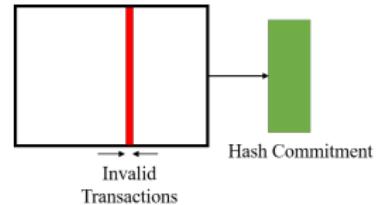
# Data Availability Attack in Side Blockchains

Side Blockchains with a **majority of dishonest nodes** are vulnerable to data availability attacks [Sheng '20]

# Data Availability Attack in Side Blockchains

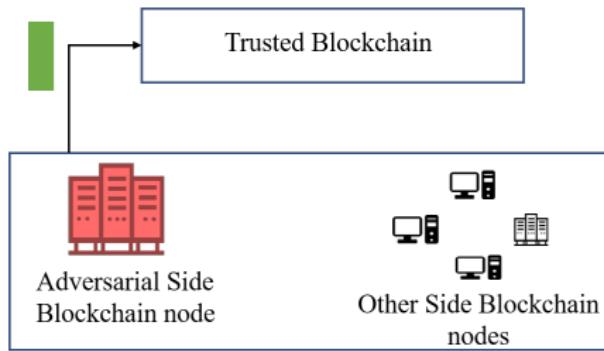
Side Blockchains with a **majority of dishonest nodes** are vulnerable to data availability attacks [Sheng '20]

Adversary creates an invalid block

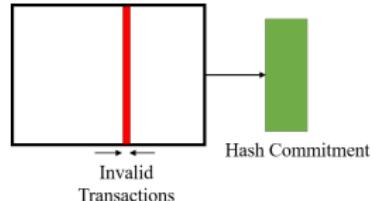


# Data Availability Attack in Side Blockchains

Side Blockchains with a **majority of dishonest nodes** are vulnerable to data availability attacks [Sheng '20]



Adversary creates an invalid block

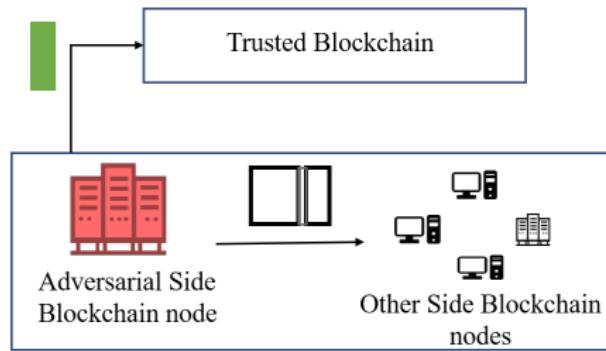


Adversarial Side blockchain node:

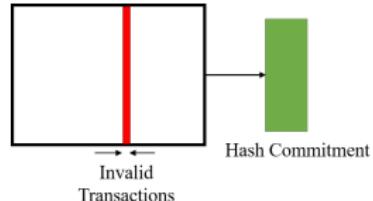
- ▶ Pushes hash commitment to the trusted blockchain

# Data Availability Attack in Side Blockchains

Side Blockchains with a **majority of dishonest nodes** are vulnerable to data availability attacks [Sheng '20]



Adversary creates an invalid block

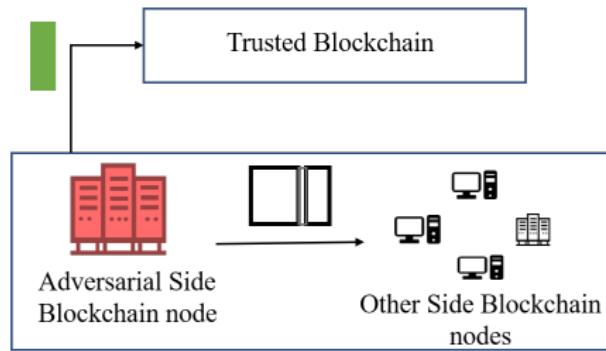


Adversarial Side blockchain node:

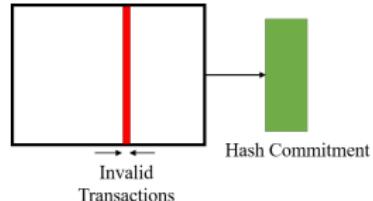
- ▶ Pushes hash commitment to the trusted blockchain
- ▶ Full block not available to other side blockchain nodes

# Data Availability Attack in Side Blockchains

Side Blockchains with a **majority of dishonest nodes** are vulnerable to data availability attacks [Sheng '20]



Adversary creates an invalid block

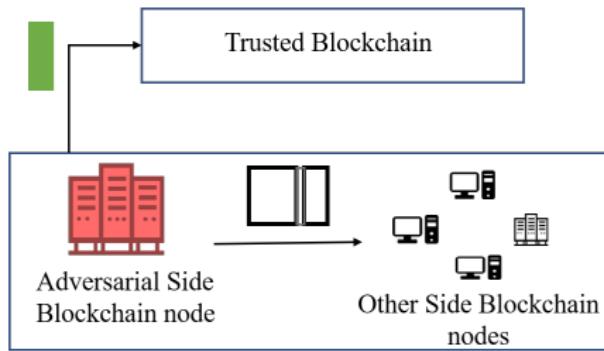


Adversarial Side blockchain node:

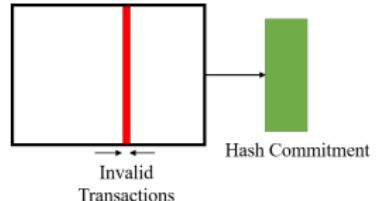
- ▶ Pushes hash commitment to the trusted blockchain
- ▶ Full block not available to other side blockchain nodes
- ▶ The invalid block becomes part of the transaction ordering in the trusted blockchain

# Data Availability Attack in Side Blockchains

Side Blockchains with a **majority of dishonest nodes** are vulnerable to data availability attacks [Sheng '20]



Adversary creates an invalid block



Note: DA attack cannot occur when side blockchains have a majority of honest nodes → majority vote on “is something missing”

Adversarial Side blockchain node:

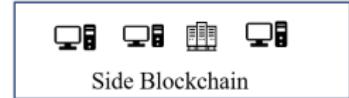
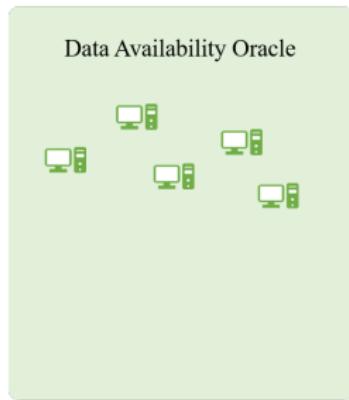
- ▶ Pushes hash commitment to the trusted blockchain
- ▶ Full block not available to other side blockchain nodes
- ▶ The invalid block becomes part of the transaction ordering in the trusted blockchain

## Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]



# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

Oracle layer goal

Trusted Blockchain

Data Availability Oracle



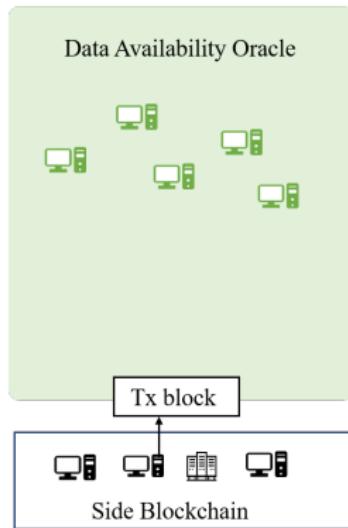
Side Blockchain

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

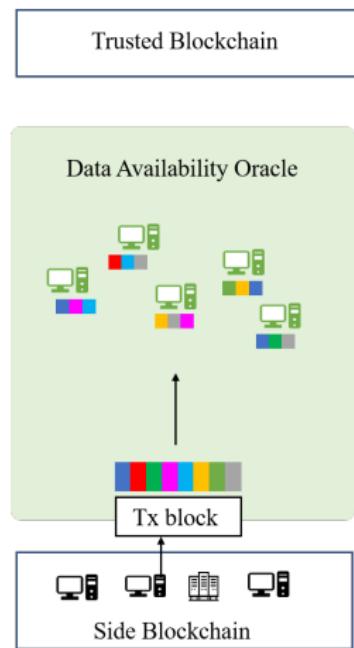
Oracle layer goal

- ▶ Accept a Tx block



# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

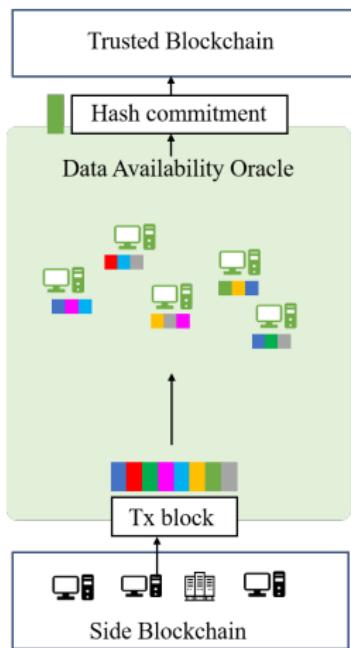


Oracle layer goal

- ▶ Accept a Tx block
- ▶ Collectively and efficiently store chunks of the Tx block  
(to guarantee availability)

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

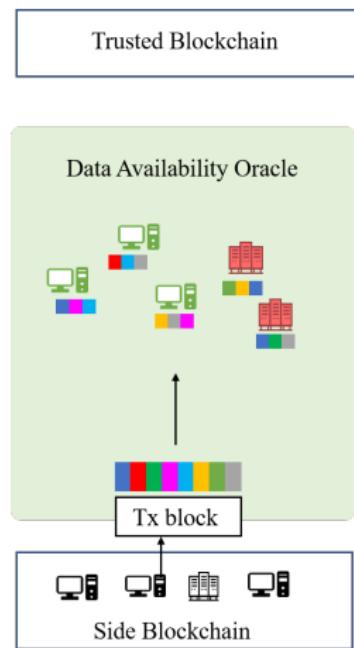


## Oracle layer goal

- ▶ Accept a Tx block
- ▶ Collectively and efficiently store chunks of the Tx block (to guarantee availability)
- ▶ Push the Tx block's hash commitment iff the block is available

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]



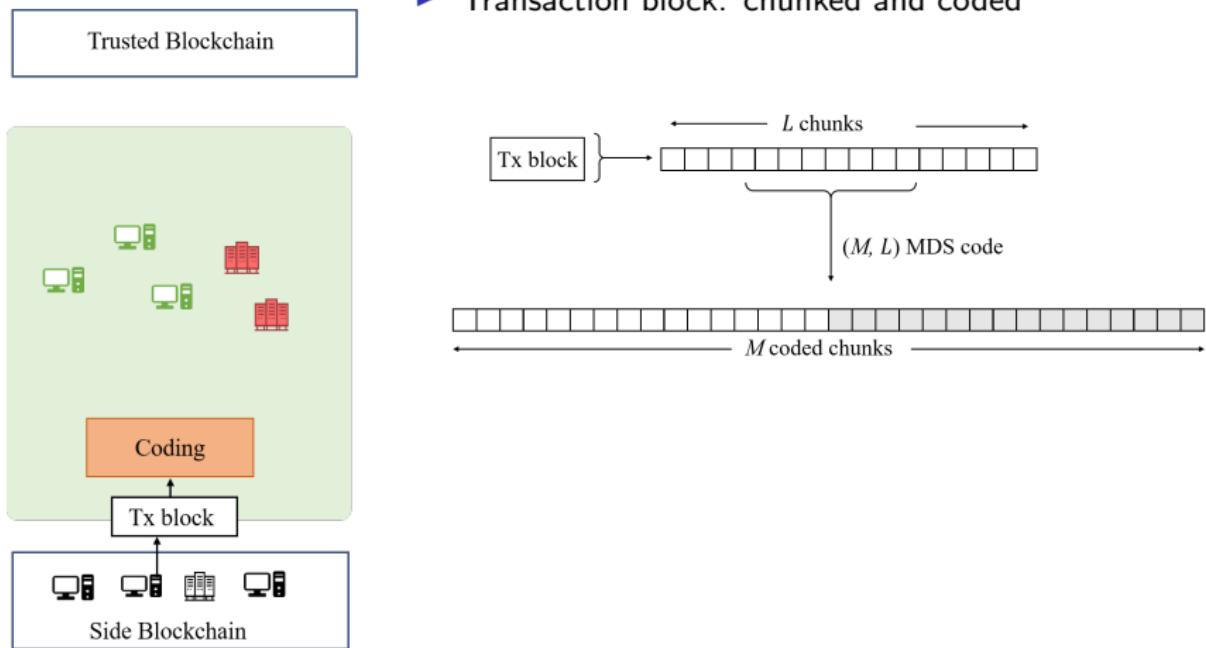
Oracle layer goal

- ▶ Accept a Tx block
- ▶ Collectively and efficiently store chunks of the Tx block (to guarantee availability)
- ▶ Push the Tx block's hash commitment iff the block is available
- ▶ Oracle nodes can be malicious (honest majority)

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

- ▶ Transaction block: chunked and coded

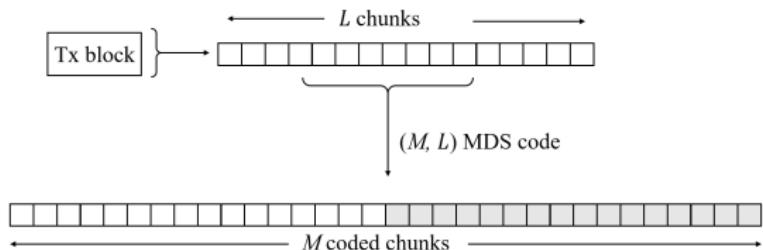
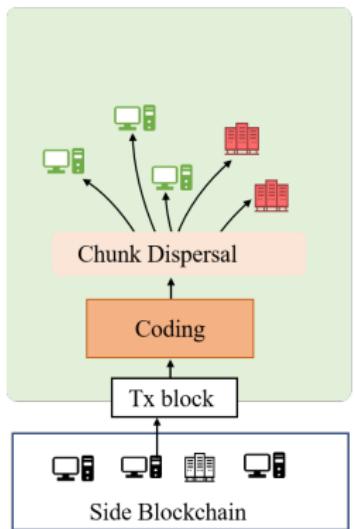


# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

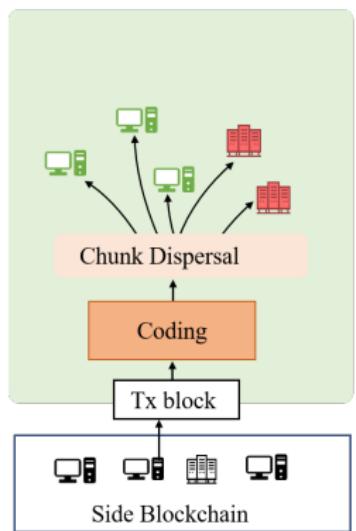


- ▶ Transaction block: chunked and coded
- ▶ Coded chunks dispersed among oracle nodes

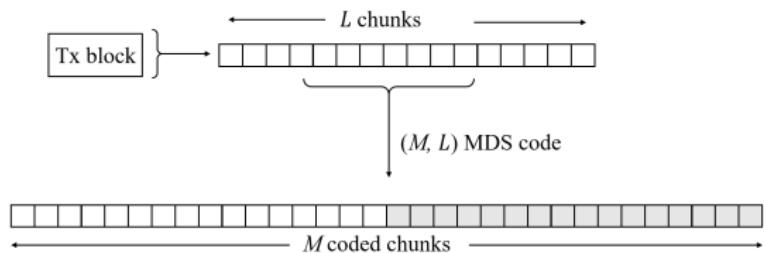


# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]



- ▶ Transaction block: chunked and coded
- ▶ Coded chunks dispersed among oracle nodes

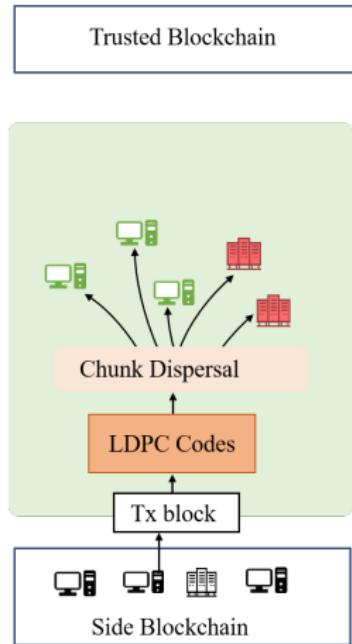


For MDS codes, iff at least  $L$  coded chunks are present among honest oracle layer nodes → block availability is guaranteed

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

LDPC codes are used to code the Tx block

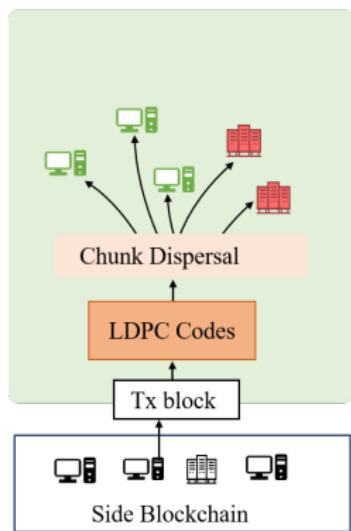


# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

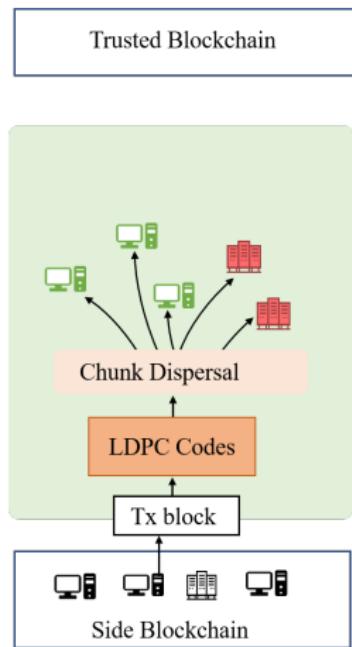
LDPC codes are used to code the Tx block

- ▶ Linear decoding complexity



# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

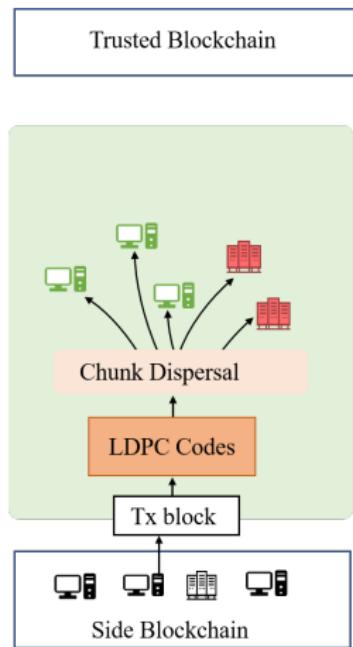


LDPC codes are used to code the Tx block

- ▶ Linear decoding complexity
- ▶ Small incorrect coding proof size due to sparse parity check matrix

# Solution using a Data Availability Oracle

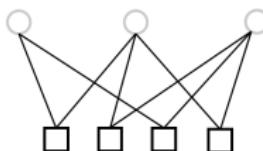
An oracle layer was introduced to ensure data availability [Sheng '20]



LDPC codes are used to code the Tx block

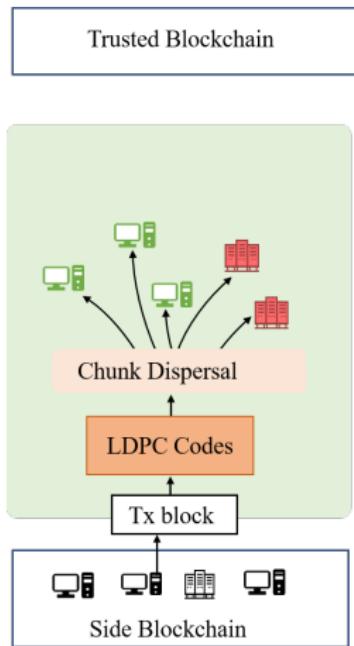
- ▶ Linear decoding complexity
- ▶ Small incorrect coding proof size due to sparse parity check matrix

Issues with LDPC codes: small stopping sets



# Solution using a Data Availability Oracle

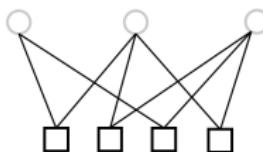
An oracle layer was introduced to ensure data availability [Sheng '20]



LDPC codes are used to code the Tx block

- ▶ Linear decoding complexity
- ▶ Small incorrect coding proof size due to sparse parity check matrix

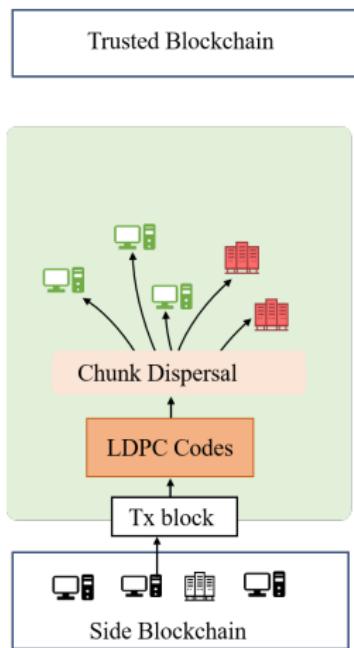
Issues with LDPC codes: small stopping sets



- ▶ If VNs corresponding to a small stopping set are hidden from the oracle nodes, original block cannot be decoded back by a peeling decoder

# Solution using a Data Availability Oracle

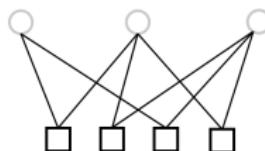
An oracle layer was introduced to ensure data availability [Sheng '20]



LDPC codes are used to code the Tx block

- ▶ Linear decoding complexity
- ▶ Small incorrect coding proof size due to sparse parity check matrix

Issues with LDPC codes: small stopping sets

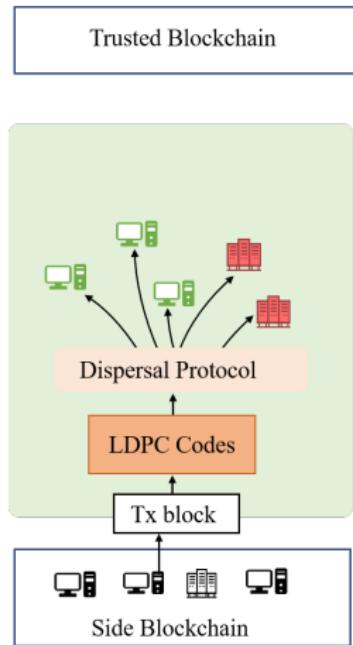


- ▶ If VNs corresponding to a small stopping set are hidden from the oracle nodes, original block cannot be decoded back by a peeling decoder
- ▶ In [Sheng '20] randomly constructed LDPC codes were used which provides a guarantee on the minimum stopping set size **w.h.p**

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

Dispersal Protocol

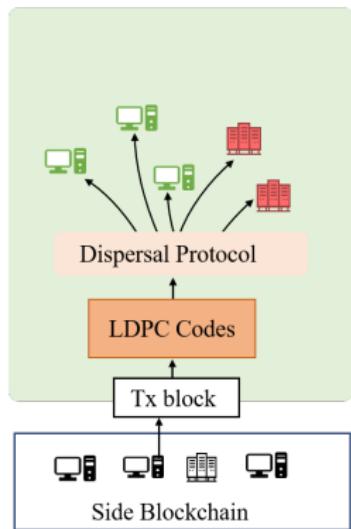


# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

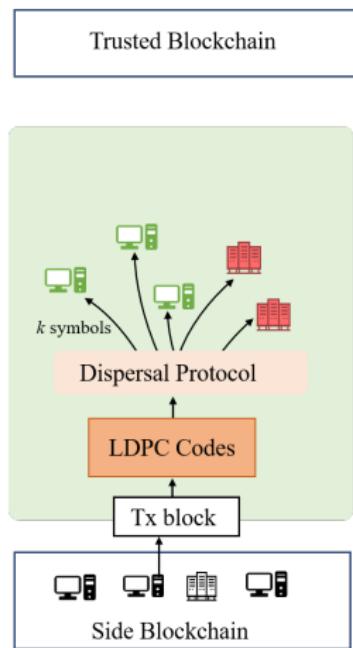
Dispersal Protocol

- Rule about which oracle node stores which coded chunks



# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

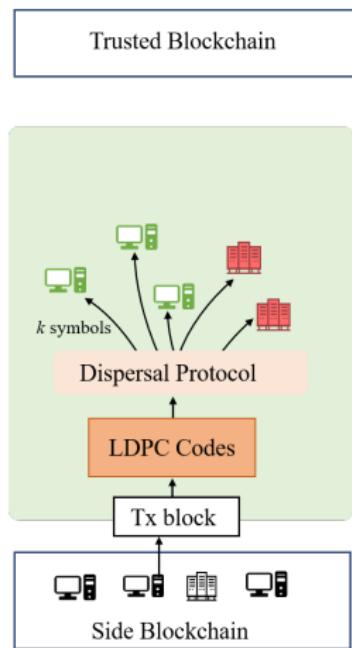


## Dispersal Protocol

- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

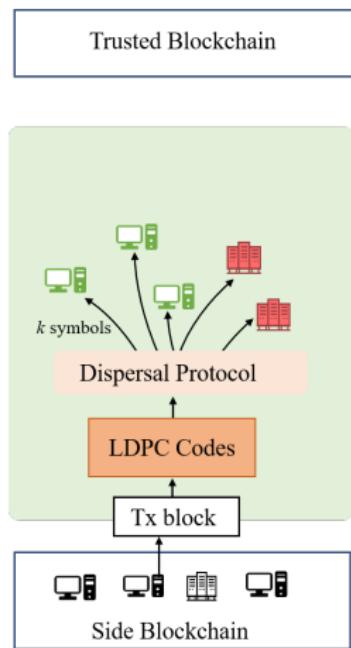


## Dispersal Protocol

- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive
- ▶  $M_{\min} :=$  minimum stopping set size of the LDPC code of block length  $M$

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

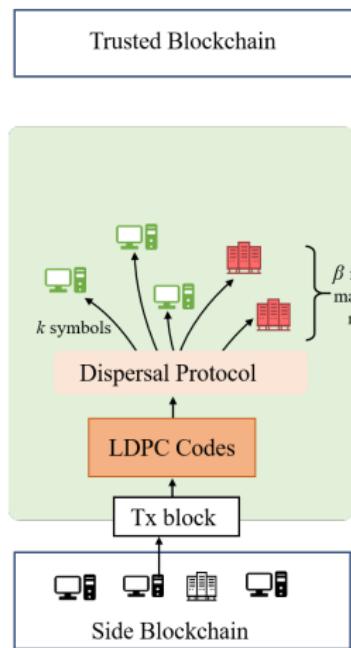


## Dispersal Protocol

- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive
- ▶  $M_{\min} :=$  minimum stopping set size of the LDPC code of block length  $M$
- ▶ Every  $\gamma$  fraction of nodes should receive at least  $M - M_{\min} + 1$  coded chunks

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]

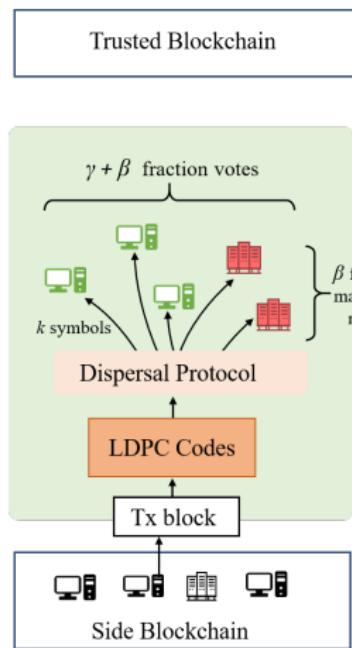


## Dispersal Protocol

- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive
- ▶  $M_{\min} :=$  minimum stopping set size of the LDPC code of block length  $M$
- ▶ Every  $\gamma$  fraction of nodes should receive at least  $M - M_{\min} + 1$  coded chunks
- $\beta :=$  fraction of malicious oracle nodes

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]



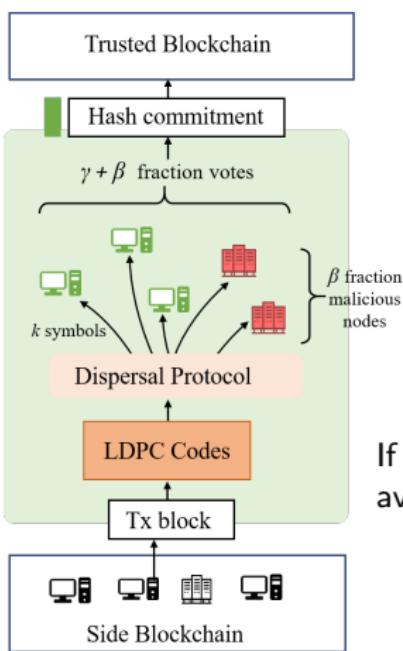
## Dispersal Protocol

- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive
- ▶  $M_{\min} :=$  minimum stopping set size of the LDPC code of block length  $M$
- ▶ Every  $\gamma$  fraction of nodes should receive at least  $M - M_{\min} + 1$  coded chunks
  - $\beta :=$  fraction of malicious oracle nodes

If more than  $\gamma + \beta$  fraction of nodes vote that the block is available,

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]



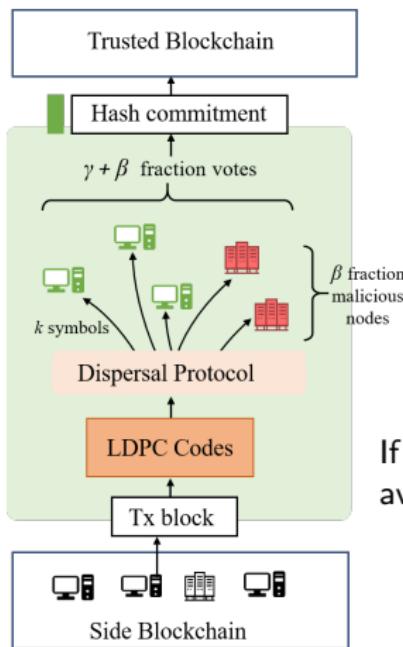
## Dispersal Protocol

- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive
- ▶  $M_{\min} :=$  minimum stopping set size of the LDPC code of block length  $M$
- ▶ Every  $\gamma$  fraction of nodes should receive at least  $M - M_{\min} + 1$  coded chunks
  - $\beta :=$  fraction of malicious oracle nodes

If more than  $\gamma + \beta$  fraction of nodes vote that the block is available, then the hash commitment is pushed.

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]



## Dispersal Protocol

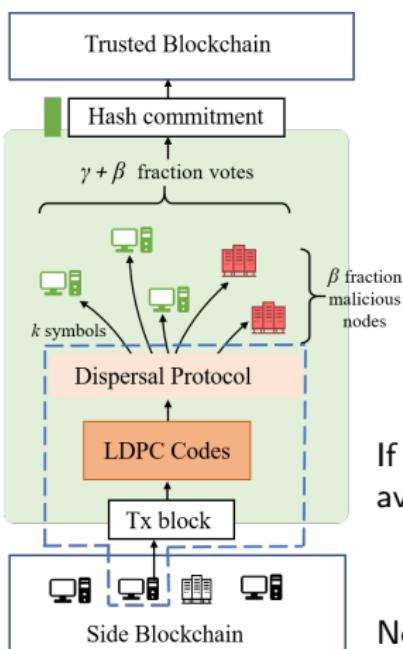
- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive
- ▶  $M_{\min} :=$  minimum stopping set size of the LDPC code of block length  $M$
- ▶ Every  $\gamma$  fraction of nodes should receive at least  $M - M_{\min} + 1$  coded chunks
  - $\beta :=$  fraction of malicious oracle nodes

If more than  $\gamma + \beta$  fraction of nodes vote that the block is available, then the hash commitment is pushed.

The oracle guarantees block availability

# Solution using a Data Availability Oracle

An oracle layer was introduced to ensure data availability [Sheng '20]



## Dispersal Protocol

- ▶ Rule about which oracle node stores which coded chunks
- ▶ Specifies  $k$  coded symbols that each oracle node should receive
- ▶  $M_{\min} :=$  minimum stopping set size of the LDPC code of block length  $M$
- ▶ Every  $\gamma$  fraction of nodes should receive at least  $M - M_{\min} + 1$  coded chunks
  - $\beta :=$  fraction of malicious oracle nodes

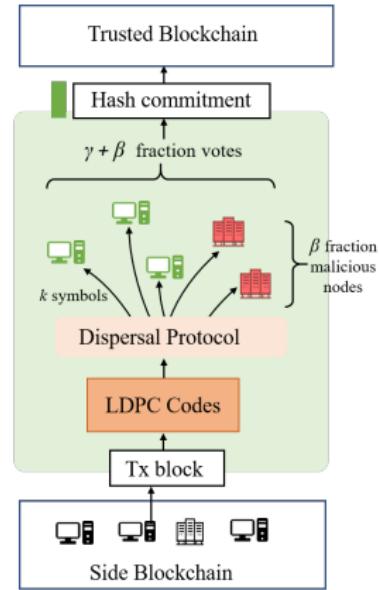
If more than  $\gamma + \beta$  fraction of nodes vote that the block is available, then the hash commitment is pushed.

The oracle guarantees block availability

Note: Side blockchain nodes perform LDPC encoding and dispersal

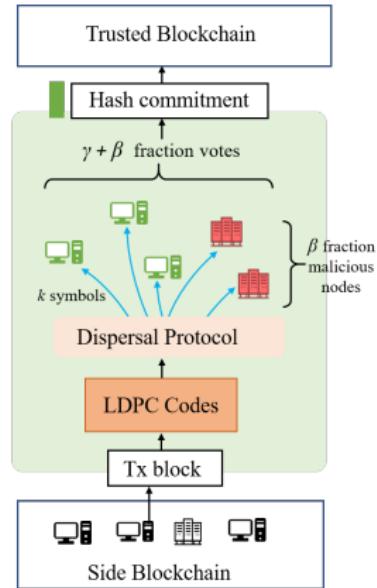
# Design Objective: Minimize Communication Cost

- ▶ Communication cost:



# Design Objective: Minimize Communication Cost

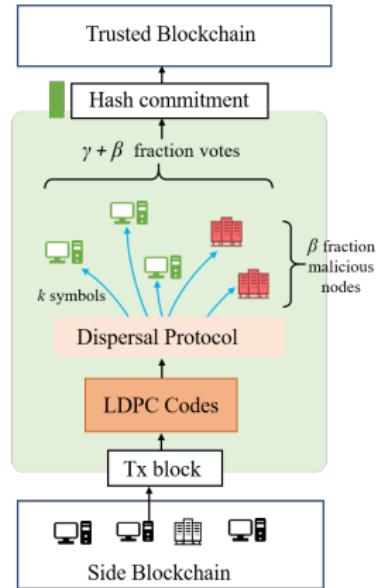
- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal



# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal

Dispersal protocol in prior work [Sheng '20]:

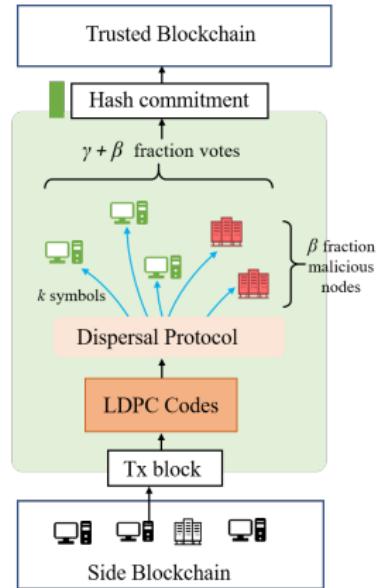


# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal

Dispersal protocol in prior work [Sheng '20]:

- ▶ Every  $\gamma$  fraction of oracle nodes receives  $\geq M - M_{\min} + 1$  coded chunks

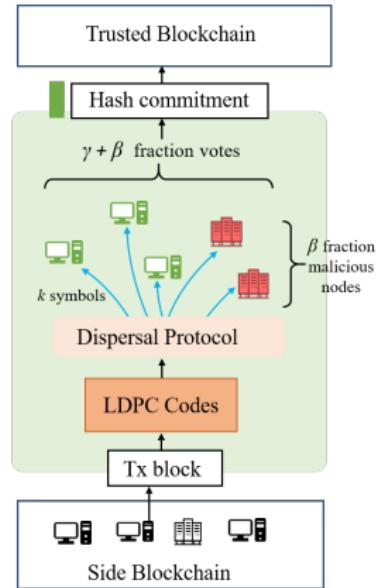


# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal

Dispersal protocol in prior work [Sheng '20]:

- ▶ Every  $\gamma$  fraction of oracle nodes receives  $\geq M - M_{\min} + 1$  coded chunks
  - designed randomly (sampling with replacement)

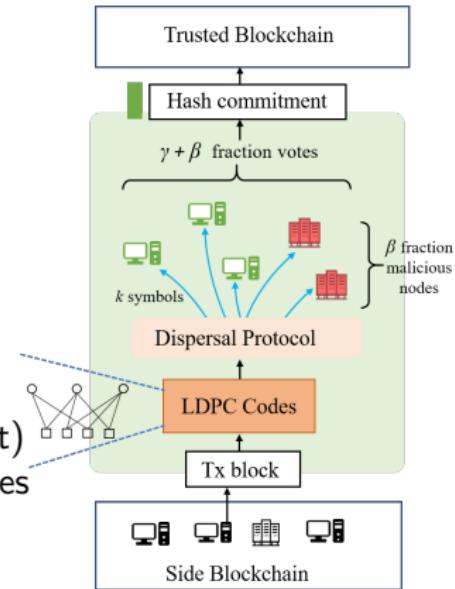


# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal

Dispersal protocol in prior work [Sheng '20]:

- ▶ Every  $\gamma$  fraction of oracle nodes receives  $\geq M - M_{\min} + 1$  coded chunks
  - designed randomly (sampling with replacement)
  - $M_{\min} \downarrow \implies$  send more chunks to oracle nodes  
 $\implies$  communication cost  $\uparrow$

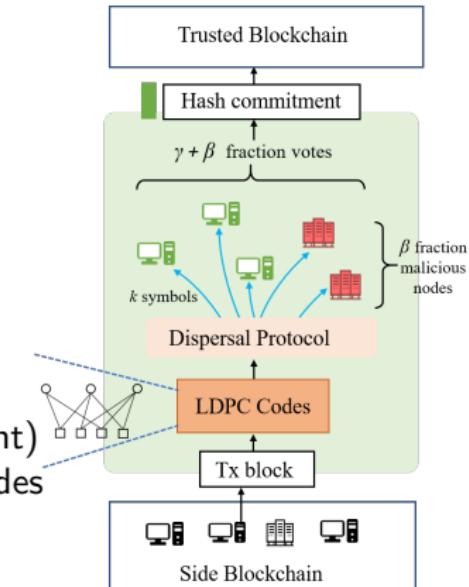


# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal

Dispersal protocol in prior work [Sheng '20]:

- ▶ Every  $\gamma$  fraction of oracle nodes receives  $\geq M - M_{\min} + 1$  coded chunks
  - designed randomly (sampling with replacement)
  - $M_{\min} \downarrow \implies$  send more chunks to oracle nodes  
 $\implies$  communication cost  $\uparrow$



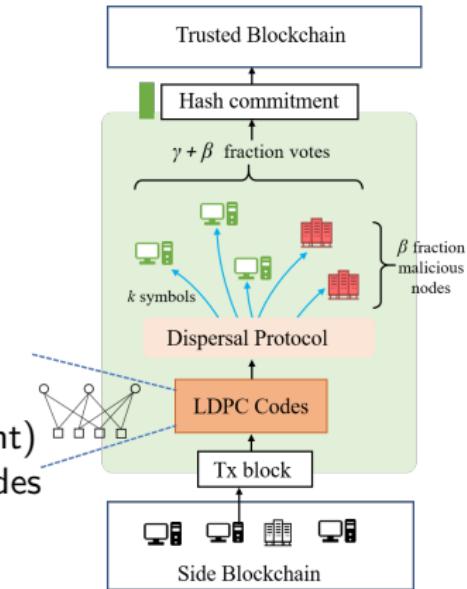
Simply design LDPC codes with large smallest stopping set size  $M_{\min}$ ?

# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal

Dispersal protocol in prior work [Sheng '20]:

- ▶ Every  $\gamma$  fraction of oracle nodes receives  $\geq M - M_{\min} + 1$  coded chunks
  - designed randomly (sampling with replacement)
  - $M_{\min} \downarrow \implies$  send more chunks to oracle nodes  
 $\implies$  communication cost  $\uparrow$



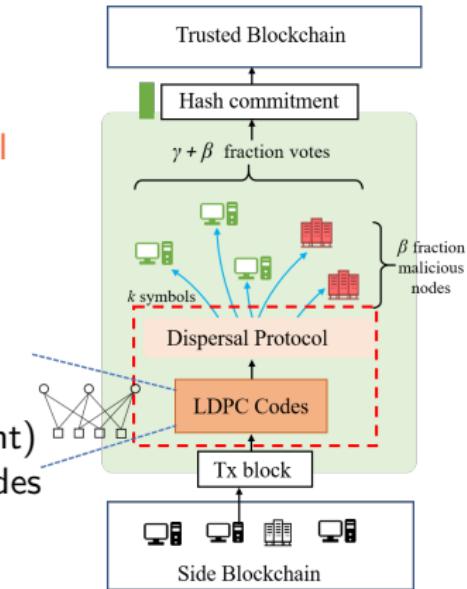
Simply design LDPC codes with large smallest stopping set size  $M_{\min}$ ?  
 $\rightarrow$  known hard problem [Jiao '09], [He '11]

# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal
  - ↳ Affected by the co-design of Dispersal protocol and LDPC code

Dispersal protocol in prior work [Sheng '20]:

- ▶ Every  $\gamma$  fraction of oracle nodes receives  $\geq M - M_{\min} + 1$  coded chunks
  - designed randomly (sampling with replacement)
  - $M_{\min} \downarrow \implies$  send more chunks to oracle nodes  
 $\implies$  communication cost  $\uparrow$



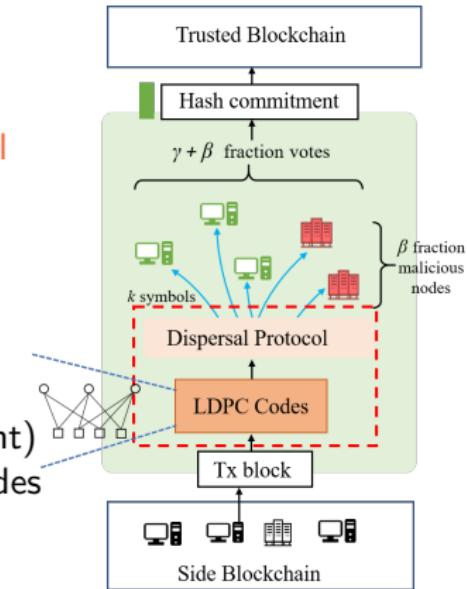
Simply design LDPC codes with large smallest stopping set size  $M_{\min}$ ?  
 $\rightarrow$  known hard problem [Jiao '09], [He '11]

# Design Objective: Minimize Communication Cost

- ▶ Communication cost: amount of data communicated to oracle nodes during dispersal
  - ↳ Affected by the co-design of Dispersal protocol and LDPC code

Dispersal protocol in prior work [Sheng '20]:

- ▶ Every  $\gamma$  fraction of oracle nodes receives  $\geq M - M_{\min} + 1$  coded chunks
  - designed randomly (sampling with replacement)
  - $M_{\min} \downarrow \implies$  send more chunks to oracle nodes  
 $\implies$  communication cost  $\uparrow$



Our work: Design of specialized LDPC codes and a tailored dispersal protocol to significantly lower the communication cost.

# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

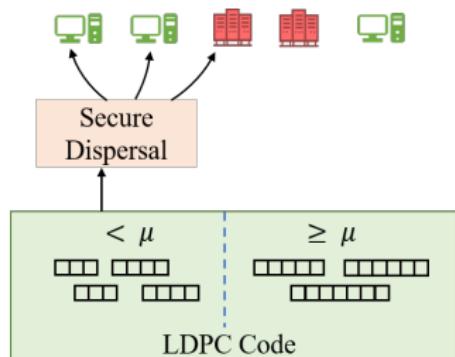
1. Secure Phase

# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

## 1. Secure Phase

All small SSs (size  $< \mu$ ) are treated in an unified manner



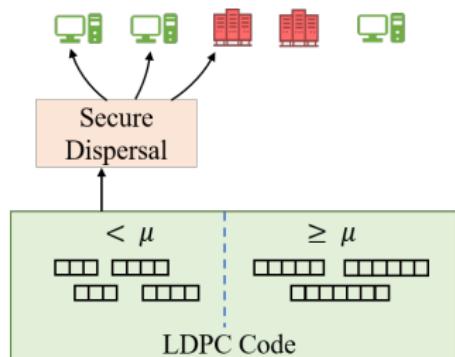
# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

## 1. Secure Phase

All small SSs (size  $< \mu$ ) are treated in an unified manner

- ▶ Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur



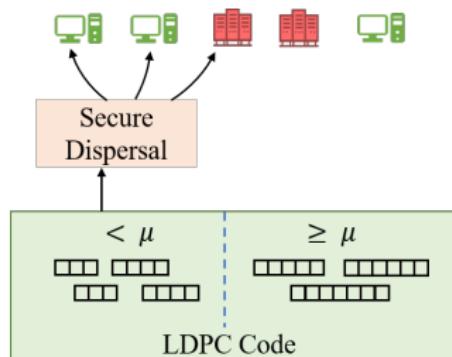
# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

## 1. Secure Phase

All small SSs (size  $< \mu$ ) are treated in an unified manner

- ▶ Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur



## 2. Valid Phase

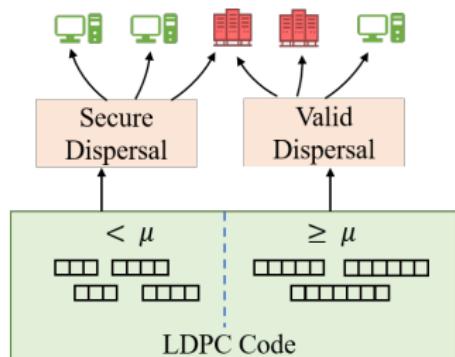
# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

## 1. Secure Phase

All small SSs (size  $< \mu$ ) are treated in an unified manner

- ▶ Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur



## 2. Valid Phase

A refinement of the dispersal protocol used in [Sheng '20] for larger SSs (size  $\geq \mu$ )

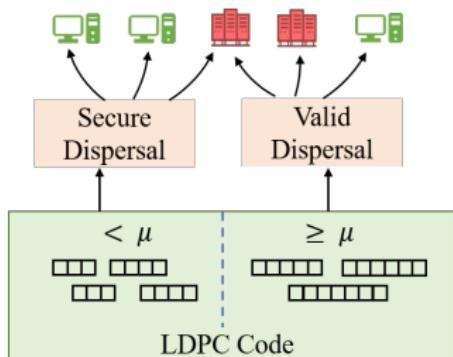
# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

## 1. Secure Phase

All small SSs (size  $< \mu$ ) are treated in an unified manner

- ▶ Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur



## 2. Valid Phase

A refinement of the dispersal protocol used in [Sheng '20] for larger SSs (size  $\geq \mu$ )

- ▶ Coded chunks are dispersed in a communication-efficient way such that availability is guaranteed under the large SS failures

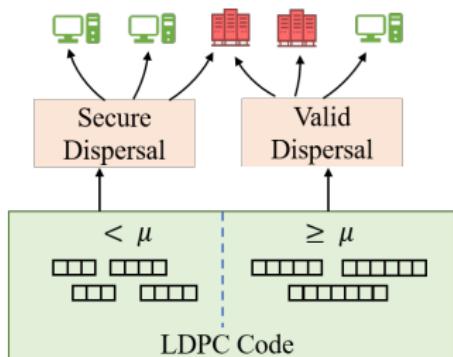
# Dispersal Protocol Design

Our dispersal strategy is a two step protocol

## 1. Secure Phase

All small SSs (size  $< \mu$ ) are treated in an unified manner

- ▶ Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur



## 2. Valid Phase

A refinement of the dispersal protocol used in [Sheng '20] for larger SSs (size  $\geq \mu$ )

- ▶ Coded chunks are dispersed in a communication-efficient way such that availability is guaranteed under the large SS failures

### Code Design Strategy:

Design LDPC codes that reduce communication cost of the secure phase

## Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$

## Secure Phase

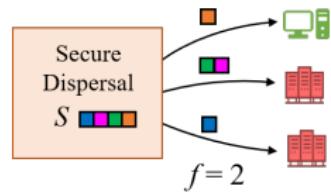
- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes

## Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$

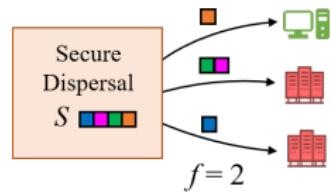
# Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$



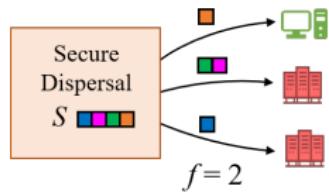
# Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$



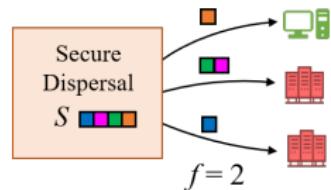
# Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur



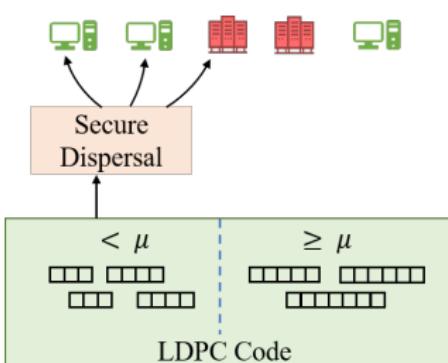
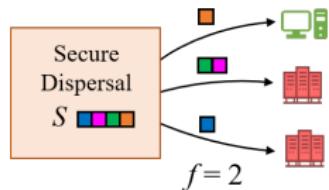
# Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur
  - ▶  $\mathcal{S} = \text{All SSs of size } < \mu$



# Secure Phase

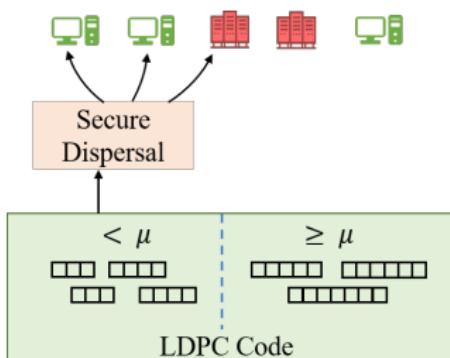
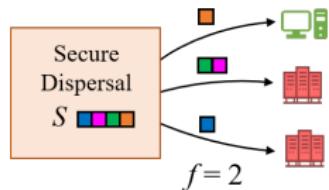
- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur



- ▶  $\mathcal{S} =$  All SSs of size  $< \mu$   
Secure phase: all SSs in  $\mathcal{S}$  are *securely dispersed*

# Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur

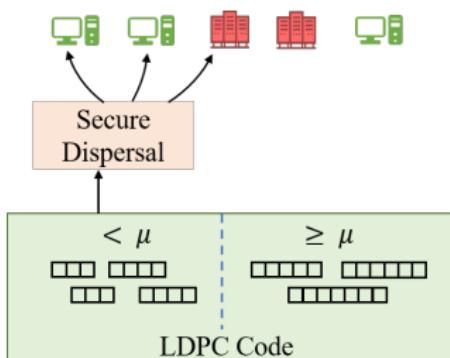
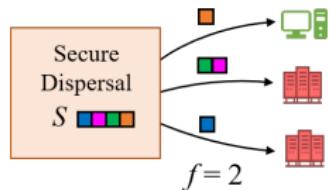


- ▶  $\mathcal{S} =$  All SSs of size  $< \mu$   
Secure phase: all SSs in  $\mathcal{S}$  are *securely dispersed*

$< \mu$  size SSs cannot cause block unavailability

# Secure Phase

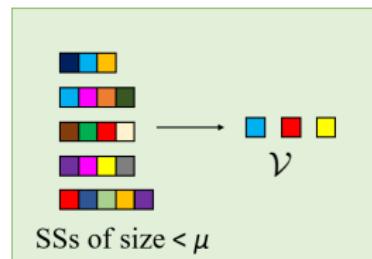
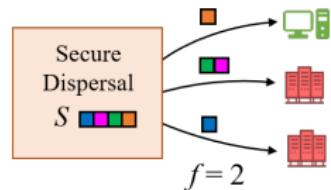
- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur



- ▶  $\mathcal{S} =$  All SSs of size  $< \mu$   
Secure phase: all SSs in  $\mathcal{S}$  are *securely dispersed*
- $< \mu$  size SSs cannot cause block unavailability
- $\mathcal{V}$ : set of VNs that *cover* all SSs in  $\mathcal{S}$

# Secure Phase

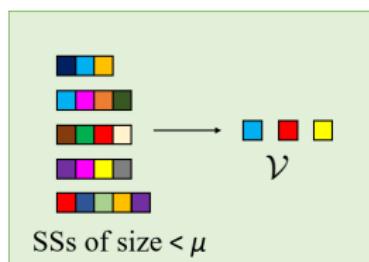
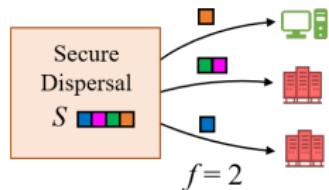
- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur



- ▶  $\mathcal{S} =$  All SSs of size  $< \mu$   
Secure phase: all SSs in  $\mathcal{S}$  are *securely dispersed*
- $< \mu$  size SSs cannot cause block unavailability
- $\mathcal{V}$ : set of VNs that *cover* all SSs in  $\mathcal{S}$

# Secure Phase

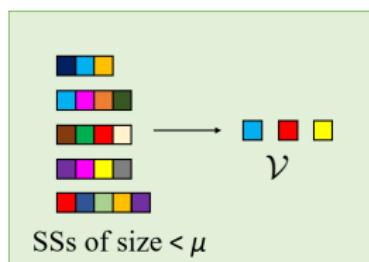
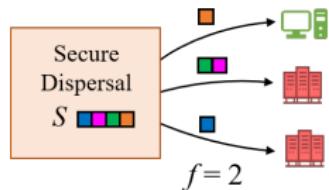
- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur



- ▶  $\mathcal{S} =$  All SSs of size  $< \mu$   
Secure phase: all SSs in  $\mathcal{S}$  are *securely dispersed*
- $< \mu$  size SSs cannot cause block unavailability
- $\mathcal{V}$ : set of VNs that *cover* all SSs in  $\mathcal{S}$
- Each VN in  $\mathcal{V}$  is dispersed to  $f + 1$  nodes

# Secure Phase

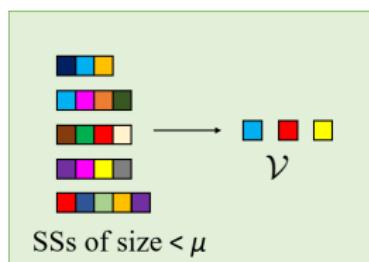
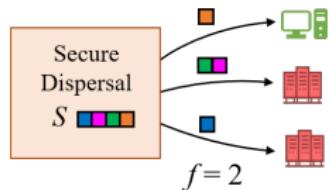
- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur



- ▶  $\mathcal{S} =$  All SSs of size  $< \mu$   
Secure phase: all SSs in  $\mathcal{S}$  are *securely dispersed*
- $< \mu$  size SSs cannot cause block unavailability
- $\mathcal{V}$ : set of VNs that *cover* all SSs in  $\mathcal{S}$
- Each VN in  $\mathcal{V}$  is dispersed to  $f + 1$  nodes  
→ ensures all SSs in  $\mathcal{S}$  are securely dispersed

# Secure Phase

- ▶  $\text{Neigh}(S) :=$  number of oracle nodes having at least one coded chunk of stopping set  $S$
- ▶  $f :=$  maximum number of malicious oracle nodes
- ▶  $S$  is *securely dispersed* if  $|\text{Neigh}(S)| \geq f + 1$
- ▶ If a stopping set  $S$  is *securely dispersed*, at least one honest node will have a coded chunk corresponding to  $S$   
→ Failure of stopping set  $S$  cannot occur



- ▶  $\mathcal{S} =$  All SSs of size  $< \mu$   
Secure phase: all SSs in  $\mathcal{S}$  are *securely dispersed*
- $< \mu$  size SSs cannot cause block unavailability
- $\mathcal{V}$ : set of VNs that *cover* all SSs in  $\mathcal{S}$   
→ found greedily: *Greedy-Set*( $\mathcal{S}$ )
- Each VN in  $\mathcal{V}$  is dispersed to  $f + 1$  nodes  
→ ensures all SSs in  $\mathcal{S}$  are securely dispersed

## Valid Phase

Consider the following dispersal protocol

## Valid Phase

Consider the following dispersal protocol

$\mu$ -SS-Valid dispersal

Every  $\gamma$  fraction of oracle nodes receives  $\geq M - \mu + 1$  coded chunks

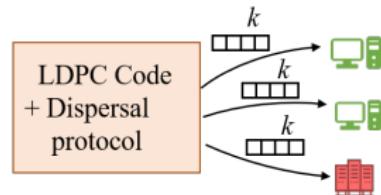
## Valid Phase

Consider the following dispersal protocol

$\mu$ -SS-Valid dispersal

Every  $\gamma$  fraction of oracle nodes receives  $\geq M - \mu + 1$  coded chunks

- ▶ Each oracle node receives coded chunks corresponding to a uniformly chosen  $k$ -element subset of all the  $k$ -element subsets of the  $M$  coded chunks



# Valid Phase

Consider the following dispersal protocol

## $\mu$ -SS-Valid dispersal

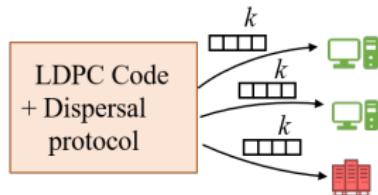
Every  $\gamma$  fraction of oracle nodes receives  $\geq M - \mu + 1$  coded chunks

- ▶ Each oracle node receives coded chunks corresponding to a uniformly chosen  $k$ -element subset of all the  $k$ -element subsets of the  $M$  coded chunks

## Lemma

$\text{Prob}(\text{dispersal is not } \mu\text{-SS valid}) \leq e^{NH_e(\gamma)} P_f$

$$P_f = \sum_{j=0}^{M-\mu} (-1)^{M-\mu-j} \binom{M}{j} \binom{M-j-1}{\mu-1} \left[ \frac{\binom{j}{k}}{\binom{M}{k}} \right]^{\gamma N}$$



↳ Coupon Collector's problem with group drawings [Stadje '90]

# Valid Phase

Consider the following dispersal protocol

## $\mu$ -SS-Valid dispersal

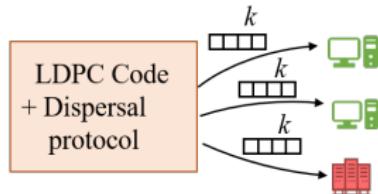
Every  $\gamma$  fraction of oracle nodes receives  $\geq M - \mu + 1$  coded chunks

- ▶ Each oracle node receives coded chunks corresponding to a uniformly chosen  $k$ -element subset of all the  $k$ -element subsets of the  $M$  coded chunks

## Lemma

$\text{Prob}(\text{dispersal is not } \mu\text{-SS valid}) \leq e^{NH_e(\gamma)} P_f$

$$P_f = \sum_{j=0}^{M-\mu} (-1)^{M-\mu-j} \binom{M}{j} \binom{M-j-1}{\mu-1} \left[ \frac{\binom{j}{k}}{\binom{M}{k}} \right]^{\gamma N}$$



↳ Coupon Collector's problem with group drawings [Stadje '90]

- ▶  $k^*(\mu) := \min k$  such that  $e^{NH_e(\gamma)} P_f \leq p_{th}$

# Valid Phase

Consider the following dispersal protocol

## $\mu$ -SS-Valid dispersal

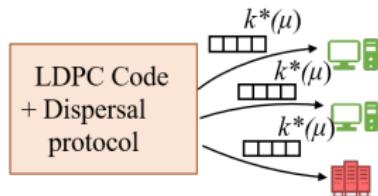
Every  $\gamma$  fraction of oracle nodes receives  $\geq M - \mu + 1$  coded chunks

- ▶ Each oracle node receives coded chunks corresponding to a uniformly chosen  $k$ -element subset of all the  $k$ -element subsets of the  $M$  coded chunks

## Lemma

$\text{Prob}(\text{dispersal is not } \mu\text{-SS valid}) \leq e^{NH_e(\gamma)} P_f$

$$P_f = \sum_{j=0}^{M-\mu} (-1)^{M-\mu-j} \binom{M}{j} \binom{M-j-1}{\mu-1} \left[ \frac{\binom{j}{k}}{\binom{M}{k}} \right]^{\gamma N}$$



↳ Coupon Collector's problem with group drawings [Stadje '90]

- ▶  $k^*(\mu) := \min k$  such that  $e^{NH_e(\gamma)} P_f \leq p_{th}$

# Valid Phase

Consider the following dispersal protocol

## $\mu$ -SS-Valid dispersal

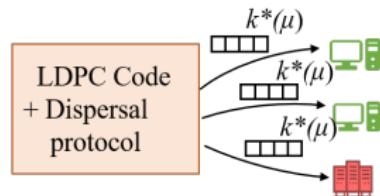
Every  $\gamma$  fraction of oracle nodes receives  $\geq M - \mu + 1$  coded chunks

- Each oracle node receives coded chunks corresponding to a uniformly chosen  $k$ -element subset of all the  $k$ -element subsets of the  $M$  coded chunks

## Lemma

$\text{Prob}(\text{dispersal is not } \mu\text{-SS valid}) \leq e^{NH_e(\gamma)} P_f$

$$P_f = \sum_{j=0}^{M-\mu} (-1)^{M-\mu-j} \binom{M}{j} \binom{M-j-1}{\mu-1} \left[ \frac{\binom{j}{k}}{\binom{M}{k}} \right]^{\gamma N}$$



↳ Coupon Collector's problem with group drawings [Stadje '90]

- $k^*(\mu) := \min k$  such that  $e^{NH_e(\gamma)} P_f \leq p_{th}$

Guarantees availability w.p.  $\geq 1 - p_{th}$

# Overall Dispersal Strategy and Code Design

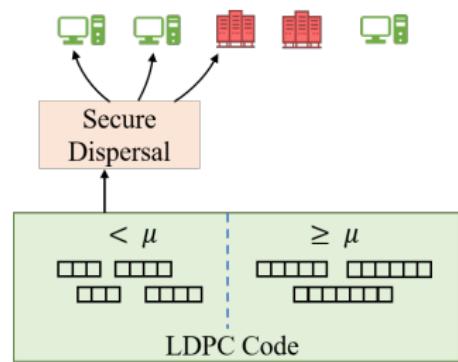
$k^*$ -secure dispersal protocol

# Overall Dispersal Strategy and Code Design

$k^*$ -secure dispersal protocol

## 1. Secure Phase

All SSs of size  $< \mu$  are securely dispersed



# Overall Dispersal Strategy and Code Design

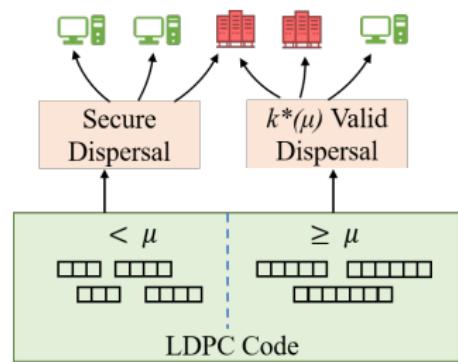
$k^*$ -secure dispersal protocol

## 1. Secure Phase

All SSs of size  $< \mu$  are securely dispersed

## 2. Valid Phase

$k^*(\mu)$  valid dispersal protocol



# Overall Dispersal Strategy and Code Design

$k^*$ -secure dispersal protocol

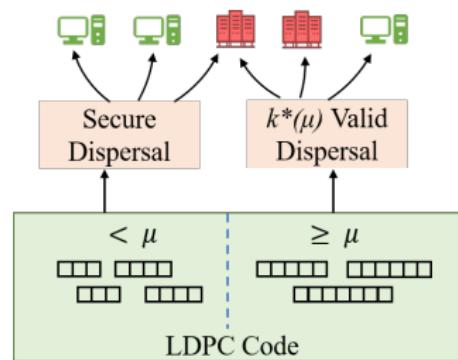
## 1. Secure Phase

All SSs of size  $< \mu$  are securely dispersed

$< \mu$  size SSs cannot cause block unavailability

## 2. Valid Phase

$k^*(\mu)$  valid dispersal protocol



# Overall Dispersal Strategy and Code Design

$k^*$ -secure dispersal protocol

## 1. Secure Phase

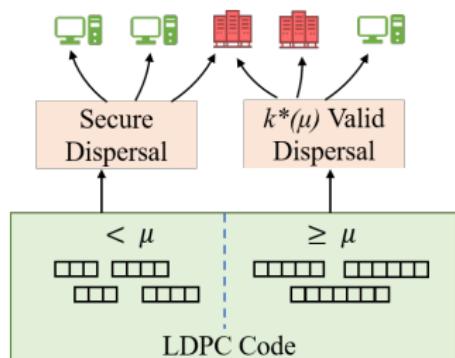
All SSs of size  $< \mu$  are securely dispersed

$< \mu$  size SSs cannot cause block unavailability

## 2. Valid Phase

$k^*(\mu)$  valid dispersal protocol

Guarantees availability w.p.  $\geq 1 - p_{th}$  for SSs of size  $\geq \mu$



# Overall Dispersal Strategy and Code Design

$k^*$ -secure dispersal protocol

## 1. Secure Phase

All SSs of size  $< \mu$  are securely dispersed

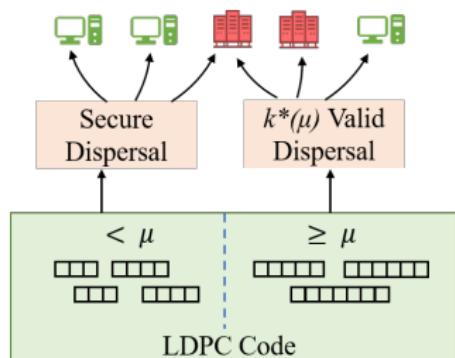
$< \mu$  size SSs cannot cause block unavailability

- Recall: Each VN in  $\mathcal{V}$  is dispersed to  $f + 1$  nodes

## 2. Valid Phase

$k^*(\mu)$  valid dispersal protocol

Guarantees availability w.p.  $\geq 1 - p_{th}$  for SSs of size  $\geq \mu$



# Overall Dispersal Strategy and Code Design

$k^*$ -secure dispersal protocol

## 1. Secure Phase

All SSs of size  $< \mu$  are securely dispersed

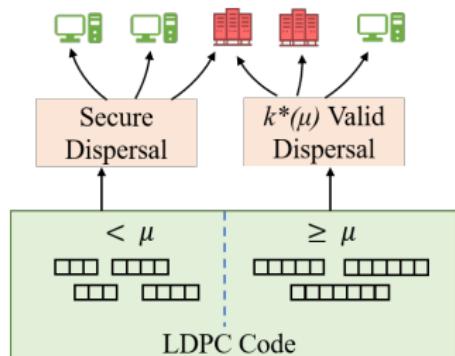
$< \mu$  size SSs cannot cause block unavailability

- Recall: Each VN in  $\mathcal{V}$  is dispersed to  $f + 1$  nodes
- Communication cost  $\propto (f + 1)|\mathcal{V}|$

## 2. Valid Phase

$k^*(\mu)$  valid dispersal protocol

Guarantees availability w.p.  $\geq 1 - p_{th}$  for SSs of size  $\geq \mu$



# Overall Dispersal Strategy and Code Design

$k^*$ -secure dispersal protocol

## 1. Secure Phase

All SSs of size  $< \mu$  are securely dispersed

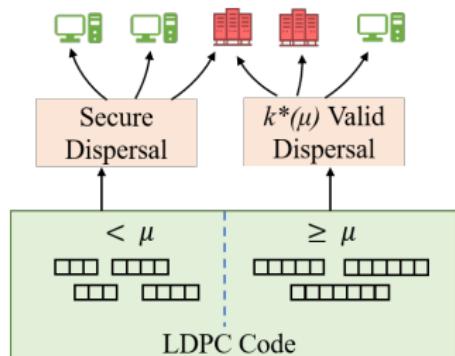
$< \mu$  size SSs cannot cause block unavailability

- Recall: Each VN in  $\mathcal{V}$  is dispersed to  $f + 1$  nodes
- Communication cost  $\propto (f + 1)|\mathcal{V}|$

## 2. Valid Phase

$k^*(\mu)$  valid dispersal protocol

Guarantees availability w.p.  $\geq 1 - p_{th}$  for SSs of size  $\geq \mu$



Code Design Strategy:  
Design LDPC codes that have low  $|\mathcal{V}|$

# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

# LDPC Code Design Strategy

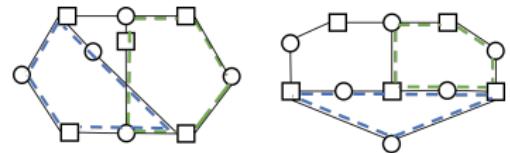
$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

-Modify the PEG algorithm

# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?  
-Modify the PEG algorithm

- ▶ SSs are made up of cycles

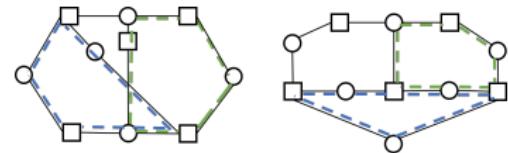


# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

-Modify the PEG algorithm

- ▶ SSs are made up of cycles
- ▶  $\mathcal{L}$  = List of cycles of length  $\leq g$

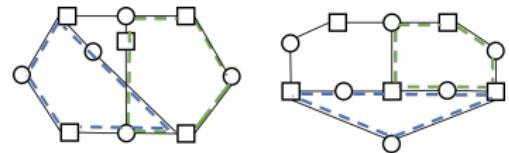


# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

-Modify the PEG algorithm

- ▶ SSs are made up of cycles
- ▶  $\mathcal{L} = \text{List of cycles of length } \leq g$
- LDPC codes to reduce  $|\text{Greedy-Set}(\mathcal{L})|$

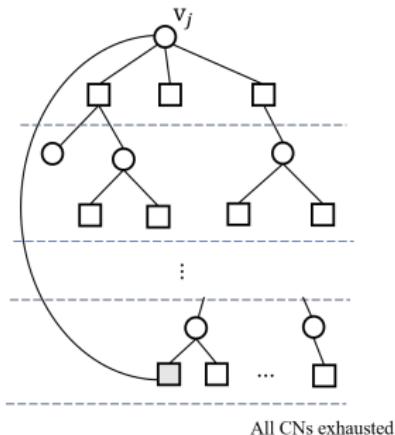
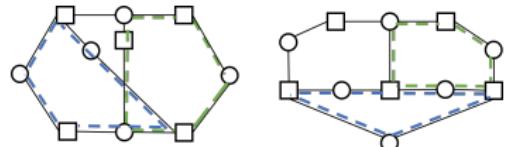


# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

-Modify the PEG algorithm

- ▶ SSs are made up of cycles
- ▶  $\mathcal{L} = \text{List of cycles of length } \leq g$
- LDPC codes to reduce  $|\text{Greedy-Set}(\mathcal{L})|$



DE (Dispersal Efficient)-PEG Algorithm

**For** each VN  $v_j$

Expand Tanner Graph in a BFS fashion

**If**  $\exists$  CNs not connected to  $v_j$

- Select a CN with min degree not connected to  $v_j$

**Else (new cycles created)**

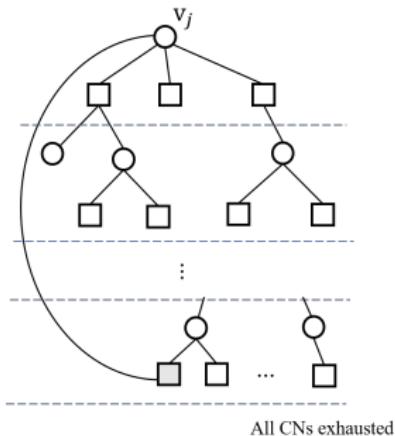
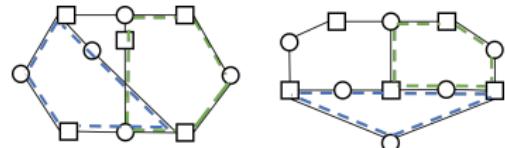
- Find CNs most distant to  $v_j$
- Select CNs with minimum degree

# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

-Modify the PEG algorithm

- ▶ SSs are made up of cycles
- ▶  $\mathcal{L} = \text{List of cycles of length } \leq g$
- LDPC codes to reduce  $|\text{Greedy-Set}(\mathcal{L})|$



DE (Dispersal Efficient)-PEG Algorithm

**For** each VN  $v_j$

Expand Tanner Graph in a BFS fashion

**If**  $\exists$  CNs not connected to  $v_j$

- Select a CN with min degree not connected to  $v_j$

**Else (new cycles created)**

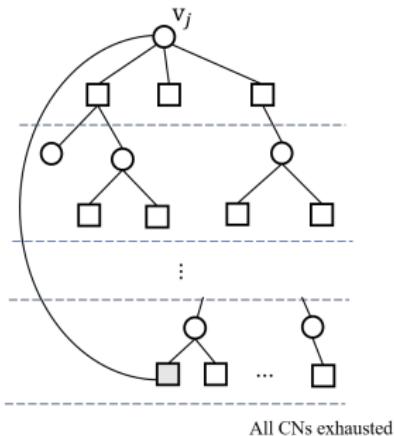
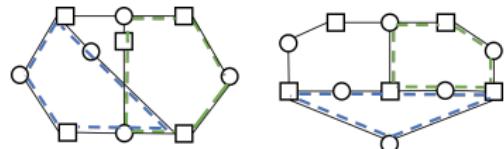
- Find CNs most distant to  $v_j$
- Select CNs with minimum degree
- Select one with minimum  $|\text{Greedy-Set}(\mathcal{L})|$

# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

-Modify the PEG algorithm

- ▶ SSs are made up of cycles
- ▶  $\mathcal{L} = \text{List of cycles of length } \leq g$
- LDPC codes to reduce  $|\text{Greedy-Set}(\mathcal{L})|$



DE (Dispersal Efficient)-PEG Algorithm

For each VN  $v_j$

Expand Tanner Graph in a BFS fashion

If  $\exists$  CNs not connected to  $v_j$

- Select a CN with min degree not connected to  $v_j$

Else (new cycles created)

- Find CNs most distant to  $v_j$
- Select CNs with minimum degree
- Select one with minimum  $|\text{Greedy-Set}(\mathcal{L})|$

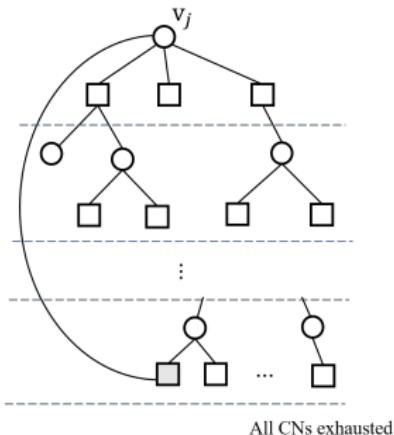
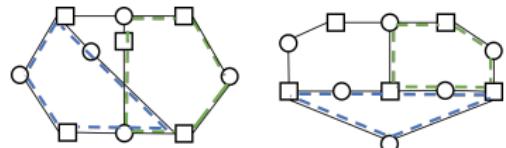
Issue: Using  $\mathcal{L}$  that contains all cycles of length  $\leq g$  does not reduce  $|\mathcal{V}|$

# LDPC Code Design Strategy

$\mathcal{S}$  = SSs of size  $< \mu$ ,  $\mathcal{V} = \text{Greedy-Set}(\mathcal{S})$ . How to design codes with small  $|\mathcal{V}|$ ?

-Modify the PEG algorithm

- ▶ SSs are made up of cycles
- ▶  $\mathcal{L} = \text{List of cycles of length } \leq g$
- LDPC codes to reduce  $|\text{Greedy-Set}(\mathcal{L})|$



DE (Dispersal Efficient)-PEG Algorithm

For each VN  $v_j$

Expand Tanner Graph in a BFS fashion

If  $\exists$  CNs not connected to  $v_j$

- Select a CN with min degree not connected to  $v_j$

Else (new cycles created)

- Find CNs most distant to  $v_j$
- Select CNs with minimum degree
- Select one with minimum  $|\text{Greedy-Set}(\mathcal{L})|$

Solution: Make  $\mathcal{L}$  contain only low EMD [Tian '04] cycles

## Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  
 $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

## Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  
 $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)

Valid Phase

$\mu$	$C^v$
17	5.116
18	4.887
19	4.658
20	4.428
21	4.276

## Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  
 $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)

Valid Phase

$\mu$	$C^v$
17	5.116
18	4.887
19	4.658
20	4.428
21	4.276

- ▶ As  $\mu$  is increased,  $C^v$  decreases,

# Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)
- ▶  $|\mathcal{V}|$ : greedy set size for  $M = 256$

Valid Phase

$\mu$	$C^v$	$ \mathcal{V} $		
		PEG	DE-PEG	
17	5.116	0	0	
18	4.887	1	0	
19	4.658	3	1	
20	4.428	7	4	
21	4.276	14	13	

- ▶ As  $\mu$  is increased,  $C^v$  decreases,

## Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)
- ▶  $|\mathcal{V}|$ : greedy set size for  $M = 256$

Valid Phase

$\mu$	$C^v$	$ \mathcal{V} $		
		PEG	DE-PEG	
17	5.116	0	0	
18	4.887	1	0	
19	4.658	3	1	
20	4.428	7	4	
21	4.276	14	13	

- ▶ As  $\mu$  is increased,  $C^v$  decreases,
- ▶ DE-PEG always results in lower  $|\mathcal{V}|$  compared to PEG

## Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)
- ▶  $|\mathcal{V}|$ : greedy set size for  $M = 256$
- ▶  $C^s$ : communication cost of secure phase of dispersal

$\mu$	$C^v$	Valid Phase		Secure Phase		
		PEG	$ \mathcal{V} $	PEG	$C^s$	
17	5.116	0	0	0	0	
18	4.887	1	0	0.037	0	
19	4.658	3	1	0.112	0.037	
20	4.428	7	4	0.262	0.149	
21	4.276	14	13	0.524	0.486	

- ▶ As  $\mu$  is increased,  $C^v$  decreases,
- ▶ DE-PEG always results in lower  $|\mathcal{V}|$  compared to PEG

## Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)
- ▶  $|\mathcal{V}|$ : greedy set size for  $M = 256$
- ▶  $C^s$ : communication cost of secure phase of dispersal

$\mu$	$C^v$	Valid Phase		Secure Phase		
		PEG	$ \mathcal{V} $ DE-PEG	PEG	$C^s$ DE-PEG	
17	5.116	0	0	0	0	
18	4.887	1	0	0.037	0	
19	4.658	3	1	0.112	0.037	
20	4.428	7	4	0.262	0.149	
21	4.276	14	13	0.524	0.486	

- ▶ As  $\mu$  is increased,  $C^v$  decreases,  $C^s$  increases.  $C^s$  for DE-PEG <  $C^s$  for PEG
- ▶ DE-PEG always results in lower  $|\mathcal{V}|$  compared to PEG

# Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)
- ▶  $|\mathcal{V}|$ : greedy set size for  $M = 256$
- ▶  $C^s$ : communication cost of secure phase of dispersal
- ▶  $C^T$ : total communication cost =  $C^v + C^s + \Delta$  (small additional overhead)

$\mu$	$C^v$	Valid Phase		Secure Phase		$C^T$	
		PEG	$ \mathcal{V} $ DE-PEG	PEG	$C^s$ DE-PEG	PEG	$C^T$ DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- ▶ As  $\mu$  is increased,  $C^v$  decreases,  $C^s$  increases.  $C^s$  for DE-PEG <  $C^s$  for PEG
- ▶ DE-PEG always results in lower  $|\mathcal{V}|$  compared to PEG

# Simulation Results: Communication Cost Reduction

System Parameters:  $N = 9000$ ,  $\beta = 0.49$ ,  $M = 256$ , Block size = 1MB,  $p_{th} = 10^{-8}$ , LDPC code rate =  $\frac{1}{2}$ ,  $\gamma = 1 - 2\beta$

- ▶  $C^v$ : communication cost of valid phase of dispersal (each node gets  $k^*(\mu)$  chunks)
- ▶  $|\mathcal{V}|$ : greedy set size for  $M = 256$
- ▶  $C^s$ : communication cost of secure phase of dispersal
- ▶  $C^T$ : total communication cost =  $C^v + C^s + \Delta$  (small additional overhead)

$\mu$	$C^v$	Valid Phase		Secure Phase		$C^T$	
		PEG	$ \mathcal{V} $ DE-PEG	PEG	$C^s$ DE-PEG	PEG	$C^T$ DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- ▶ As  $\mu$  is increased,  $C^v$  decreases,  $C^s$  increases.  $C^s$  for DE-PEG <  $C^s$  for PEG
- ▶ DE-PEG always results in lower  $|\mathcal{V}|$  compared to PEG
- ▶  $C^T$  is lowest for  $\mu = 20$ , lower for DE-PEG

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- $M_{\min}$  for PEG LDPC code is 17.

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)

---

Baseline  
 $k^*(M_{\min})$  valid  
 dispersal + PEG

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)
- ▶ Using  $k^*$ -secure dispersal protocol with  $\mu = 20$  reduces  $C^T$  from baseline:

---

Baseline  
 $k^*(M_{\min})$  valid  
dispersal + PEG

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)
- ▶ Using  $k^*$ -secure dispersal protocol with  $\mu = 20$  reduces  $C^T$  from baseline:  
Reduction for PEG: 0.425GB

---

Baseline  
 $k^*(M_{\min})$  valid  
dispersal + PEG

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)
- ▶ Using  $k^*$ -secure dispersal protocol with  $\mu = 20$  reduces  $C^T$  from baseline:  
Reduction for PEG: 0.425GB  
Reduction for DE-PEG: 0.528GB

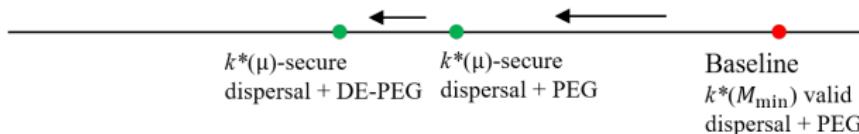
---

●  
Baseline  
 $k^*(M_{\min})$  valid  
dispersal + PEG

# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

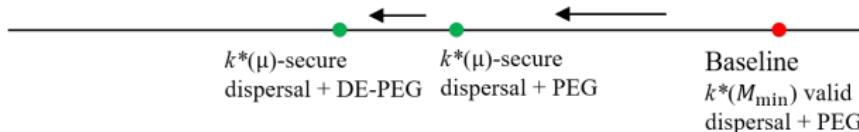
- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)
- ▶ Using  $k^*$ -secure dispersal protocol with  $\mu = 20$  reduces  $C^T$  from baseline:  
Reduction for PEG: 0.425GB  
Reduction for DE-PEG: 0.528GB



# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

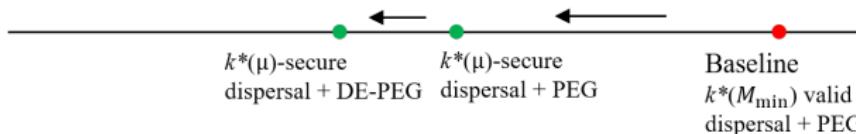
- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)
- ▶ Using  $k^*$ -secure dispersal protocol with  $\mu = 20$  reduces  $C^T$  from baseline:  
Reduction for PEG: 0.425GB  
Reduction for DE-PEG: 0.528GB
- ▶ Lower bound on  $C^T$  for  $\mu = 20$  is 4.438GB (assuming  $C^s = 0$ )



# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

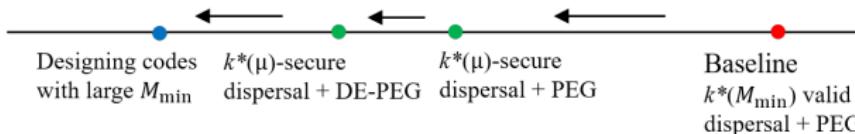
- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)
- ▶ Using  $k^*$ -secure dispersal protocol with  $\mu = 20$  reduces  $C^T$  from baseline:  
Reduction for PEG: 0.425GB  
Reduction for DE-PEG: 0.528GB
- ▶ Lower bound on  $C^T$  for  $\mu = 20$  is 4.438GB (assuming  $C^s = 0$ )  
→ equivalent to designing codes with larger minimum SS size which is hard



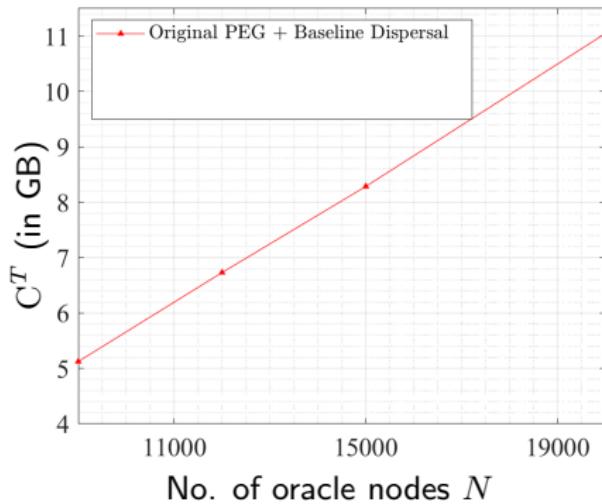
# Simulation Results: Communication Cost Reduction

$\mu$	$C^v$	$ \mathcal{V} $		$C^s$		$C^T$	
		PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	5.116	0	0	0	0	5.125	5.125
18	4.887	1	0	0.037	0	4.933	4.896
19	4.658	3	1	0.112	0.037	4.779	4.704
20	4.428	7	4	0.262	0.149	4.700	4.587
21	4.276	14	13	0.524	0.486	4.809	4.771

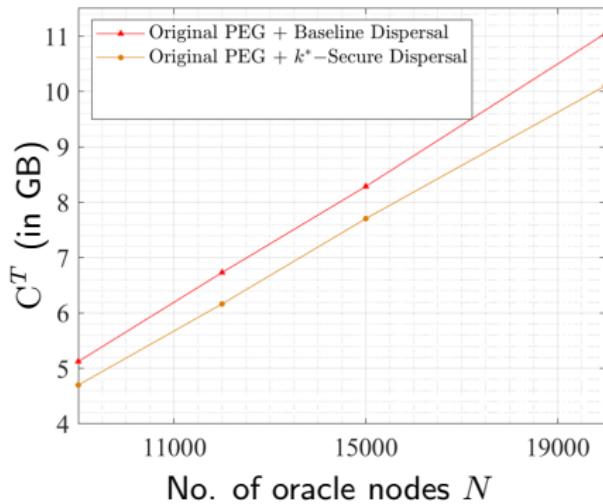
- ▶  $M_{\min}$  for PEG LDPC code is 17.
- ▶  $\mu = 17$  is considered as the baseline with  $k^*(M_{\min})$  valid dispersal protocol (no secure phase)
- ▶ Using  $k^*$ -secure dispersal protocol with  $\mu = 20$  reduces  $C^T$  from baseline:  
Reduction for PEG: 0.425GB  
Reduction for DE-PEG: 0.528GB
- ▶ Lower bound on  $C^T$  for  $\mu = 20$  is 4.438GB (assuming  $C^s = 0$ )  
→ equivalent to designing codes with larger minimum SS size which is hard  
→ corresponds to maximum 0.687GB reduction in  $C^T$ .



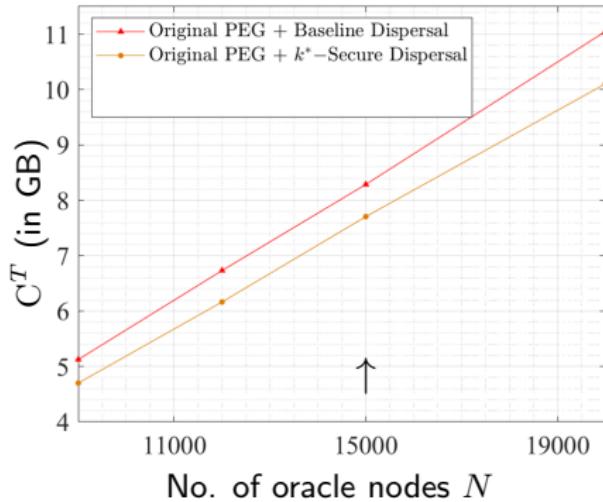
# Simulation Results



# Simulation Results



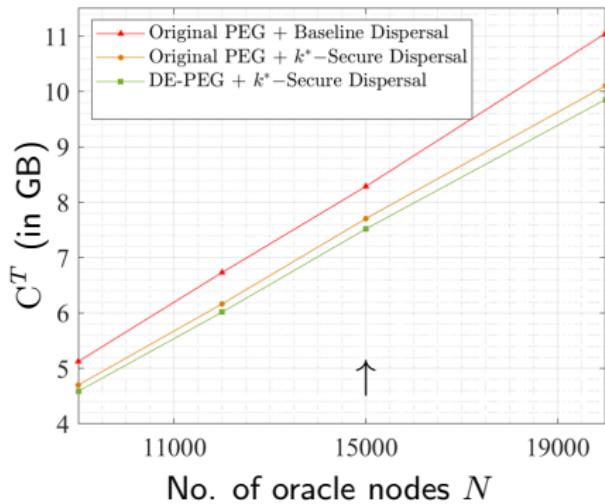
# Simulation Results



At  $N = 15000$

- Baseline  $\xrightarrow{7\% \text{ reduction}}$  PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$

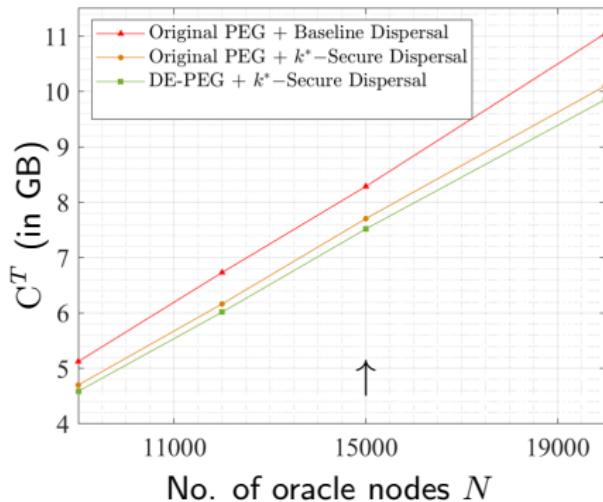
# Simulation Results



At  $N = 15000$

- Baseline  $\xrightarrow{7\% \text{ reduction}}$  PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$

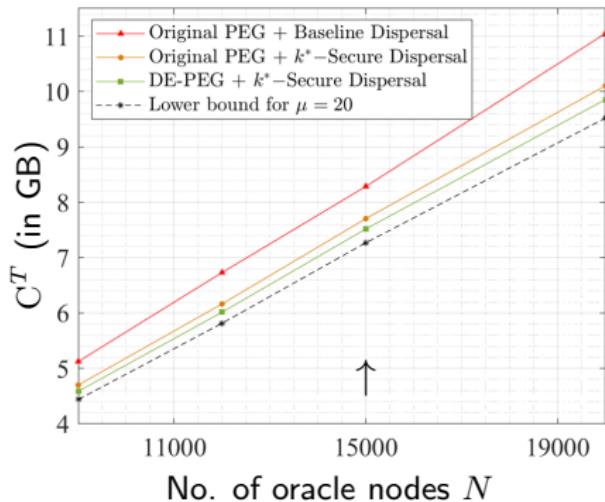
# Simulation Results



At  $N = 15000$

- ▶ Baseline  $\xrightarrow{7\% \text{ reduction}}$  PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{9.3\% \text{ reduction}}$  DE-PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$

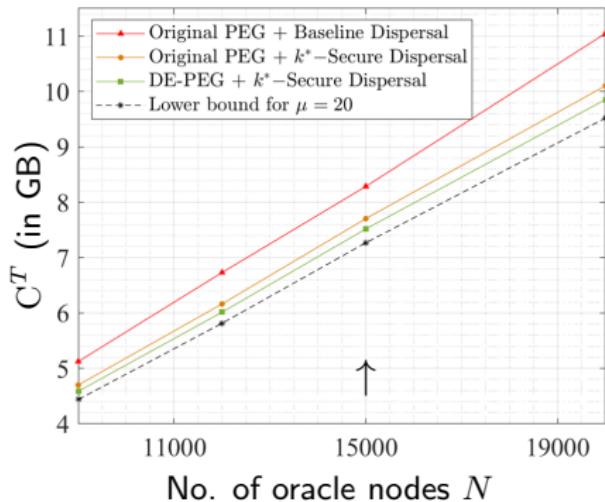
# Simulation Results



At  $N = 15000$

- ▶ Baseline  $\xrightarrow{7\% \text{ reduction}}$  PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{9.3\% \text{ reduction}}$  DE-PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$

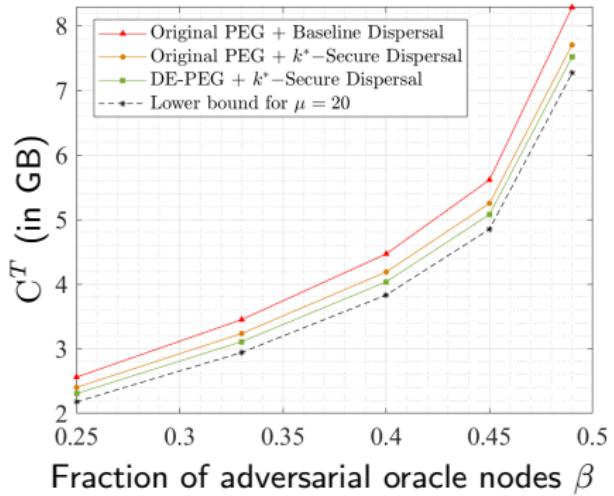
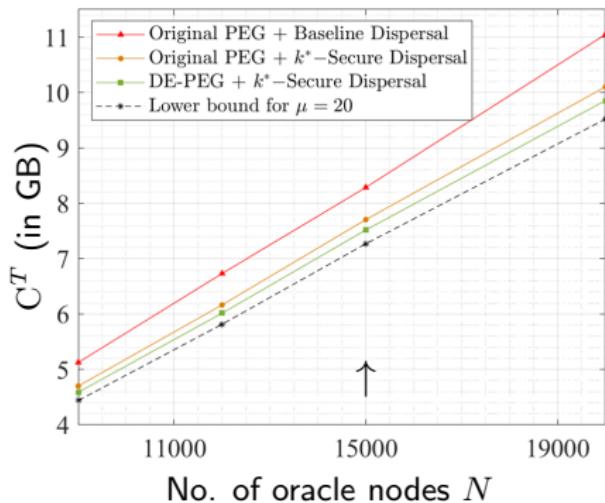
# Simulation Results



At  $N = 15000$

- ▶ Baseline  $\xrightarrow{7\% \text{ reduction}}$  PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{9.3\% \text{ reduction}}$  DE-PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{13\% \text{ reduction}}$  Lower bound for  $\mu = 20$

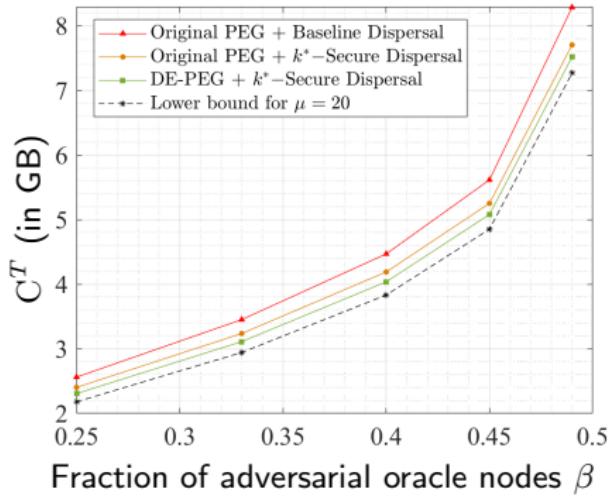
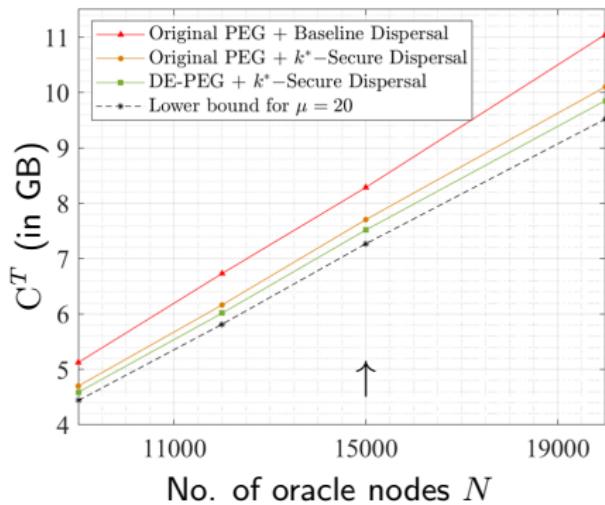
# Simulation Results



At  $N = 15000$

- ▶ Baseline  $\xrightarrow{7\% \text{ reduction}}$  PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{9.3\% \text{ reduction}}$  DE-PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{13\% \text{ reduction}}$  Lower bound for  $\mu = 20$

# Simulation Results



At  $N = 15000$

- ▶ Baseline  $\xrightarrow{7\% \text{ reduction}}$  PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{9.3\% \text{ reduction}}$  DE-PEG +  $k^*$ -secure dispersal protocol with  $\mu = 20$
- ▶ Baseline  $\xrightarrow{13\% \text{ reduction}}$  Lower bound for  $\mu = 20$
- ▶ Similar trends hold when  $C^T$  is plotted as a function of the adversary fraction  $\beta$

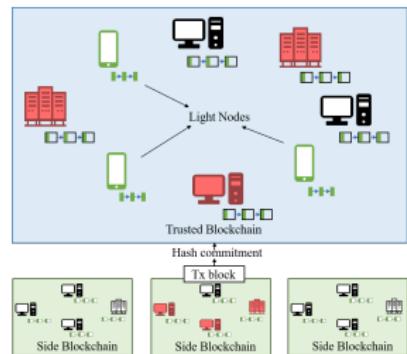
# Table of Contents

1. Background and Central Problems
2. Data Availability Attacks on Light Nodes
3. Data Availability Attacks in Side Blockchains
4. Conclusion

# Key Takeaways and Ongoing work

## ► Key Takeaways

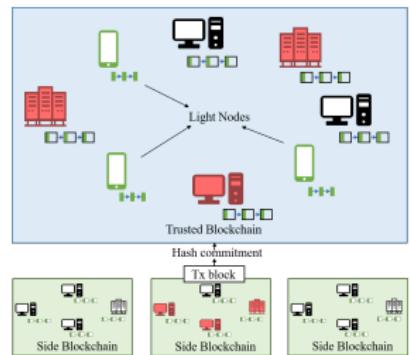
- Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:



# Key Takeaways and Ongoing work

## ► Key Takeaways

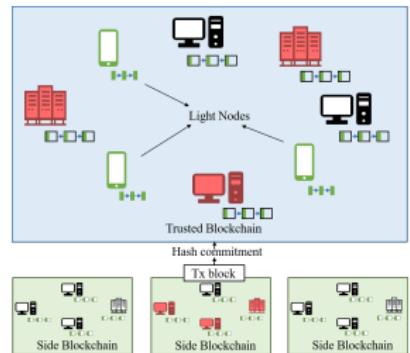
- Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
  - ↳ Adversarial erasures with light node sampling



# Key Takeaways and Ongoing work

## ► Key Takeaways

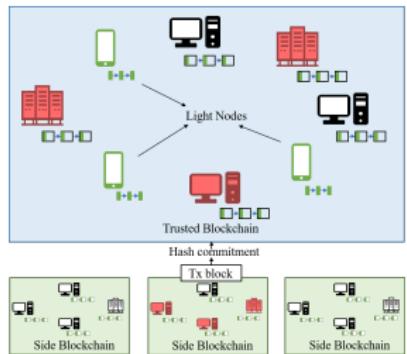
- Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
  - ↳ Adversarial erasures with light node sampling
  - ↳ Adversarial erasures with dispersal protocol



# Key Takeaways and Ongoing work

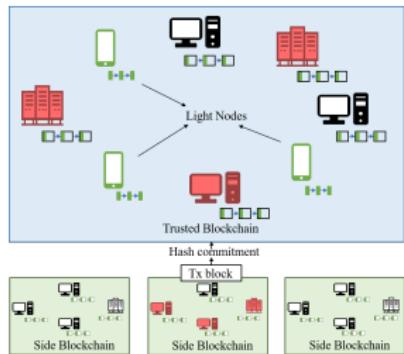
## ► Key Takeaways

- Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
  - ↳ Adversarial erasures with light node sampling
  - ↳ Adversarial erasures with dispersal protocol
- LDPC codes tailor made for these specific channels demonstrate better performance



# Key Takeaways and Ongoing work

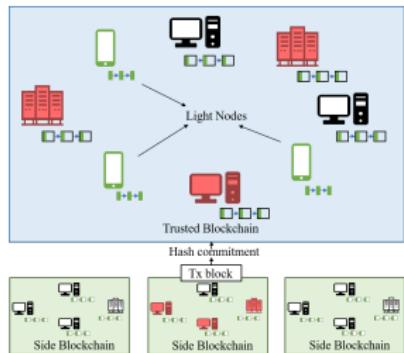
- ▶ Key Takeaways
  - Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
    - ↳ Adversarial erasures with light node sampling
    - ↳ Adversarial erasures with dispersal protocol
  - LDPC codes tailor made for these specific channels demonstrate better performance
  
- ▶ Ongoing work on DA attacks in blockchain systems
  - Considering stronger adversaries that can selectively pick a stopping set that has a lower probability of being sampled to hide in systems with light nodes



# Key Takeaways and Ongoing work

## ► Key Takeaways

- Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
  - ↳ Adversarial erasures with light node sampling
  - ↳ Adversarial erasures with dispersal protocol
- LDPC codes tailor made for these specific channels demonstrate better performance



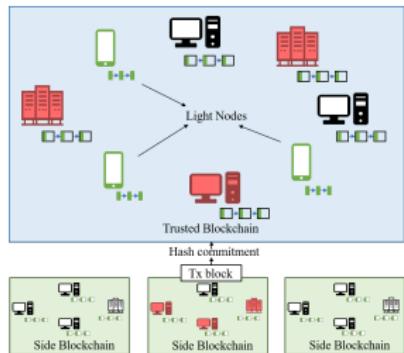
## ► Ongoing work on DA attacks in blockchain systems

- Considering stronger adversaries that can selectively pick a stopping set that has a lower probability of being sampled to hide in systems with light nodes
- Our ultimate goal is to provide optimal sampling strategies and associated code construction (e.g. LDPC, polar codes) to improve the security for a given sample complexity against such strong adversaries

# Key Takeaways and Ongoing work

## ► Key Takeaways

- Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
  - ↳ Adversarial erasures with light node sampling
  - ↳ Adversarial erasures with dispersal protocol
- LDPC codes tailor made for these specific channels demonstrate better performance



## ► Ongoing work on DA attacks in blockchain systems

- Considering stronger adversaries that can selectively pick a stopping set that has a lower probability of being sampled to hide in systems with light nodes
- Our ultimate goal is to provide optimal sampling strategies and associated code construction (e.g. LDPC, polar codes) to improve the security for a given sample complexity against such strong adversaries

Recent recognition: Best Poster award at the IEEE North American School of Information Theory 2021



Debaranab Mitra

## References

- ▶ D. Mitra, L. Tauz, and L. Dolecek, “*Concentrated Stopping Set Design for Coded Merkle Tree: Improving Security Against Data Availability Attacks in Blockchain Systems*,” in Proc. of IEEE Information Theory Workshop (ITW), Apr. 2020.  
(also available at <https://arxiv.org/abs/2010.07363>)
- ▶ D. Mitra, L. Tauz, and L. Dolecek, “*Communication-Efficient LDPC Code Design for Data Availability Oracle in Side Blockchains*,” submitted to IEEE Information Theory Workshop (ITW) 2021.  
(also available at <https://arxiv.org/abs/2105.06004>)
- (Al-Bassam '18) M. Al-Bassam, et al., “*Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities*,” arXiv preprint arXiv:1809.09044, 2018.
- (Yu '19) M. Yu, et al., “*Coded Merkle Tree: Solving Data Availability Attacks in Blockchains*,” International Conference on Financial Cryptography and Data Security, Springer, Cham, 2020.
- (Xiao '05) X.Y. Hu, et al., “*Regular and irregular progressive edge-growth tanner graphs*,” IEEE Transactions of Information Theory, vol. 51, no. 1, 2005.

## References

- (Tian '03) T. Tian, et al., "*Construction of irregular LDPC codes with low error floors,*" IEEE International Conference on Communications, May 2003.
- (Li '20) C. Li, et al., "*A Decentralized Blockchain with High Throughput and Fast Confirmation ,*" in {USENIX} Annual Technical Conference, 2020.
- (Sheng '20) P. Sheng, et al., "*ACeD: Scalable Data Availability Oracle*" arXiv preprint arXiv:2011.00102, Oct. 2020.
- (Jiao '09) X. Jiao, et al. "*Eliminating small stopping sets in irregular low-density parity-check codes,*" IEEE Communications Letters, vol. 13, no. 6, Jun. 2009.
- (He '11) Y. He, et al. "*A survey of error floor of LDPC codes,*" International ICST Conference on Communications and Networking in China (CHINACOM), Aug. 2011.
- (Tian '04) T. Tian, et al. "*Selective avoidance of cycles in irregular LDPC code construction,*" IEEE Transactions on Communications, vol. 52, no. 8, Aug. 2004.
- (Stadje '90) W. Stadje, "*The Collector's Problem with Group Drawings,*" Advances in Applied Probability, vol. 22, no. 4, JSTOR, 1990.

# Appendix

## EC-PEG vs. DE-PEG algorithms

DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

# EC-PEG vs. DE-PEG algorithms

DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

- ▶ Concentrate SSs so that by greedy sampling a larger fraction of SSs are touched

# EC-PEG vs. DE-PEG algorithms

DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

- ▶ Concentrate SSs so that by greedy sampling a larger fraction of SSs are touched
- ▶ EC-PEG algorithm designed to minimize entropy of cycle distributions to concentrate them

# EC-PEG vs. DE-PEG algorithms

## DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

- ▶ Concentrate SSs so that by greedy sampling a larger fraction of SSs are touched
- ▶ EC-PEG algorithm designed to minimize entropy of cycle distributions to concentrate them

## DA attack in Side Blockchains:

Goal: Reduce the communication cost of the dispersal process

# EC-PEG vs. DE-PEG algorithms

## DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

- ▶ Concentrate SSs so that by greedy sampling a larger fraction of SSs are touched
- ▶ EC-PEG algorithm designed to minimize entropy of cycle distributions to concentrate them

## DA attack in Side Blockchains:

Goal: Reduce the communication cost of the dispersal process

- ▶ Reduce the size  $|\mathcal{V}|$  so that the communication cost of the secure dispersal phase is reduced

# EC-PEG vs. DE-PEG algorithms

## DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

- ▶ Concentrate SSs so that by greedy sampling a larger fraction of SSs are touched
- ▶ EC-PEG algorithm designed to minimize entropy of cycle distributions to concentrate them

## DA attack in Side Blockchains:

Goal: Reduce the communication cost of the dispersal process

- ▶ Reduce the size  $|\mathcal{V}|$  so that the communication cost of the secure dispersal phase is reduced
- ▶ DE-PEG algorithm designed to minimize the size  $|\text{greedy-set}(\mathcal{L})|$  of the list  $\mathcal{L}$  of bad cycles

# EC-PEG vs. DE-PEG algorithms

## DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

- ▶ Concentrate SSs so that by greedy sampling a larger fraction of SSs are touched
- ▶ EC-PEG algorithm designed to minimize entropy of cycle distributions to concentrate them

## DA attack in Side Blockchains:

Goal: Reduce the communication cost of the dispersal process

- ▶ Reduce the size  $|\mathcal{V}|$  so that the communication cost of the secure dispersal phase is reduced
- ▶ DE-PEG algorithm designed to minimize the size  $|\text{greedy-set}(\mathcal{L})|$  of the list  $\mathcal{L}$  of bad cycles

- ▶ Although both codes are modifications of the PEG algorithm, both try to optimize different objectives based on the problem specifics

# EC-PEG vs. DE-PEG algorithms

## DA attack in systems with Light Nodes:

Goal: Reduce the probability of failure

- ▶ Concentrate SSs so that by greedy sampling a larger fraction of SSs are touched
- ▶ EC-PEG algorithm designed to minimize entropy of cycle distributions to concentrate them

## DA attack in Side Blockchains:

Goal: Reduce the communication cost of the dispersal process

- ▶ Reduce the size  $|\mathcal{V}|$  so that the communication cost of the secure dispersal phase is reduced
- ▶ DE-PEG algorithm designed to minimize the size  $|\text{greedy-set}(\mathcal{L})|$  of the list  $\mathcal{L}$  of bad cycles

- ▶ Although both codes are modifications of the PEG algorithm, both try to optimize different objectives based on the problem specifics
- ▶ Using one algorithm in the other scenario may not necessarily work

## Using EMD to select Bad Cycles

Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

## Using EMD to select Bad Cycles

Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

- ▶ EMD of stopping set = 0

## Using EMD to select Bad Cycles

Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

- ▶ EMD of stopping set = 0
- ▶ Cycles with low EMD are more likely to form stopping sets → bad cycles

## Using EMD to select Bad Cycles

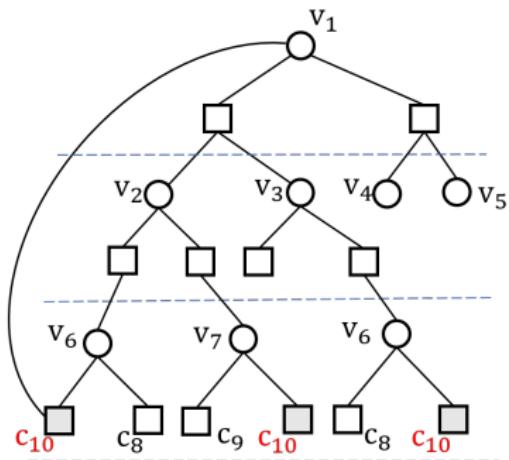
Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

- ▶ EMD of stopping set = 0
  - ▶ Cycles with low EMD are more likely to form stopping sets → bad cycles
- 
- ▶ During DE-PEG algorithm, we maintain list  $\mathcal{L}$  of bad cycles that had  $\text{EMD} \leq \text{Thr}$  during formation

## Using EMD to select Bad Cycles

Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

- ▶ EMD of stopping set = 0
- ▶ Cycles with low EMD are more likely to form stopping sets → bad cycles

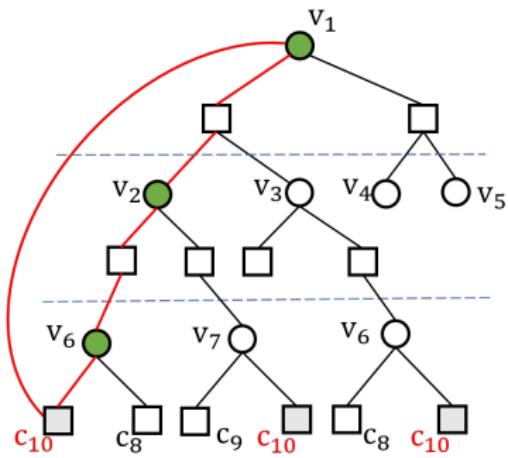


- ▶ During DE-PEG algorithm, we maintain list  $\mathcal{L}$  of bad cycles that had  $\text{EMD} \leq \text{Thr}$  during formation
- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update  $\mathcal{L}$

## Using EMD to select Bad Cycles

Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

- ▶ EMD of stopping set = 0
- ▶ Cycles with low EMD are more likely to form stopping sets → bad cycles

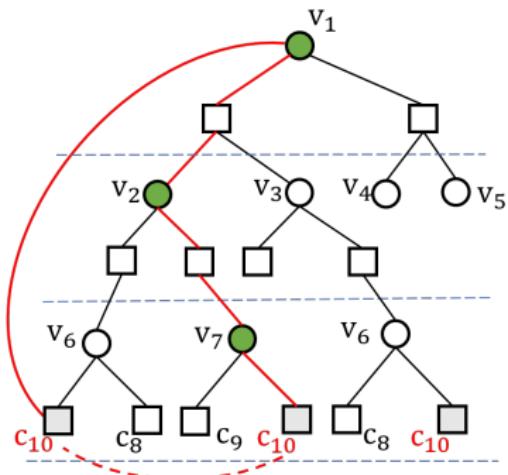


- ▶ During DE-PEG algorithm, we maintain list  $\mathcal{L}$  of bad cycles that had  $\text{EMD} \leq \text{Thr}$  during formation
- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update  $\mathcal{L}$
- If  $\text{EMD}(v_1, v_2, v_6) \leq \text{Thr}$ :  $\mathcal{L} = \mathcal{L} \cup \{v_1, v_2, v_6\}$

## Using EMD to select Bad Cycles

Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

- ▶ EMD of stopping set = 0
- ▶ Cycles with low EMD are more likely to form stopping sets → bad cycles

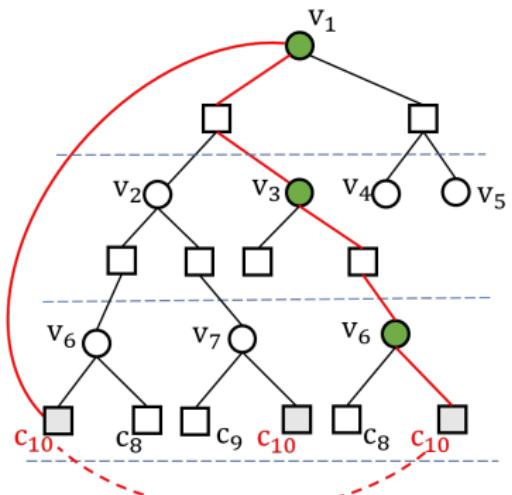


- ▶ During DE-PEG algorithm, we maintain list  $\mathcal{L}$  of bad cycles that had  $\text{EMD} \leq \text{Thr}$  during formation
- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update  $\mathcal{L}$ 
  - If  $\text{EMD}(v_1, v_2, v_6) \leq \text{Thr}$ :  $\mathcal{L} = \mathcal{L} \cup \{v_1, v_2, v_6\}$
  - If  $\text{EMD}(v_1, v_2, v_7) \leq \text{Thr}$ :  $\mathcal{L} = \mathcal{L} \cup \{v_1, v_2, v_7\}$

## Using EMD to select Bad Cycles

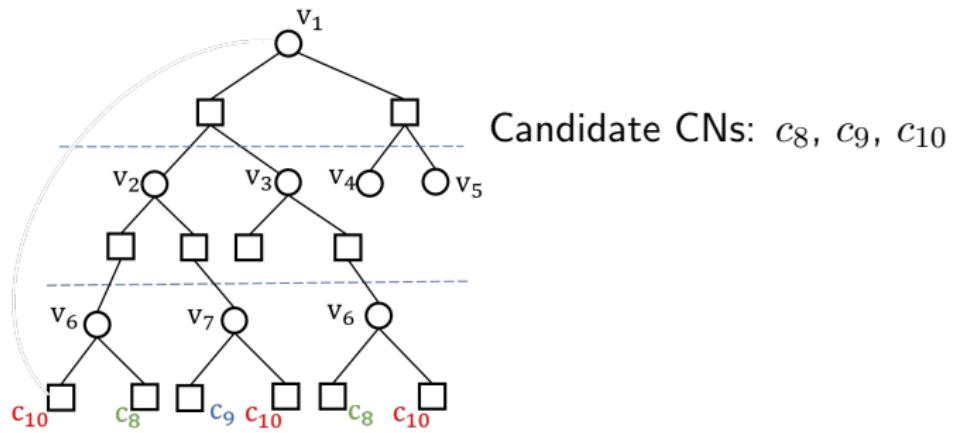
Extrinsic Message Degree (EMD) of a set of VNs := no. of CN neighbours singly connected to the set of VNs [Tian '04]

- ▶ EMD of stopping set = 0
- ▶ Cycles with low EMD are more likely to form stopping sets → bad cycles

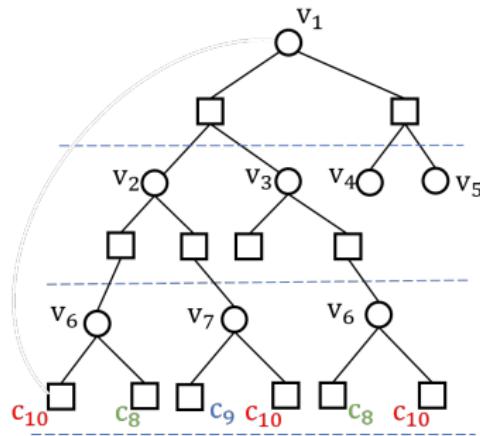


- ▶ During DE-PEG algorithm, we maintain list  $\mathcal{L}$  of bad cycles that had  $\text{EMD} \leq \text{Thr}$  during formation
- ▶ Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update  $\mathcal{L}$ 
  - If  $\text{EMD}(v_1, v_2, v_6) \leq \text{Thr}$ :  $\mathcal{L} = \mathcal{L} \cup \{v_1, v_2, v_6\}$
  - If  $\text{EMD}(v_1, v_2, v_7) \leq \text{Thr}$ :  $\mathcal{L} = \mathcal{L} \cup \{v_1, v_2, v_7\}$
  - If  $\text{EMD}(v_1, v_3, v_6) \leq \text{Thr}$ :  $\mathcal{L} = \mathcal{L} \cup \{v_1, v_3, v_6\}$

## DE-PEG Algorithm: CN Selection Procedure



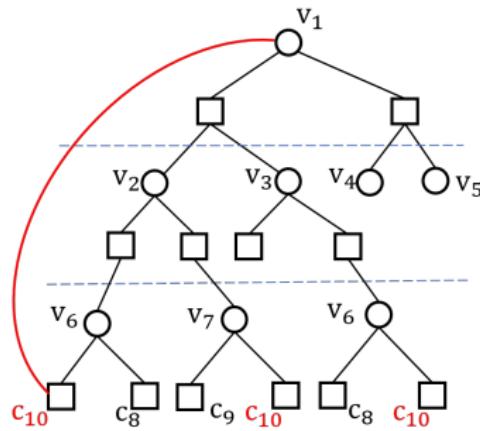
## DE-PEG Algorithm: CN Selection Procedure



Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN

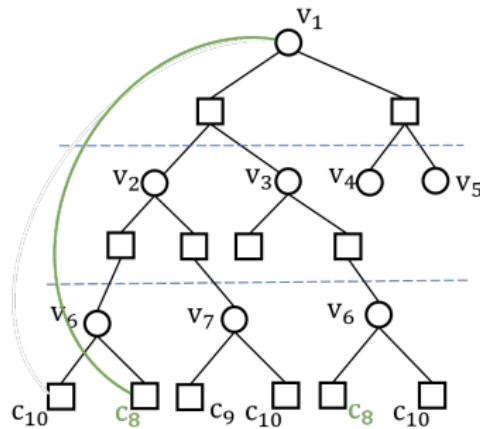
## DE-PEG Algorithm: CN Selection Procedure



Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}$

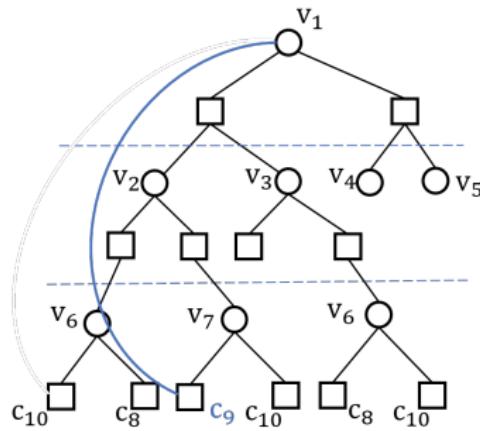
## DE-PEG Algorithm: CN Selection Procedure



Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}, \mathcal{L}_{cycles}$

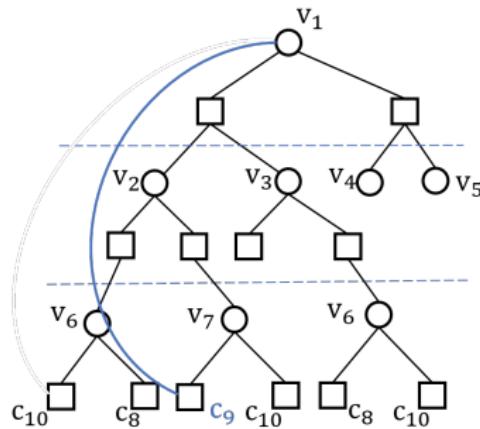
## DE-PEG Algorithm: CN Selection Procedure



Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}, \mathcal{L}_{cycles}, \mathcal{L}_{cycles}$

## DE-PEG Algorithm: CN Selection Procedure

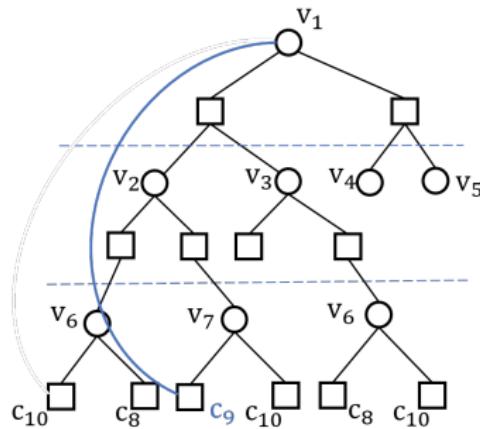


Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}, \mathcal{L}_{cycles}, \mathcal{L}_{cycles}$

CN selection procedure:

## DE-PEG Algorithm: CN Selection Procedure



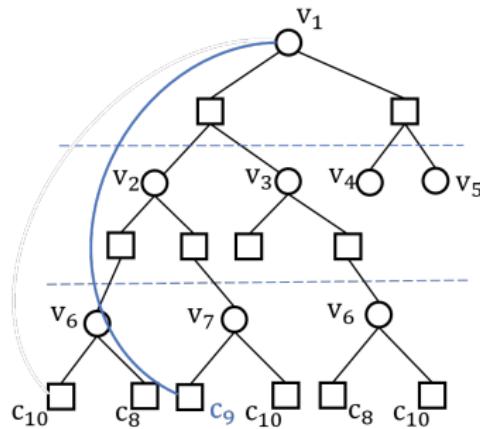
Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}, \mathcal{L}_{cycles}, \mathcal{L}_{cycles}$

CN selection procedure:

Select CN that results in minimum  $|\text{greedy-set}(\mathcal{L} \cup \mathcal{L}_{cycles}, v_1)|$

## DE-PEG Algorithm: CN Selection Procedure



Candidate CNs:  $c_8, c_9, c_{10}$

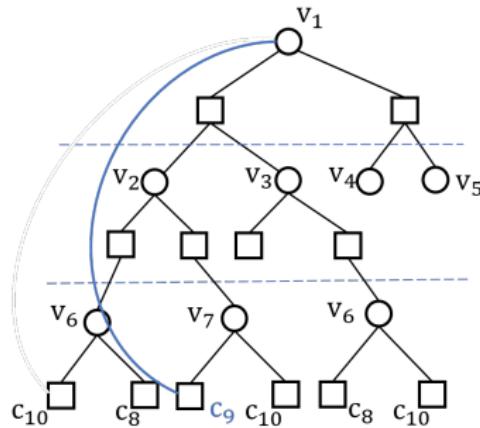
- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}, \mathcal{L}_{cycles}, \mathcal{L}_{cycles}$

CN selection procedure:

Select CN that results in minimum  $|\text{greedy-set}(\mathcal{L} \cup \mathcal{L}_{cycles}, v_1)|$

Remark:

## DE-PEG Algorithm: CN Selection Procedure



Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}, \mathcal{L}_{cycles}, \mathcal{L}_{cycles}$

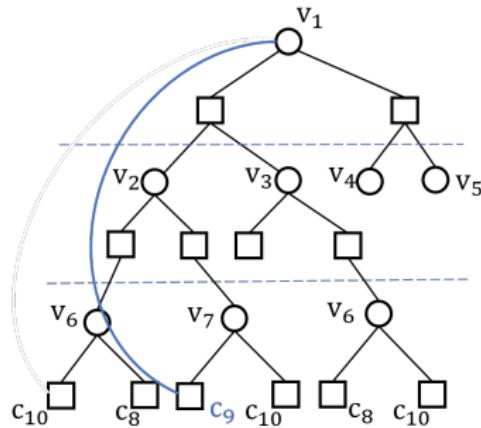
CN selection procedure:

Select CN that results in minimum  $|\text{greedy-set}(\mathcal{L} \cup \mathcal{L}_{cycles}, v_1)|$

Remark:

- ▶  $\text{greedy-set}(, v)$  ignores the VN  $v$  during the greedy selection procedure

## DE-PEG Algorithm: CN Selection Procedure



Candidate CNs:  $c_8, c_9, c_{10}$

- ▶ For each CN candidate, find the list of new cycles  $\mathcal{L}_{cycles}$  formed due to the CN
- ▶  $\mathcal{L}_{cycles}, \mathcal{L}_{cycles}, \mathcal{L}_{cycles}$

CN selection procedure:

Select CN that results in minimum  $|\text{greedy-set}(\mathcal{L} \cup \mathcal{L}_{cycles}, v_1)|$

Remark:

- ▶  $\text{greedy-set}(\cdot, v)$  ignores the VN  $v$  during the greedy selection procedure
- ▶ Ignoring  $v_1$  (the current VN) allows to better distinguish between the CN candidates as  $v_1$  is part of all new cycles formed for each CN candidate