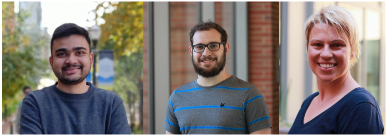
Communication-Efficient LDPC Code Design for Data Availability Oracle in Side Blockchains

Debarnab Mitra, Lev Tauz, and Lara Dolecek

Electrical and Computer Engineering
University of California, Los Angeles

ITW 2021

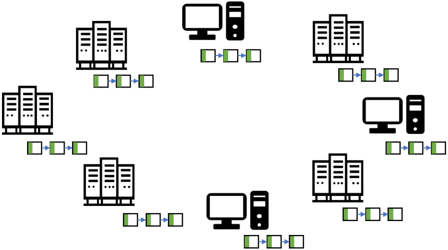


UCLA Samueli School of Engineering

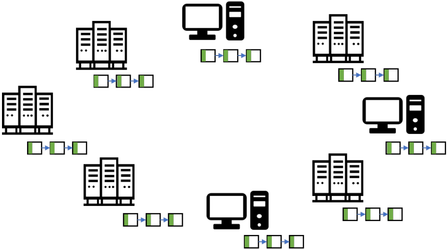
Blockchain

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

Central Problem: Poor Throughput and Latency

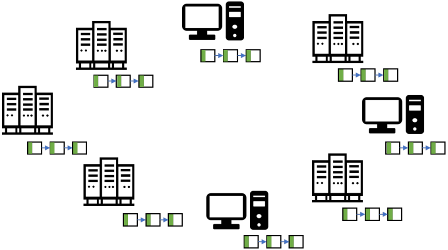


Central Problem: Poor Throughput and Latency



- Ledger of transaction blocks maintained by a network of nodes

Central Problem: Poor Throughput and Latency

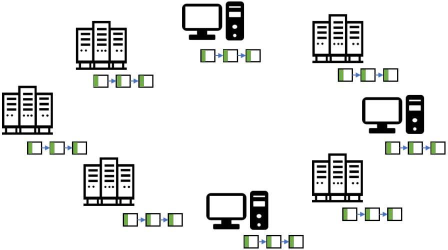


- Ledger of transaction blocks maintained by a network of nodes

Metrics:

- Transaction throughout: number of transactions processed in the system per second

Central Problem: Poor Throughput and Latency

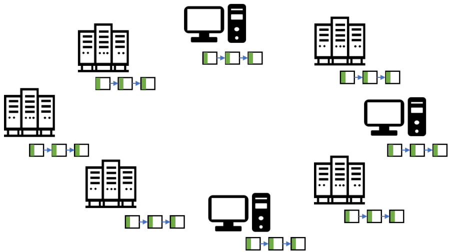


- Ledger of transaction blocks maintained by a network of nodes

Metrics:

- Transaction throughout: number of transactions processed in the system per second
- Confirmation latency: amount of time required for a transaction to be confirmed and deemed trustworthy

Central Problem: Poor Throughput and Latency

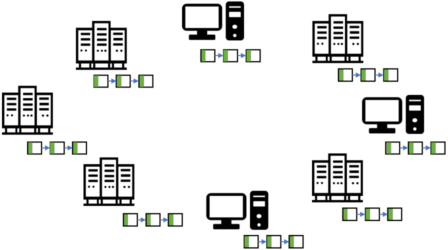


- Ledger of transaction blocks maintained by a network of nodes

	Transaction throughput	Confirmation Latency
Bitcoin Ethereum		

[Li '20]

Central Problem: Poor Throughput and Latency

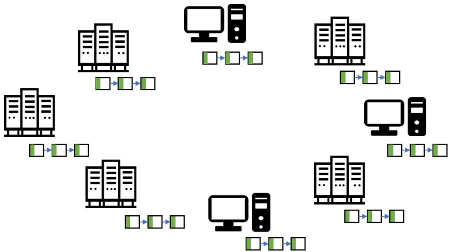


- Ledger of transaction blocks maintained by a network of nodes

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

[Li '20]

Central Problem: Poor Throughput and Latency

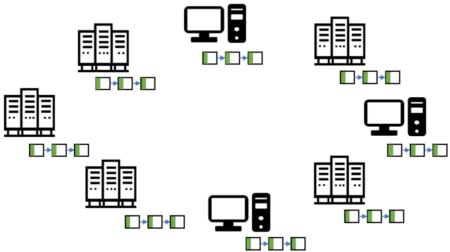


- Ledger of transaction blocks maintained by a network of nodes

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

[Li '20]

Central Problem: Poor Throughput and Latency

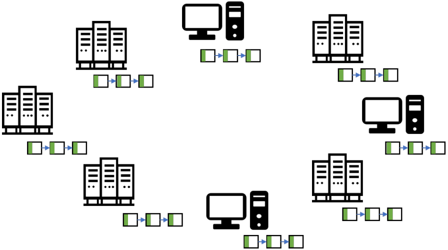


- Ledger of transaction blocks maintained by a network of nodes

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

[Li '20]

Central Problem: Poor Throughput and Latency



- Ledger of transaction blocks maintained by a network of nodes

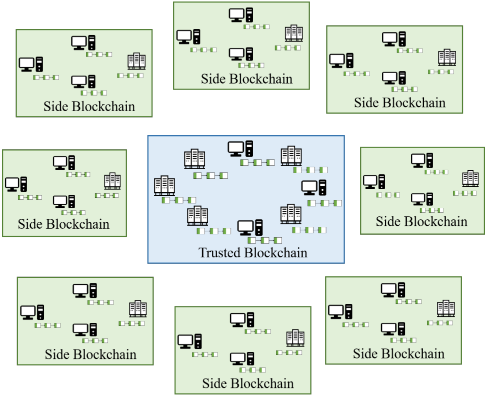
	Transaction throughput	Confirmation Latency
Bitcoin	5-7 transactions/s	hours
Ethereum	30 transactions/s	tens of minutes

[Li '20]

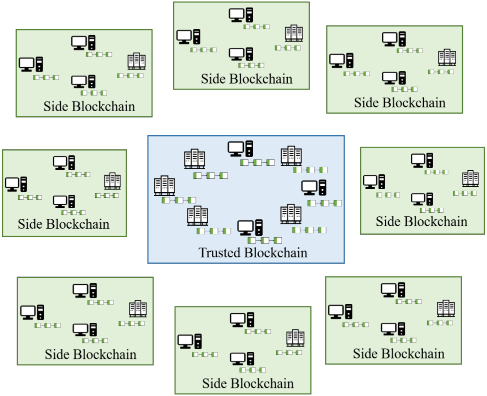
Contrast: Visa processes more than 10,000 transactions/s[3]

[3] https://usa.visa.com

Solution: Running Side Blockchains

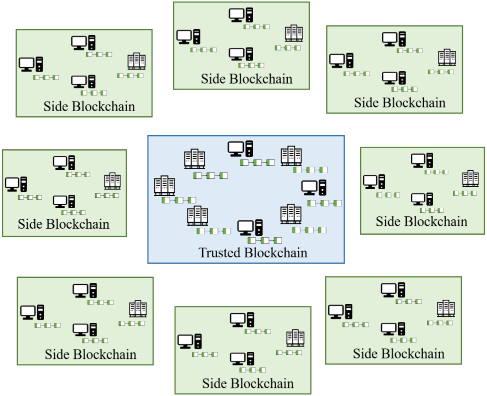


Solution: Running Side Blockchains



Side Blockchain:

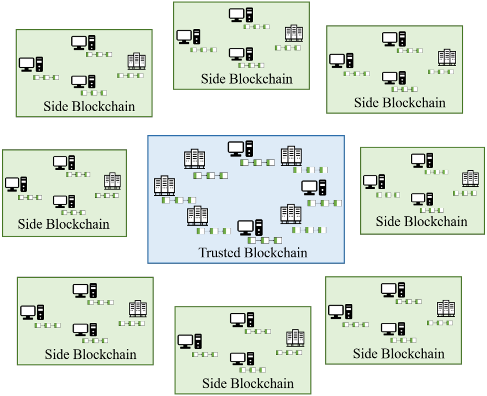
Solution: Running Side Blockchains



Side Blockchain:

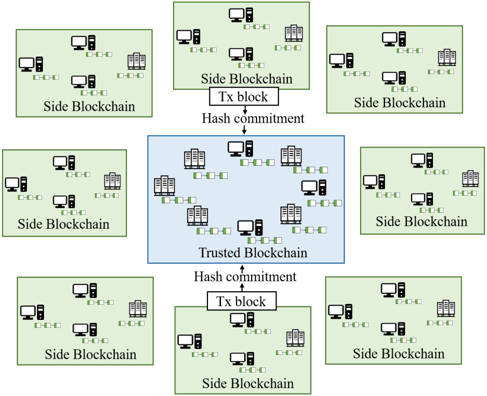
- Smaller blockchain systems

Solution: Running Side Blockchains



Side Blockchain nodes:

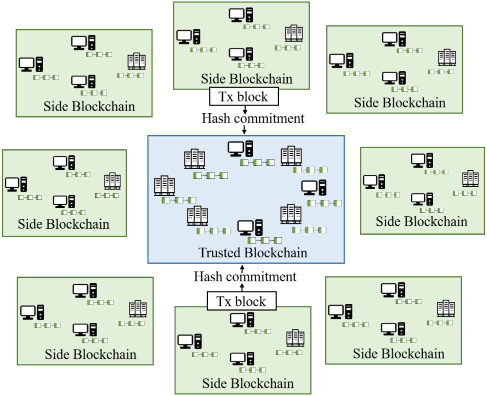
Solution: Running Side Blockchains



Side Blockchain nodes:

- Push hash commitment of their block to the trusted blockchain

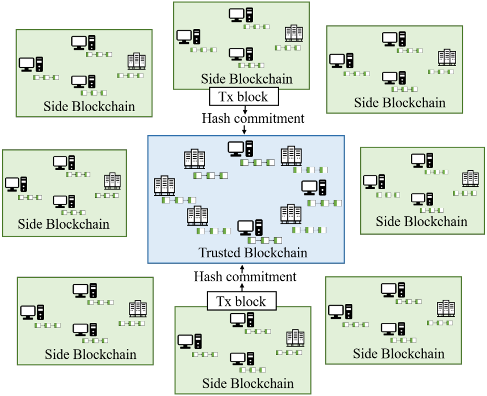
Solution: Running Side Blockchains



Side Blockchain nodes:

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

Solution: Running Side Blockchains

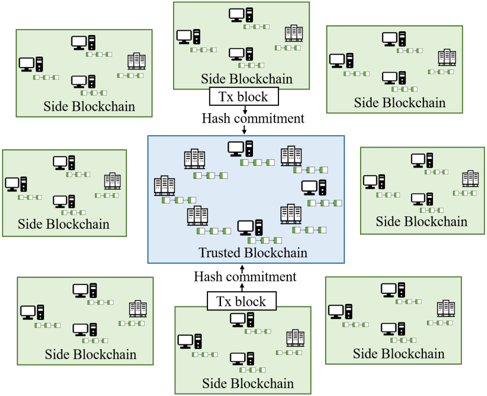


Side Blockchain nodes:

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

Trusted Blockchain:

Solution: Running Side Blockchains



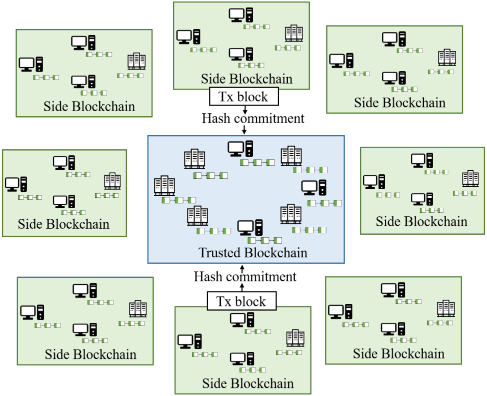
Side Blockchain nodes:

- Push hash commitment of their block to the 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



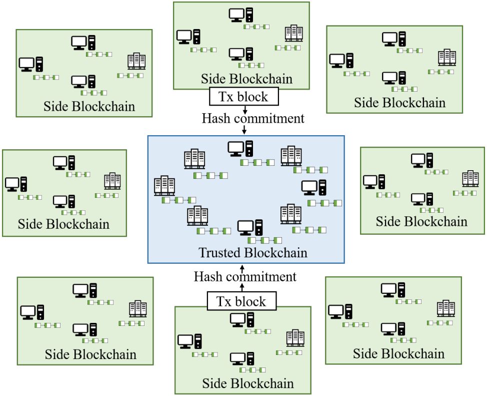
Side Blockchain nodes:

- Push hash commitment of their block to the 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



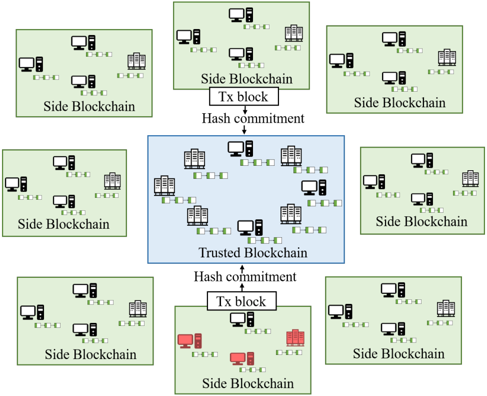
Side Blockchain nodes:

- Push hash commitment of their block to the 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

Solution: Running Side Blockchains



Side Blockchain nodes:

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

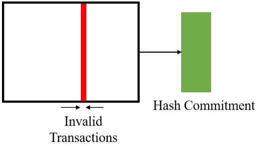
Trusted Blockchain:

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

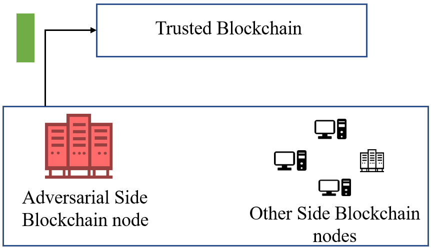
Issue: Side Blockchains with a majority of dishonest nodes are vulnerable to data availability attacks [Sheng '20]

Data Availability (DA) Attack in Side Blockchains

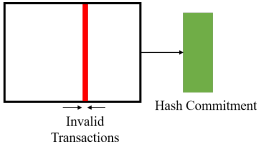
Adversary creates an invalid block



Data Availability (DA) Attack in Side Blockchains



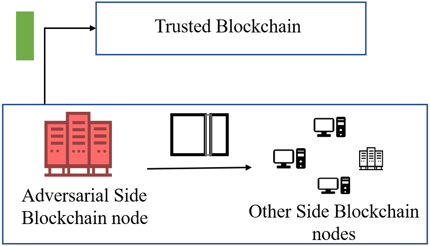
Adversary creates an invalid block



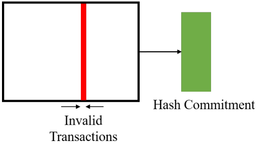
Adversarial Side Blockchain node:

- Pushes hash commitment to the trusted blockchain

Data Availability (DA) Attack in Side Blockchains



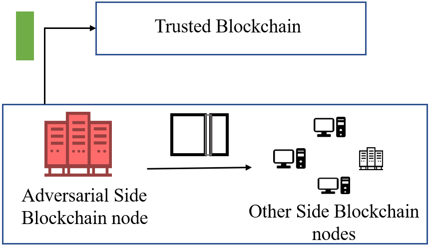
Adversary creates an invalid block



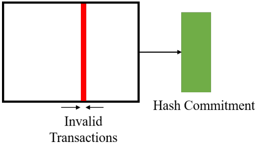
Adversarial Side Blockchain node:

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

Data Availability (DA) Attack in Side Blockchains



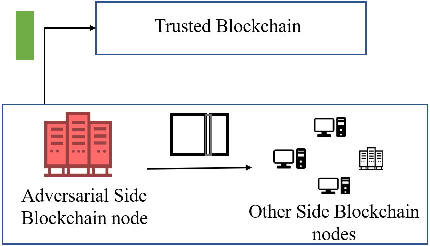
Adversary creates an invalid block



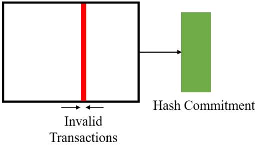
Adversarial Side Blockchain node:

- 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 (DA) Attack in Side Blockchains



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:

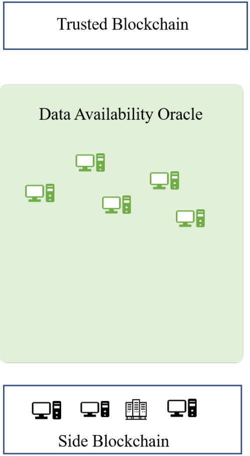
- Pushes hash commitment to the trusted blockchain
- Full block not available to other side blockchain nodes
- The invalid block becomes part of the transaction ordering in the trusted blockchain

Solution using a Data Availability Oracle

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

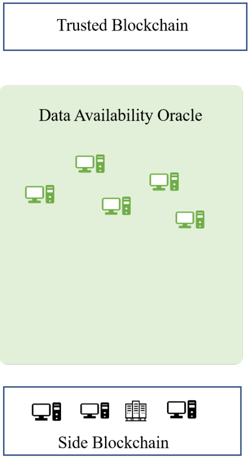
Solution using a Data Availability Oracle

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



Solution using a Data Availability Oracle

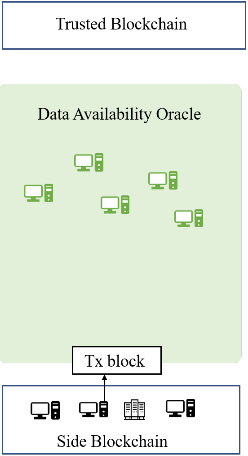
An oracle layer was introduced to ensure data availability [Sheng '20]



Oracle layer goal

Solution using a Data Availability Oracle

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

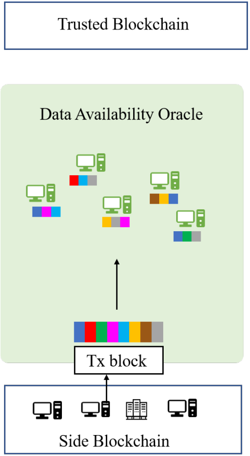


Oracle layer goal

- Accept a Tx block

Solution using a Data Availability Oracle

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

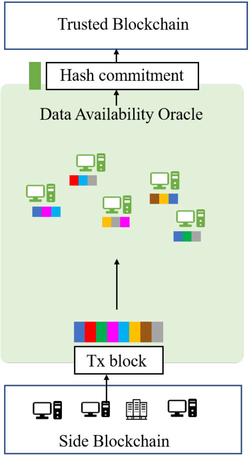


Oracle layer goal

- 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]

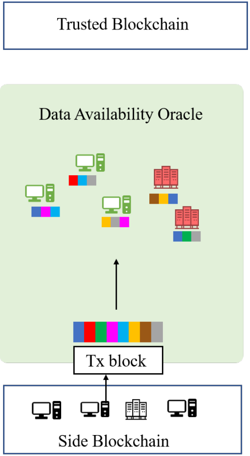


Oracle layer goal

- 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]

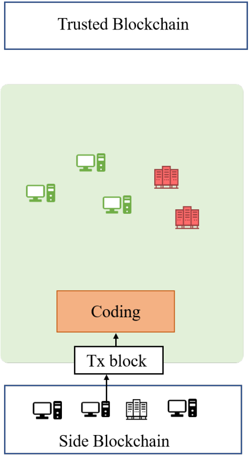


Oracle layer goal

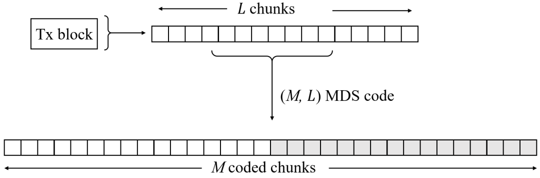
- Accept a Tx block
- Collectively and efficiently store chunks of the Tx block (to guarantee availability)
- Push the Tx block's hash commitment iff the block is available
- Oracle nodes can be malicious (honest majority)

Solution using a Data Availability Oracle

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

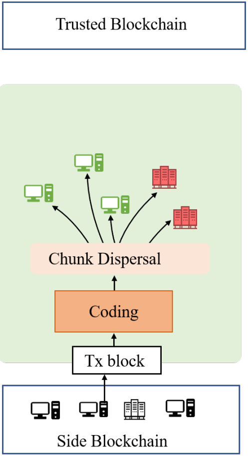


- Transaction block: chunked and coded

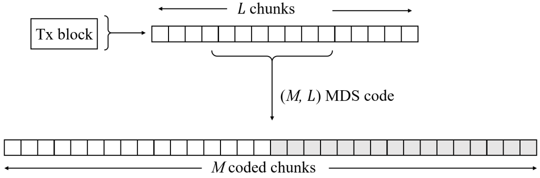


Solution using a Data Availability Oracle

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

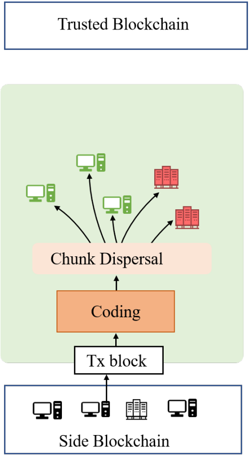


- Transaction block: chunked and coded
- Coded chunks dispersed among N oracle nodes

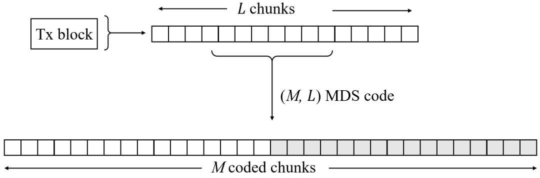


Solution using a Data Availability Oracle

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



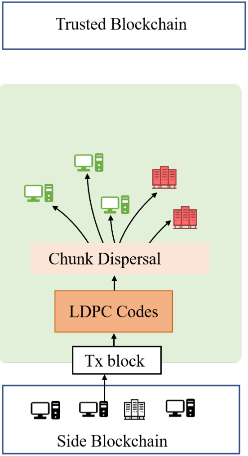
- Transaction block: chunked and coded
- Coded chunks dispersed among N oracle nodes



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

Solution using a Data Availability Oracle

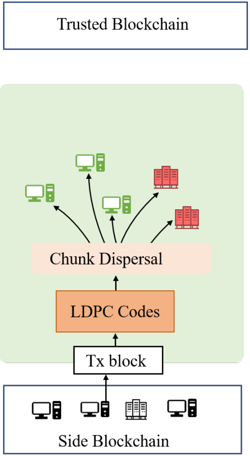
An oracle layer was introduced to ensure data availability [Sheng '20]



Low-Density Parity Check (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]

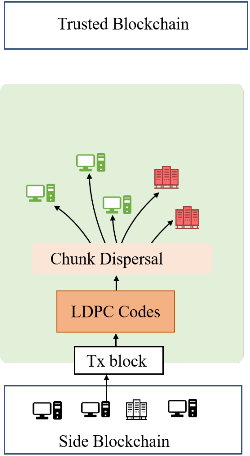


Low-Density Parity Check (LDPC) codes are used to code the Tx block

- Linear decoding complexity using a peeling decoder

Solution using a Data Availability Oracle

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

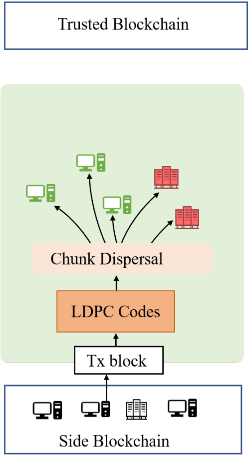


Low-Density Parity Check (LDPC) codes are used to code the Tx block

- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks

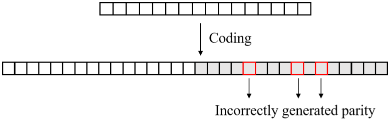
Solution using a Data Availability Oracle

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



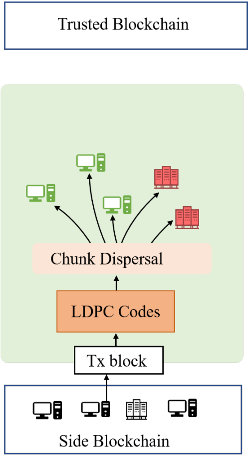
Low-Density Parity Check (LDPC) codes are used to code the Tx block

- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks



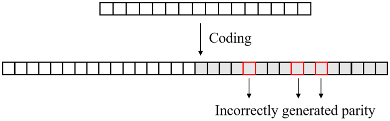
Solution using a Data Availability Oracle

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



Low-Density Parity Check (LDPC) codes are used to code the Tx block

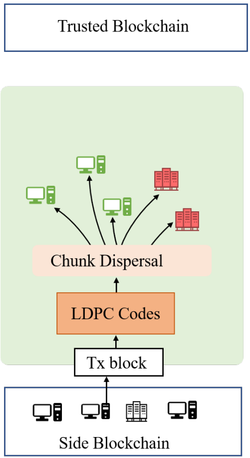
- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks



- Adversary sends incorrectly coded block to oracle nodes

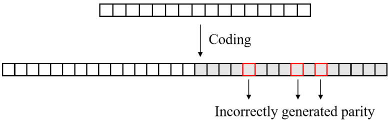
Solution using a Data Availability Oracle

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



Low-Density Parity Check (LDPC) codes are used to code the Tx block

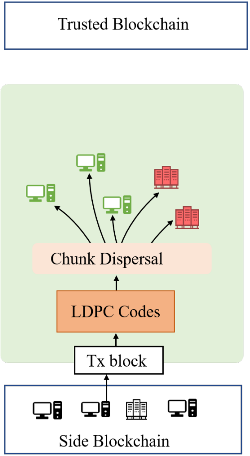
- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks



- Adversary sends incorrectly coded block to oracle nodes
- Incorrect coding proof size:
  $\mathcal{O}$(sparsity of parity check equations)

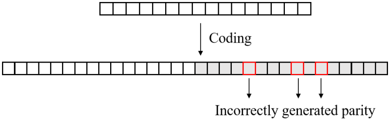
Solution using a Data Availability Oracle

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



Low-Density Parity Check (LDPC) codes are used to code the Tx block

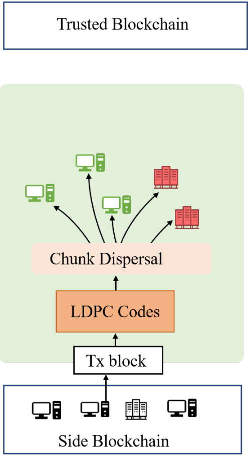
- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks



- Adversary sends incorrectly coded block to oracle nodes
- Incorrect coding proof size:
  $\mathcal{O}$(sparsity of parity check equations)
- LDPC code have small incorrect coding proof size due to sparse parity check matrix

Solution using a Data Availability Oracle

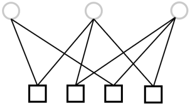
An oracle layer was introduced to ensure data availability [Sheng '20]



Low-Density Parity Check (LDPC) codes are used to code the Tx block

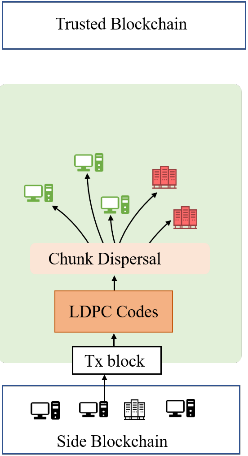
- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks

Issues with LDPC codes: small stopping sets



Solution using a Data Availability Oracle

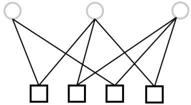
An oracle layer was introduced to ensure data availability [Sheng '20]



Low-Density Parity Check (LDPC) codes are used to code the Tx block

- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks

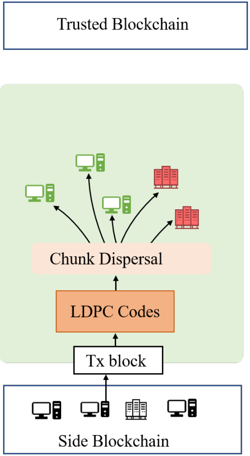
Issues with LDPC codes: small stopping sets



- 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

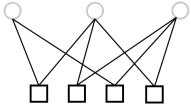
An oracle layer was introduced to ensure data availability [Sheng '20]



Low-Density Parity Check (LDPC) codes are used to code the Tx block

- Linear decoding complexity using a peeling decoder
- Good performance under incorrect coding attacks

Issues with LDPC codes: small stopping sets

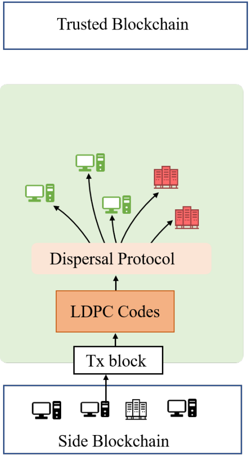


- If VNs corresponding to a small stopping set are hidden from the oracle nodes, original block cannot be decoded back by a peeling decoder
- In [Sheng '20] randomly constructed LDPC codes were used which provides a guarantee on the minimum stopping set size w.h.p

Solution using a Data Availability Oracle

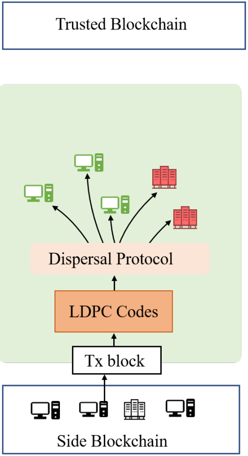
An oracle layer was introduced to ensure data availability [Sheng '20]

Dispersal Protocol



Solution using a Data Availability Oracle

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

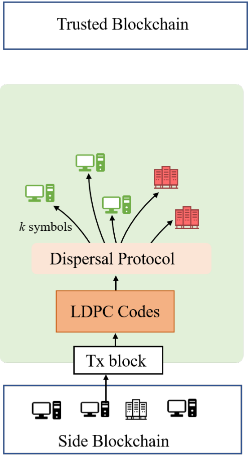


Dispersal Protocol

- 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]

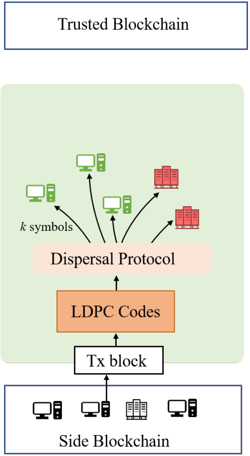


Dispersal Protocol

- 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]

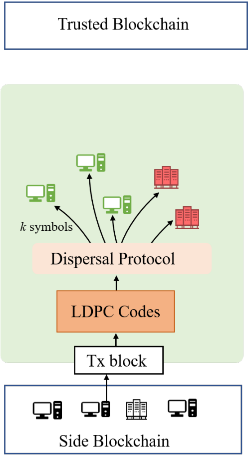


Dispersal Protocol

- Rule about which oracle node stores which coded chunks
- Specifies k coded symbols that each oracle node should receive
- $M_{\text{min}} :=$ 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]

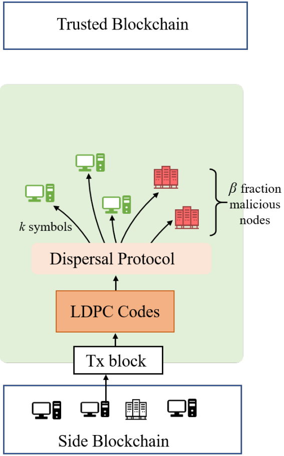


Dispersal Protocol

- 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 γ 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]

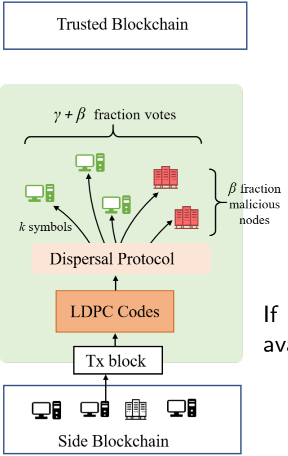


Dispersal Protocol

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



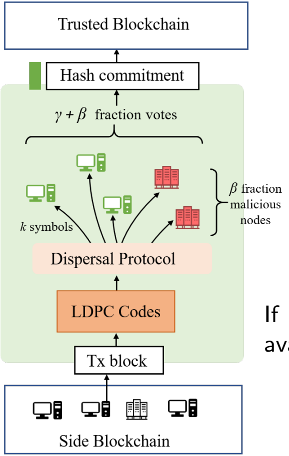
Dispersal Protocol

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

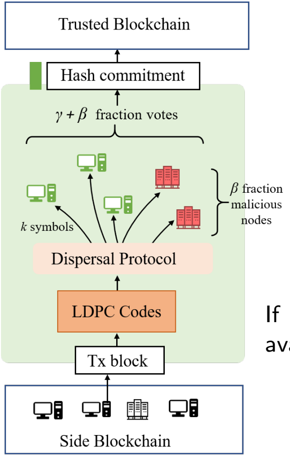
- Rule about which oracle node stores which coded chunks
- Specifies k coded symbols that each oracle node should receive
- $M_{\min}$:= minimum stopping set size of the LDPC code of block length M
- Every γ fraction of nodes should receive at least $M - M_{\min} + 1$ coded chunks

• β := 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

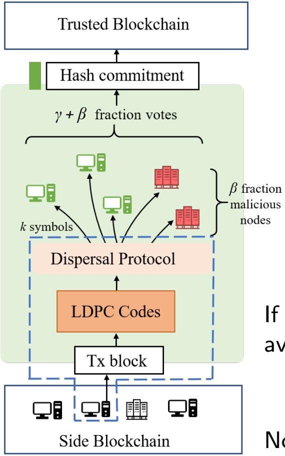
- Rule about which oracle node stores which coded chunks
- Specifies k coded symbols that each oracle node should receive
- M_{min} := minimum stopping set size of the LDPC code of block length M
- Every γ fraction of nodes should receive at least $M - M_{\text{min}} + 1$ coded chunks
- β := fraction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

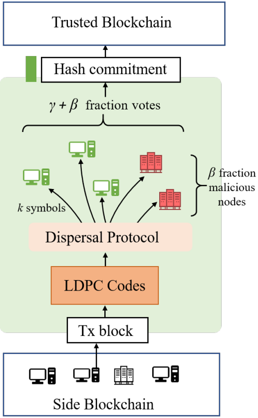
If more than $\gamma + \beta$ fraction of nodes vote that the block is available, then the hash commitment is pushed.

The oracle guarantees block availability

Note: Side blockchain nodes perform LDPC encoding and dispersal

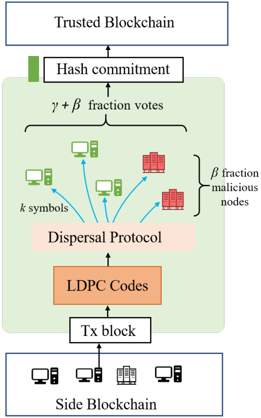
Design Objective: Minimize Communication Cost

- Communication cost:



Design Objective: Minimize Communication Cost

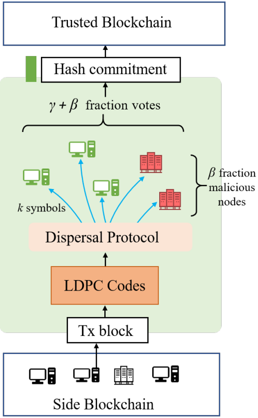
- Communication cost: amount of data communicated to oracle nodes during dispersal



Design Objective: Minimize Communication Cost

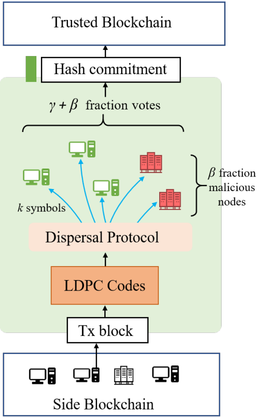
- Communication cost: amount of data communicated to oracle nodes during dispersal

Dispersal protocol in prior work [Sheng '20]:



Design Objective: Minimize Communication Cost

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

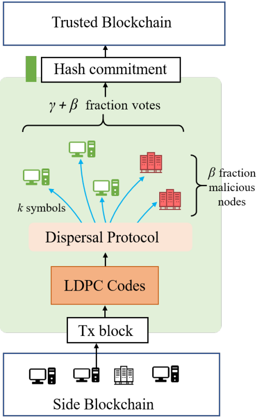


Dispersal protocol in prior work [Sheng '20]:

- Every γ 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]:

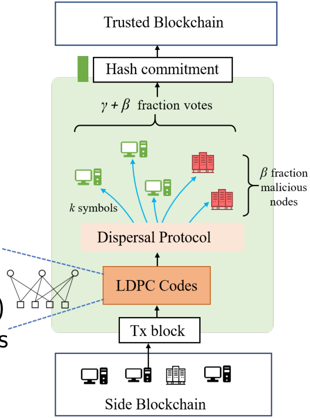
- Every γ fraction of oracle nodes receives $\geq M - M_{\text{min}} + 1$ coded chunks
  - designed randomly (sampling with replacement)

Design Objective: Minimize Communication Cost

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

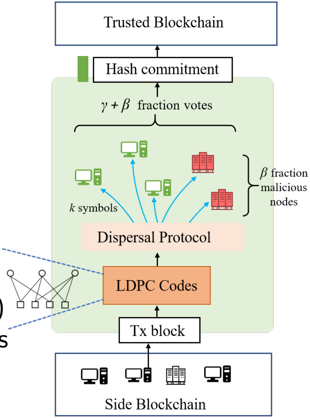
Dispersal protocol in prior work [Sheng '20]:

- Every γ 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 γ 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 $\uparrow$

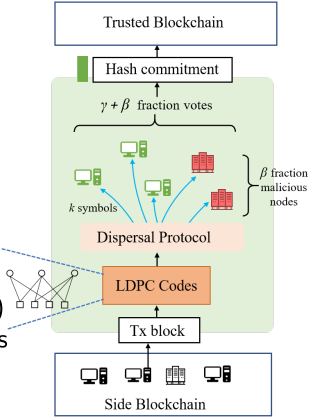
Simply design LDPC codes with large minimum stopping set size $M_{\min}$?

Design Objective: Minimize Communication Cost

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

Dispersal protocol in prior work [Sheng '20]:

- Every γ 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 minimum stopping set size $M_{\min}$?

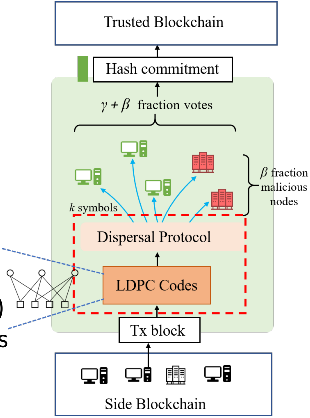
$\rightarrow$ known hard problem [Jiao '09], [He '11]

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 γ 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 $\uparrow$



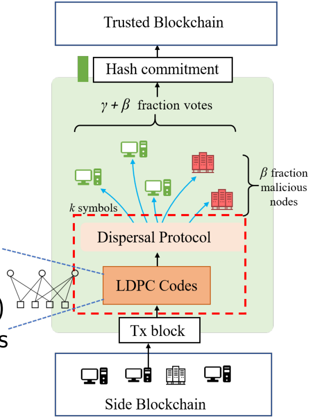
Simply design LDPC codes with large minimum stopping set size $M_{\min}$?
→ known hard problem [Jiao '09], [He '11]

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 γ 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

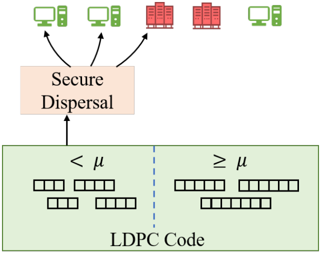
1. Secure Phase

Dispersal Protocol Design

Our dispersal strategy is a two step protocol

1. Secure Phase

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



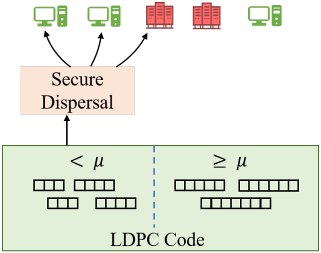
Dispersal Protocol Design

Our dispersal strategy is a two step protocol

1. Secure Phase

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

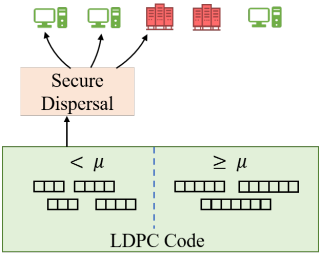
   - Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur



Dispersal Protocol Design

Our dispersal strategy is a two step protocol

1. Secure Phase
   All small stopping sets (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

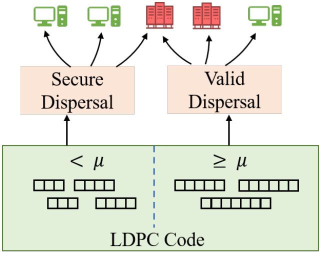


2. Valid Phase

Dispersal Protocol Design

Our dispersal strategy is a two step protocol

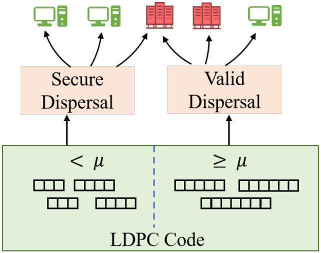
1. Secure Phase
   All small stopping sets (SSs) (size $< \mu$) are treated in an unified manner
   - Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur



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

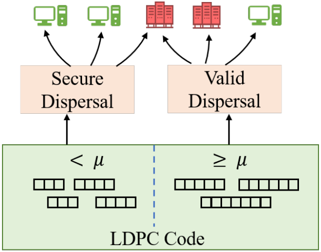


1. Secure Phase
   All small stopping sets (SSs) (size $< \mu$) are treated in an unified manner
   - Coded chunks are dispersed in a communication-efficient way such that the small SS failures cannot occur
2. Valid Phase
   A refinement of the dispersal protocol used in [Sheng '20] for larger SSs (size $\geq \mu$)
   - 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 stopping sets (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

- $Neigh(S) :=$ set of oracle nodes having at least one coded chunk of stopping set S

Secure Phase

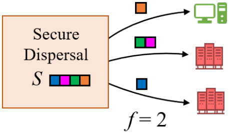
- $Neigh(S)$:= set of oracle nodes having at least one coded chunk of stopping set S
- f := maximum number of malicious oracle nodes

Secure Phase

- $Neigh(S) :=$ set 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 $|Neigh(S)| \geq f + 1$

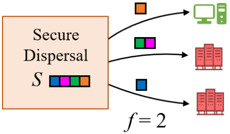
Secure Phase

- $Neigh(S)$:= set 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 $|Neigh(S)| \geq f + 1$



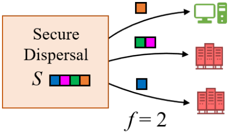
Secure Phase

- $Neigh(S) :=$ set 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 $|Neigh(S)| \geq f + 1$



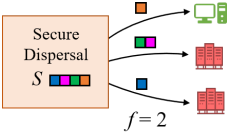
- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S

Secure Phase



- $Neigh(S)$:= set 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 $|Neigh(S)| \geq f + 1$
- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur

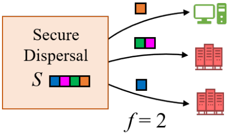
Secure Phase



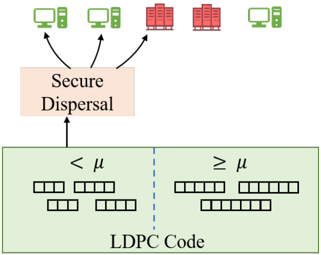
- $Neigh(S)$:= set of oracle nodes having at least one coded chunk of stopping set S
- f := maximum number of malicious oracle nodes
- S is *securely dispersed* if $|Neigh(S)| \geq f + 1$
- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to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

Secure Phase

- $Neigh(S)$:= set 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 $|Neigh(S)| \geq f + 1$



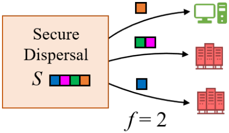
- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur



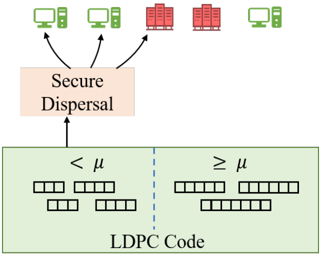
- $\mathcal{S}$ = All SSs of size $< \mu$
  Secure phase: all SSs in $\mathcal{S}$ are *securely dispersed*

Secure Phase

- $Neigh(S) :=$ set 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 $|Neigh(S)| \geq f + 1$



- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur

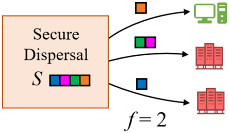


- $\mathcal{S} =$ All SSs of size $< \mu$
  Secure phase: all SSs in $\mathcal{S}$ are *securely dispersed*

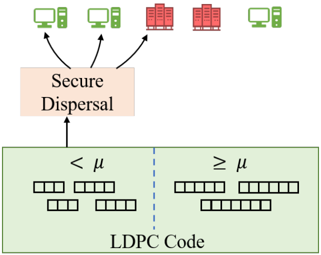
$< \mu$ size SSs cannot cause block unavailability

Secure Phase

- $Neigh(S)$:= set 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 $|Neigh(S)| \geq f + 1$



- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur



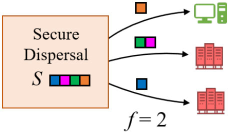
- $\mathcal{S}$ = All SSs of size $< \mu$
  Secure phase: all SSs in $\mathcal{S}$ are *securely dispersed*

$< \mu$ size SSs cannot cause block unavailability

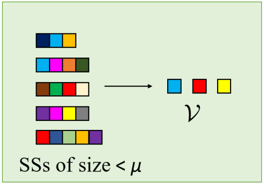
- $\mathcal{V}$: set of VNs that *cover* all SSs in $\mathcal{S}$

Secure Phase

- $Neigh(S) :=$ set 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 $|Neigh(S)| \geq f + 1$



- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur

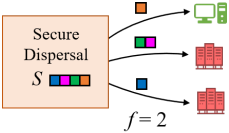


- $\mathcal{S} =$ All SSs of size $< \mu$
  Secure phase: all SSs in $\mathcal{S}$ are *securely dispersed*

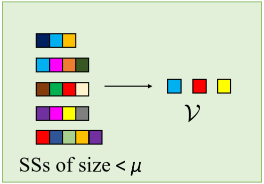
$< \mu$ size SSs cannot cause block unavailability

- $\mathcal{V}$: set of VNs that *cover* all SSs in $\mathcal{S}$

Secure Phase



- $Neigh(S)$:= set 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 $|Neigh(S)| \geq f + 1$
- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur



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

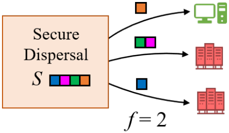
$< \mu$ size SSs cannot cause block unavailability

- $\mathcal{V}$: set of VNs that *cover* all SSs in $\mathcal{S}$

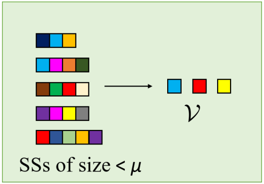
- Each VN in $\mathcal{V}$ is dispersed to $f + 1$ nodes

Secure Phase

- $Neigh(S) :=$ set 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 $|Neigh(S)| \geq f + 1$



- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur



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

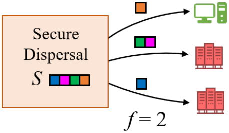
$< \mu$ size SSs cannot cause block unavailability

- $\mathcal{V}$: set of VNs that *cover* all SSs in $\mathcal{S}$

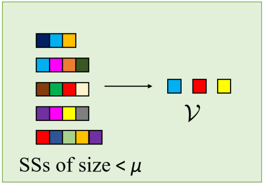
- Each VN in $\mathcal{V}$ is dispersed to $f + 1$ nodes
$\rightarrow$ ensures all SSs in $\mathcal{S}$ are securely dispersed

Secure Phase

- $Neigh(S) :=$ set 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 $|Neigh(S)| \geq f + 1$



- If a stopping set S is *securely dispersed*, at least one honest node will have a coded chunk corresponding to S
  $\rightarrow$ Failure of stopping set S cannot occur



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

$\boxed{< \mu \text{ size SSs cannot cause block unavailability}}$

- $\mathcal{V}$: set of VNs that *cover* all SSs in $\mathcal{S}$
$\rightarrow$ found greedily: $\textit{Greedy-Set}(\mathcal{S})$

- Each VN in $\mathcal{V}$ is dispersed to $f + 1$ nodes
$\rightarrow$ ensures all SSs in $\mathcal{S}$ are securely dispersed

Valid Phase

Consider the following dispersal protocol

Valid Phase

Consider the following dispersal protocol

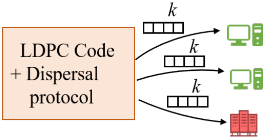
μ-SS-Valid dispersal
Every γ fraction of oracle nodes receives $\geq M - \mu + 1$ coded chunks

Valid Phase

Consider the following dispersal protocol

μ-SS-Valid dispersal
Every γ 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

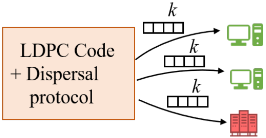
μ-SS-Valid dispersal
Every γ fraction of oracle nodes receives $\geq M - \mu + 1$ coded chunks

- 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

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

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



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

Valid Phase

Consider the following dispersal protocol

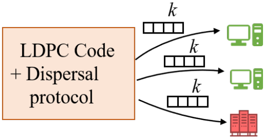
μ-SS-Valid dispersal
Every γ fraction of oracle nodes receives $\geq M - \mu + 1$ coded chunks

- 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

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

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



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

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

Valid Phase

Consider the following dispersal protocol

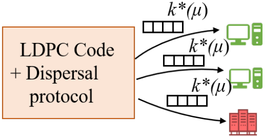
μ-SS-Valid dispersal
Every γ fraction of oracle nodes receives $\geq M - \mu + 1$ coded chunks

- 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

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

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



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

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

Valid Phase

Consider the following dispersal protocol

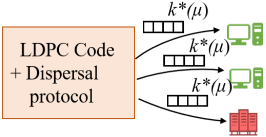
μ-SS-Valid dispersal
Every γ fraction of oracle nodes receives $\geq M - \mu + 1$ coded chunks

- 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

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

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



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

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

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

Overall Dispersal Strategy and Code Design

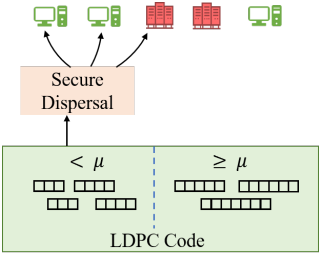
k^*-secure dispersal protocol

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

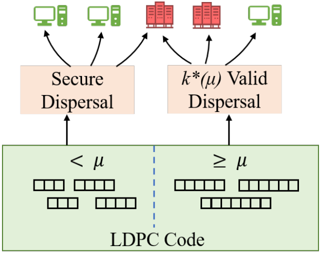
k^*-secure dispersal protocol

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

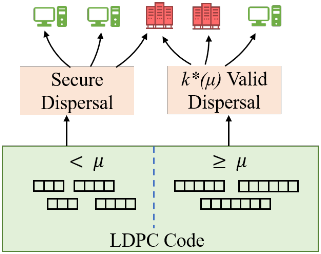
1. Secure Phase

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

   $< \mu$ size SSs cannot cause block unavailability

2. Valid Phase

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



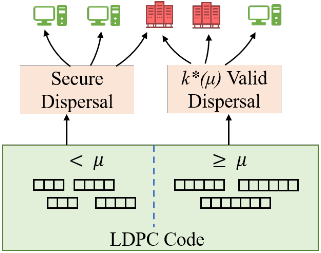
Overall Dispersal Strategy and Code Design

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

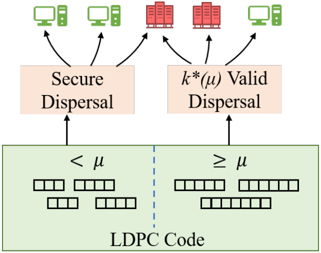
$\boxed{< \mu \text{ size SSs cannot cause block unavailability}}$

- Recall: Each VN in $Greedy\text{-}Set(\mathcal{S})$ 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

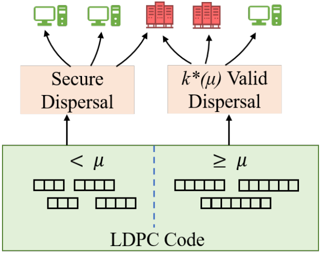
$< \mu$ size SSs cannot cause block unavailability

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

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

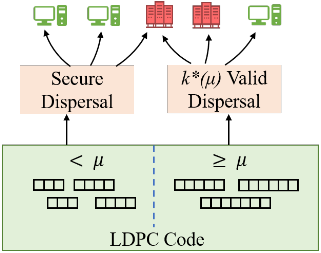
   $< \mu$ size SSs cannot cause block unavailability

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

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 $|\mathit{Greedy}\text{-}\mathit{Set}(\mathcal{S})|$

Overall Dispersal Strategy and Code Design

k^*-secure dispersal protocol

1. Secure Phase

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

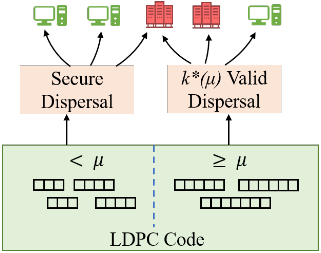
$< \mu$ size SSs cannot cause block unavailability

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

2. Valid Phase

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

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



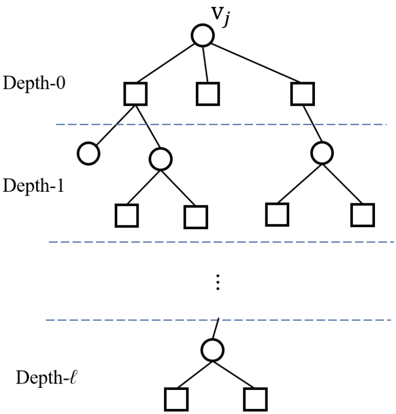
Code Design Strategy:
Design LDPC codes that have low $|Greedy\text{-}Set(\mathcal{S})|$

-Modify the PEG algorithm

PEG Algorithm

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

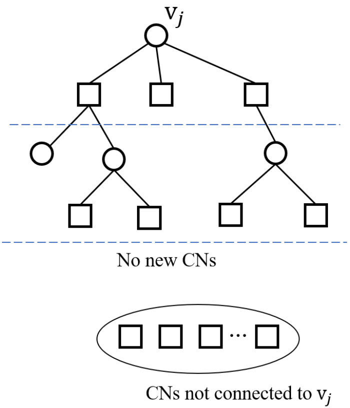
PEG Algorithm



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

For each VN v_j
  Expand Tanner Graph in a BFS fashion

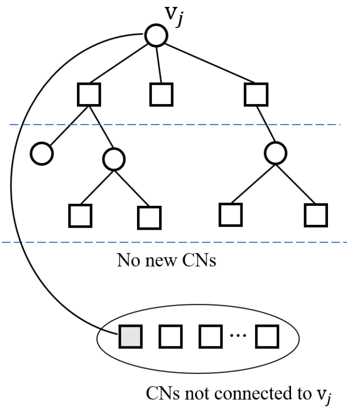
PEG Algorithm



- 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

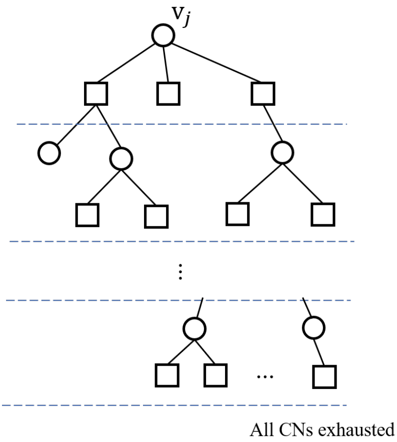
PEG Algorithm



- 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

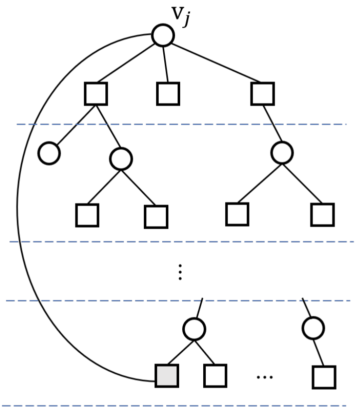


- Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

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

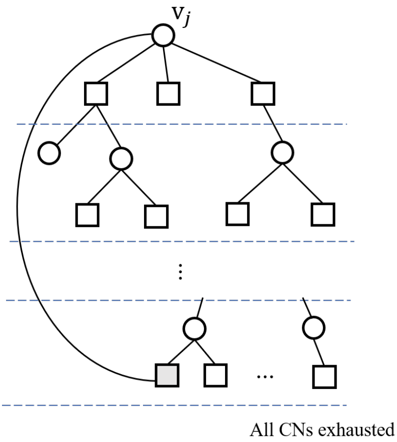


- Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

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



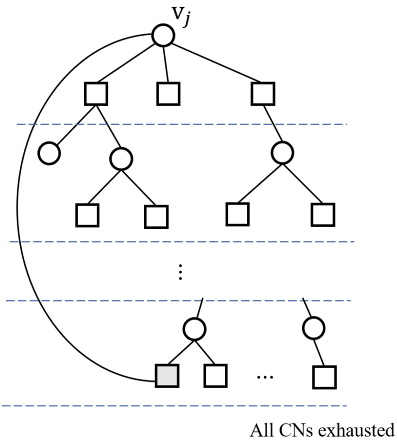
- Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

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

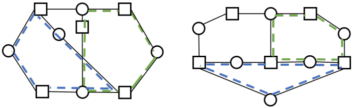
  Else
  - Find CNs most distant to v_j
  - Select one with minimum degree

    New cycles created

We modify the CN selection criteria in green to result in a low $|Greedy\text{-}Set(\mathcal{S})|$

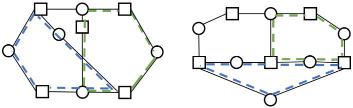
Dispersal-Efficient (DE)-PEG Algorithm

- SSs are made up of cycles [Tian '03]

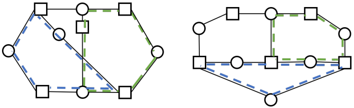


Dispersal-Efficient (DE)-PEG Algorithm

- SSs are made up of cycles [Tian '03]
- Want to design LDPC codes with low $|Greedy\text{-}Set(\mathcal{S})|$, $\mathcal{S} =$ all SSs of size $< \mu$



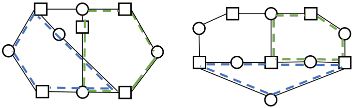
Dispersal-Efficient (DE)-PEG Algorithm



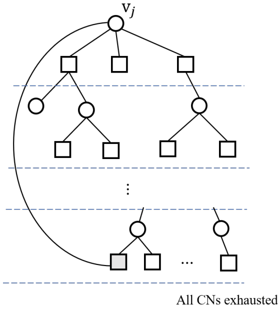
- SSs are made up of cycles [Tian '03]
- Want to design LDPC codes with low $|Greedy\text{-}Set(\mathcal{S})|$, $\mathcal{S}$ = all SSs of size $< \mu$
- Design LDPC codes to reduce $|Greedy\text{-}Set(\mathcal{L})|$, $\mathcal{L}$ = List of cycles of length $\leq g$

Dispersal-Efficient (DE)-PEG Algorithm

- SSs are made up of cycles [Tian '03]
- Want to design LDPC codes with low $|\mathit{Greedy\text{-}Set}(\mathcal{S})|$, $\mathcal{S}$ = all SSs of size $< \mu$



- Design LDPC codes to reduce $|\mathit{Greedy\text{-}Set}(\mathcal{L})|$, $\mathcal{L}$ = List of cycles of length $\leq g$



DE-PEG Algorithm

For each VN v_j

Expand Tanner Graph in a BFS fashion

If $\exists$ CNs not connected to v_j

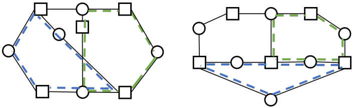
- Select a CN with min degree not connected to v_j

Else *(new cycles created)*

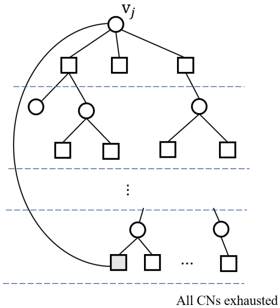
- Find CNs most distant to v_j
- Select CNs with minimum degree

Dispersal-Efficient (DE)-PEG Algorithm

- SSs are made up of cycles [Tian '03]
- Want to design LDPC codes with low $|Greedy\text{-}Set(\mathcal{S})|$, $\mathcal{S}$ = all SSs of size $< \mu$



- Design LDPC codes to reduce $|Greedy\text{-}Set(\mathcal{L})|$, $\mathcal{L}$ = List of cycles of length $\leq g$

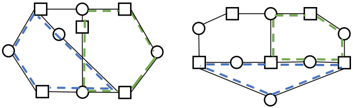


DE-PEG Algorithm
For each VN v_j
Expand Tanner Graph in a BFS fashion
If $\exists$ CNs not connected to v_j
- Select a CN with min degree not connected to v_j

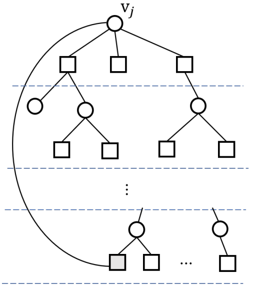
Else *(new cycles created)*
- Find CNs most distant to v_j
- Select CNs with minimum degree
- Select one with minimum $|Greedy\text{-}Set(\mathcal{L})|$

Dispersal-Efficient (DE)-PEG Algorithm

- SSs are made up of cycles [Tian '03]
- Want to design LDPC codes with low $|Greedy\text{-}Set(\mathcal{S})|$, $\mathcal{S} =$ all SSs of size $< \mu$



- Design LDPC codes to reduce $|Greedy\text{-}Set(\mathcal{L})|$, $\mathcal{L} =$ List of cycles of length $\leq g$



DE-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

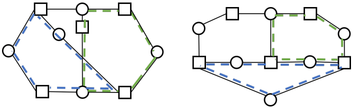
Else *(new cycles created)*

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

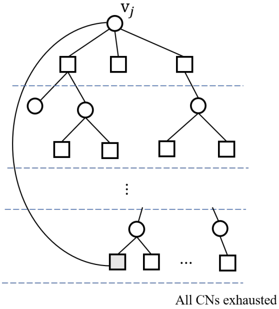
Issue: Using $\mathcal{L}$ that contains all cycles of length $\leq g$ does not reduce $|Greedy\text{-}Set(\mathcal{S})|$

Dispersal-Efficient (DE)-PEG Algorithm

- SSs are made up of cycles [Tian '03]
- Want to design LDPC codes with low $|Greedy\text{-}Set(\mathcal{S})|$, $\mathcal{S}$ = all SSs of size $< \mu$



- Design LDPC codes to reduce $|Greedy\text{-}Set(\mathcal{L})|$, $\mathcal{L}$ = List of cycles of length $\leq g$



DE-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 $|Greedy\text{-}Set(\mathcal{L})|$

Solution:
Make $\mathcal{L}$ contain only low Extrinsic Message Degree (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 = $\frac{1}{2}$, $\gamma = 1 - 2\beta$. All communication costs are in GB.

Simulation Results: Communication Cost Reduction

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

- $|\mathcal{V}| = |Greedy\text{-}Set(\mathcal{S})|$ for $M = 256$, $\mathcal{S} =$ all SS of size $< \mu$

μ	$\lvert\mathcal{V}\rvert$	
	PEG	DE-PEG
17	0	0
18	1	0
19	3	1
20	7	4
21	14	13

Simulation Results: Communication Cost Reduction

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

- $|\mathcal{V}| = |Greedy\text{-}Set(\mathcal{S})|$ for $M = 256$, $\mathcal{S} =$ all SS of size $< \mu$

μ	$\lvert\mathcal{V}\rvert$	
	PEG	DE-PEG
17	0	0
18	1	0
19	3	1
20	7	4
21	14	13

- 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 = $\frac{1}{2}$, $\gamma = 1 - 2\beta$. All communication costs are in GB.

- $|\mathcal{V}| = |Greedy\text{-}Set(\mathcal{S})|$ for $M = 256$, $\mathcal{S}$ = all SS of size $< \mu$
- C^s: communication cost of secure phase of dispersal

Secure Phase

μ	$\lvert\mathcal{V}\rvert$		C^s	
	PEG	DE-PEG	PEG	DE-PEG
17	0	0	0	0
18	1	0	0.037	0
19	3	1	0.112	0.037
20	7	4	0.262	0.149
21	14	13	0.524	0.486

- 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 = $\frac{1}{2}$, $\gamma = 1 - 2\beta$. All communication costs are in GB.

- $|\mathcal{V}| = |Greedy\text{-}Set(\mathcal{S})|$ for $M = 256$, $\mathcal{S}$ = all SS of size $< \mu$
- C^s: communication cost of secure phase of dispersal

Secure Phase

μ	$\lvert\mathcal{V}\rvert$ PEG	$\lvert\mathcal{V}\rvert$ DE-PEG	C^s PEG	C^s DE-PEG
17	0	0	0	0
18	1	0	0.037	0
19	3	1	0.112	0.037
20	7	4	0.262	0.149
21	14	13	0.524	0.486

- DE-PEG always results in lower $|\mathcal{V}|$ compared to PEG
- As μ is increased, C^s increases. C^s for DE-PEG $<$ C^s for 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 = $\frac{1}{2}$, $\gamma = 1 - 2\beta$. All communication costs are in GB.

- $|\mathcal{V}| = |Greedy\text{-}Set(\mathcal{S})|$ for $M = 256$, $\mathcal{S}$ = all SS of size $< \mu$
- C^s: communication cost of secure phase of dispersal
- C^v: communication cost of valid phase of dispersal (each node gets $k^*(\mu)$ chunks)

Secure Phase | Valid Phase

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

- DE-PEG always results in lower $|\mathcal{V}|$ compared to PEG
- As μ is increased, C^s increases. C^s for DE-PEG $<$ C^s for 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 $= \frac{1}{2}$, $\gamma = 1 - 2\beta$. All communication costs are in GB.

- $|\mathcal{V}| = |\mathit{Greedy\text{-}Set}(\mathcal{S})|$ for $M = 256$, $\mathcal{S} =$ all SS of size $< \mu$
- C^s: communication cost of secure phase of dispersal
- C^v: communication cost of valid phase of dispersal (each node gets $k^*(\mu)$ chunks)

Secure Phase Valid Phase

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

- DE-PEG always results in lower $|\mathcal{V}|$ compared to PEG
- As μ is increased, C^s increases. C^s for DE-PEG $<$ C^s for PEG, 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 = $\frac{1}{2}$, $\gamma = 1 - 2\beta$. All communication costs are in GB.

- $|\mathcal{V}| = |\mathit{Greedy\text{-}Set}(\mathcal{S})|$ for $M = 256$, $\mathcal{S}$ = all SS of size $< \mu$
- C^s: communication cost of secure phase of dispersal
- C^v: communication cost of valid phase of dispersal (each node gets $k^*(\mu)$ chunks)
- C^T: total communication cost = $\mathrm{C}^v + \mathrm{C}^s + \Delta$ (small additional overhead)

Secure Phase | Valid Phase

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

- DE-PEG always results in lower $|\mathcal{V}|$ compared to PEG
- As μ is increased, C^s increases. C^s for DE-PEG $<$ C^s for PEG, 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 = $\frac{1}{2}$, $\gamma = 1 - 2\beta$. All communication costs are in GB.

- $|\mathcal{V}| = |Greedy\text{-}Set(\mathcal{S})|$ for $M = 256$, $\mathcal{S}$ = all SS of size $< \mu$
- C^s: communication cost of secure phase of dispersal
- C^v: communication cost of valid phase of dispersal (each node gets $k^*(\mu)$ chunks)
- C^T: total communication cost = $\mathrm{C}^v + \mathrm{C}^s + \Delta$ (small additional overhead)

Secure Phase — Valid Phase

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

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

Simulation Results: Communication Cost Reduction

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

Simulation Results: Communication Cost Reduction

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

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

Simulation Results: Communication Cost Reduction

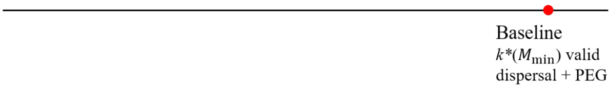
μ	$\lvert\mathcal{V}\rvert$		C^s		C^v	C^T	
	PEG	DE-PEG	PEG	DE-PEG		PEG	DE-PEG
17	0	0	0	0	5.116	5.125	5.125
18	1	0	0.037	0	4.887	4.933	4.896
19	3	1	0.112	0.037	4.658	4.779	4.704
20	7	4	0.262	0.149	4.428	4.700	4.587
21	14	13	0.524	0.486	4.276	4.809	4.771

- $M_{\min}$ 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

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

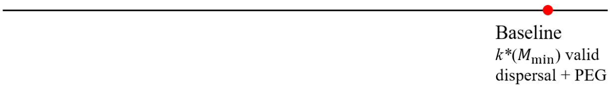
- $M_{\min}$ for PEG LDPC code is 17.
- $\mu = 17$ is considered as the baseline with $k^*(M_{\min})$ valid dispersal protocol (no secure phase)



Simulation Results: Communication Cost Reduction

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

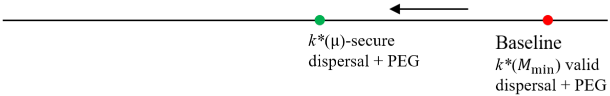
- $M_{\min}$ for PEG LDPC code is 17.
- $\mu = 17$ is considered as the baseline with $k^*(M_{\min})$ valid dispersal protocol (no secure phase)
- Using k^*-secure dispersal protocol with $\mu = 20$ reduces C^T from baseline:



Simulation Results: Communication Cost Reduction

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

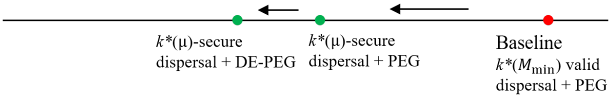
- $M_{\min}$ for PEG LDPC code is 17.
- $\mu = 17$ is considered as the baseline with $k^*(M_{\min})$ valid dispersal protocol (no secure phase)
- Using k^*-secure dispersal protocol with $\mu = 20$ reduces C^T from baseline:
  Reduction for PEG: 0.425GB



Simulation Results: Communication Cost Reduction

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

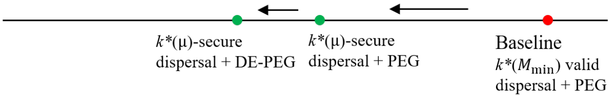
- $M_{\min}$ for PEG LDPC code is 17.
- $\mu = 17$ is considered as the baseline with $k^*(M_{\min})$ valid dispersal protocol (no secure phase)
- Using k^*-secure dispersal protocol with $\mu = 20$ reduces C^T from baseline:
  Reduction for PEG: 0.425GB
  Reduction for DE-PEG: 0.528GB



Simulation Results: Communication Cost Reduction

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

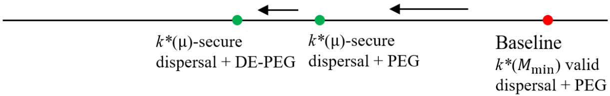
- $M_{\min}$ for PEG LDPC code is 17.
- $\mu = 17$ is considered as the baseline with $k^*(M_{\min})$ valid dispersal protocol (no secure phase)
- Using k^*-secure dispersal protocol with $\mu = 20$ reduces C^T from baseline:
  Reduction for PEG: 0.425GB
  Reduction for DE-PEG: 0.528GB
- Lower bound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



Simulation Results: Communication Cost Reduction

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

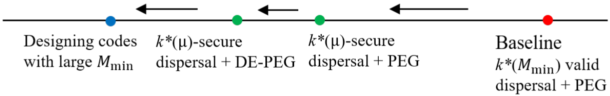
- $M_{\min}$ for PEG LDPC code is 17.
- $\mu = 17$ is considered as the baseline with $k^*(M_{\min})$ valid dispersal protocol (no secure phase)
- Using k^*-secure dispersal protocol with $\mu = 20$ reduces C^T from baseline:
  Reduction for PEG: 0.425GB
  Reduction for DE-PEG: 0.528GB
- Lower bound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



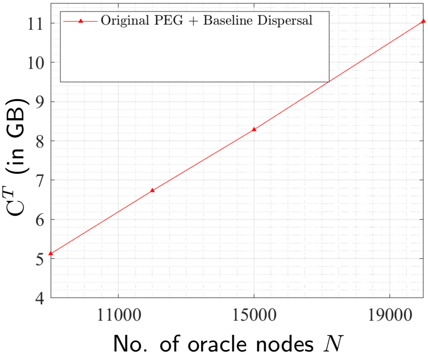
Simulation Results: Communication Cost Reduction

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

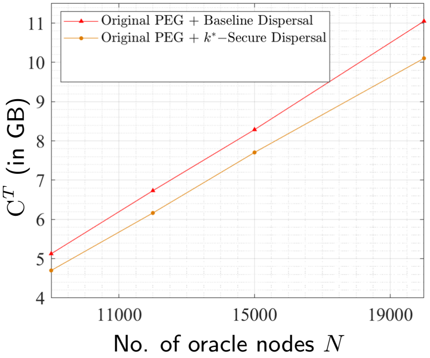
- $M_{\min}$ for PEG LDPC code is 17.
- $\mu = 17$ is considered as the baseline with $k^*(M_{\min})$ valid dispersal protocol (no secure phase)
- Using k^*-secure dispersal protocol with $\mu = 20$ reduces C^T from baseline:
  Reduction for PEG: 0.425GB
  Reduction for DE-PEG: 0.528GB
- Lower bound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



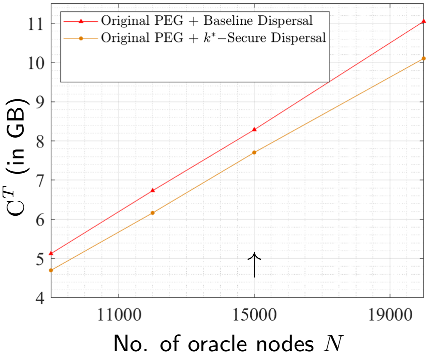
Simulation Results



Simulation Results



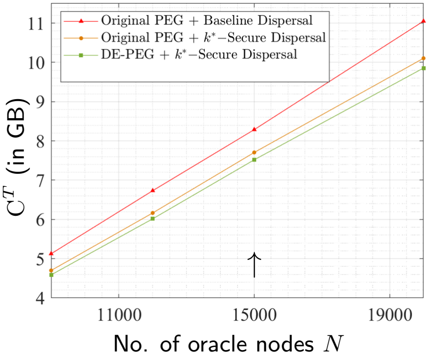
Simulation Results



At $N = 15000$

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

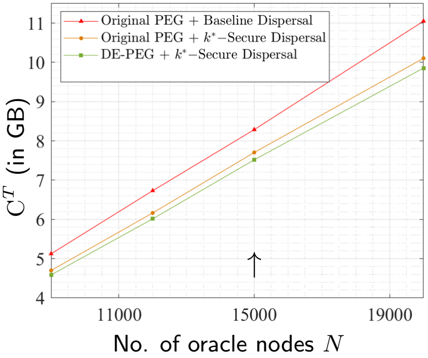
Simulation Results



At $N = 15000$

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

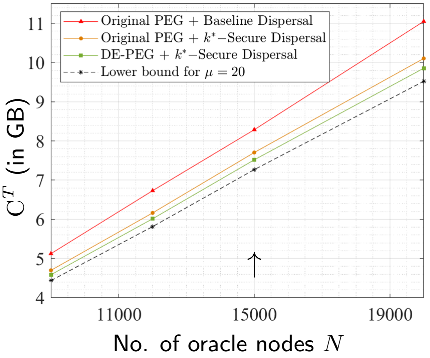
Simulation Results



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 $\mu = 20$

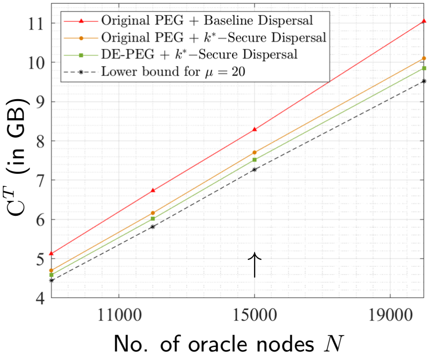
Simulation Results



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 $\mu = 20$

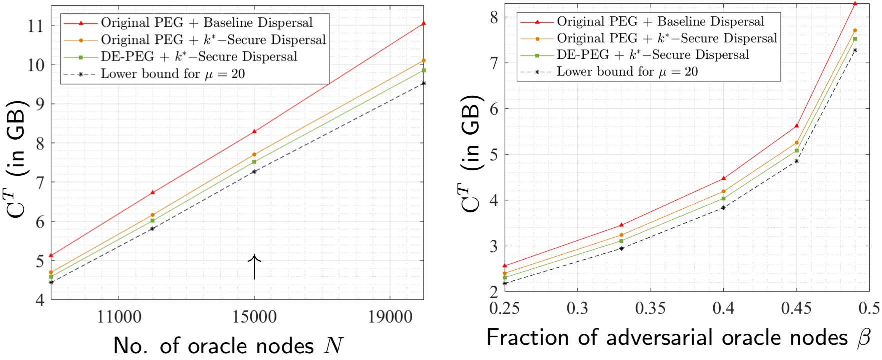
Simulation Results



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 $\mu = 20$

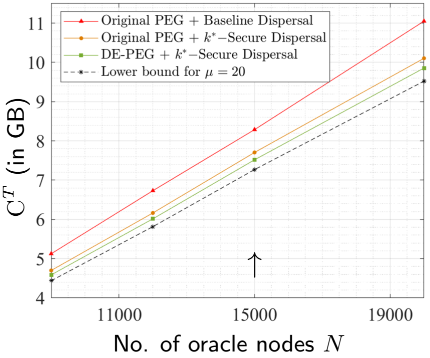
Simulation Results

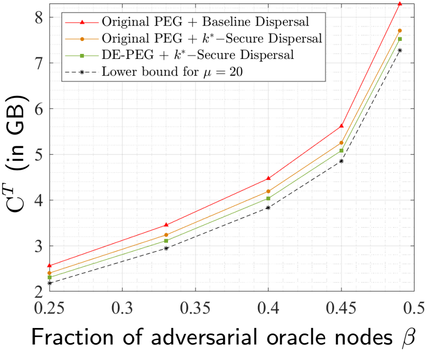


At $N = 15000$

- 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



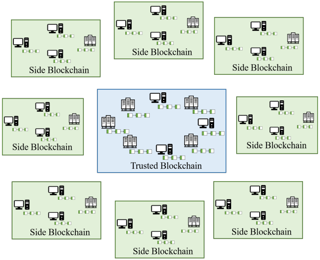


At $N = 15000$

- 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 β

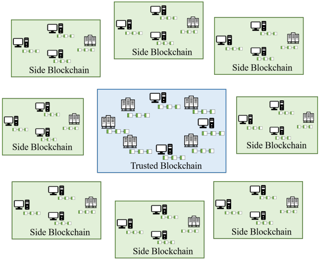
Conclusion and Ongoing work

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



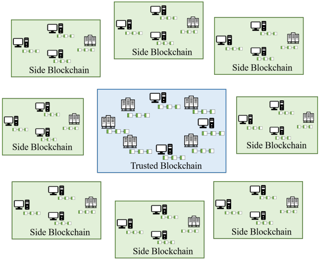
Conclusion and Ongoing work

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



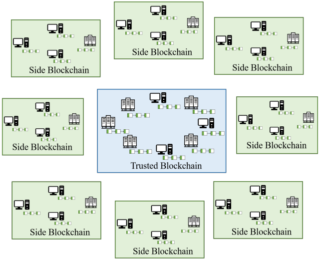
Conclusion and Ongoing work

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



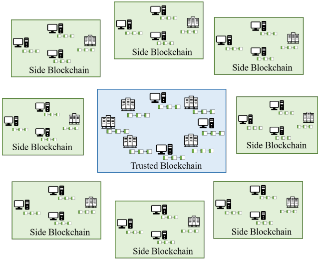
Conclusion and Ongoing work

- Conclusion
  - Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
    ↳ Adversarial erasures with dispersal protocol
  - LDPC codes tailor made for these specific channels demonstrate better performance
    ↳ k^*-secure dispersal protocol
    ↳ DE-PEG algorithm



Conclusion and Ongoing work

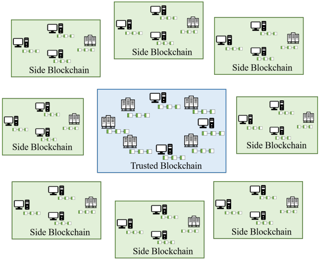
- Conclusion
  - Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
    ↳ Adversarial erasures with dispersal protocol
  - LDPC codes tailor made for these specific channels demonstrate better performance
    ↳ k^*-secure dispersal protocol
    ↳ DE-PEG algorithm



- Ongoing work

Conclusion and Ongoing work

- Conclusion
  - Off-the-shelf LDPC codes, e.g. those designed for AWGN or BSC channels, may not be optimal for:
    ↳ Adversarial erasures with dispersal protocol
  - LDPC codes tailor made for these specific channels demonstrate better performance
    ↳ k^*-secure dispersal protocol
    ↳ DE-PEG algorithm



- Ongoing work
  - Considering other code families such as Polar codes for this application.

References

- D. Mitra, L. Tauz, and L. Dolecek, *"Communication-Efficient LDPC Code Design for Data Availability Oracle in Side Blockchains,"* available at https://arxiv.org/abs/2105.06004)
- (Sheng '20) P. Sheng, et al., *"ACeD: Scalable Data Availability Oracle"* arXiv preprint arXiv:2011.00102, Oct. 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.
- (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.

References

- (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.