Traditional Designs
Observation

• Monolithic protocol sharing transaction data as part of the consensus
• Optimize overall message complexity of the consensus protocol
Current Designs
Typical leader-based protocols
Current Designs

Typical leader-based protocols
Data dissemination is the key

Reaching consensus on metadata is cheap
Narwhal

Dag-based mempool
Narwhal
The workers and the primary

Narwhal mempool

Worker 1
Worker 2
Worker n
Primary

Client transactions
Narwhal
The workers and the primary

Client transactions

Transactions

Worker 1

Worker 2

Worker n

Narwhal mempool

Primary
Narwhal
The workers and the primary

Client transactions

Transactions → Worker 1 → Batch

Transactions → Worker 2 → Batch

Transactions → Worker n → Batch

Narwhal mempool

Primary
Narwhal
The workers and the primary

- Client transactions
  - Transactions
    - Worker 1
    - Worker 2
    - Worker n
    - Primary
  - Batch
  - Digest
Narwhal
The workers and the primary

Client transactions

Transactions

Worker 1

Worker 2

Worker n

Narwhal mempool

Transactions

Transactions

Transactions

Digest

Digest

Digest

Batch

Batch

Batch

'mempool protocol'
Narwhal
The primary machine

block header

G1

G2

G3
Narwhal
The primary machine

G1
G2
G3

block header

H V

H V

H V

H V

Narwhal
The primary machine
Narwhal
The primary machine

Round 1
Narwhal
The primary machine

Round 1
Byzantine 'Reliable' Broadcast
Narwhal
The primary machine
Narwhal
Data Dissemination & Proof of Availability

• The workers ship batch of transactions
• Many workers to scale out and use resources concurrently
• The primary constantly broadcasts the batch digests
• Headers at round $r$ contains references to $2f+1$ certificates of round $r-1$
• Build a structured DAG of certificates
Tusk

Zero-message asynchronous consensus
Tusk
The random coin elects the leader of r-2
Tusk
The leader needs $f+1$ links from round $r-1$

Not enough support!
(Nothing is committed at this stage)
Tusk

Nothing is committed and we keep build the DAG

r1  r2  r3  r4  r5

L1
Tusk
Elect the leader of r3
Tusk
Leader L2 has enough support
Tusk
Leader L2 has links to leader L1

First commit L1
Then commit L2
Commit all the sub-DAG of the leader
Bullshark

Zero-message partially-synchronous consensus

* without asynchronous fallback
Bullshark
Just interpret the DAG
Bullshark
Deterministic leader every 2 rounds
Bullshark
The leader needs $f+1$ links from round $r$
Bullshark
The leader needs f+1 links from round r
Bullshark
The leader needs f+1 links from round r

Not enough support!
(Nothing is committed at this stage)
Bullshark
Elect the leader of r4
Bullshark
Leader L2 has enough support
Bullshark
Leader L2 has links to leader L1

First commit L1

Then commit L2
Bullshark
Commit all the sub-DAG of the leader

L1
r1
r2
r3
r4
r5
L2
Bullshark
Commit all the sub-DAG of the leader
Evaluation

How to properly benchmark consensus protocols
Evaluation

Typical mistakes

蝮 Forgo persistent storage
蝮 Do not sanitise messages
蝮 Local/LAN benchmark + ping
蝮 Many nodes on same machine
蝮 Change parameters across runs
蝮 Set transaction size to zero
蝮 Preconfigure nodes with txs
蝮 Send a single burst of transactions
蝮 Benchmark for a few seconds
蝮 Start timer in the batch maker
蝮 Evaluate latency w/ only the first tx
蝮 Separate latency and throughput
蝮 Only benchmark happy path
Evaluation
Set the benchmark parameters

Faults: 0 node(s)
Committee size: 10 node(s)
Transaction size: 512 B
Evaluation
Set the benchmark parameters

Faults: 0 node(s)
Committee size: 10 node(s)
Transaction size: 512 B

Header size: 1,000 B
Max header delay: 200 ms
GC depth: 50 round(s)
Sync retry delay: 5,000 ms
Sync retry nodes: 3 node(s)
batch size: 500,000 B
Max batch delay: 200 ms
Evaluation
Typical mistakes

ulant storage
Do not sanitise messages
Local/LAN benchmark + ping
Many nodes on same machine
Change parameters across runs
Set transaction size to zero
Preconfigure nodes with txs

Send a single burst of transactions
Benchmark for a few seconds
Start timer in the batch maker
Evaluate latency w/ only the first tx
Separate latency and throughput
Only benchmark happy path
Evaluation
Benchmark clients

Fixed input rate
For a long time (minutes)
Evaluation
Typical mistakes

灯笼 Forgo persistent storage
灯笼 Do not sanitise messages
灯笼 Local/LAN benchmark + ping
灯笼 Many nodes on same machine
灯笼 Change parameters across runs
灯笼 Set transaction size to zero
灯笼 Preconfigure nodes with txs
灯笼 Send a single burst of transactions
灯笼 Benchmark for a few seconds
灯笼 Start timer in the batch maker
灯笼 Evaluate latency w/ only the first tx
灯笼 Separate latency and throughput
灯笼 Only benchmark happy path
Evaluation

Typical mistake

Narwhal mempool

Propose batch 5 (pointer)

Load txs from pre-populated store & commit

Narwhal mempool

Load txs from pre-populated store & commit

Narwhal mempool

Load txs from pre-populated store & commit

Narwhal mempool

Tusk

Tusk

Tusk

Tusk
Evaluation

Typical mistakes

 очков

- Forgo persistent storage
- Do not sanitise messages
- Local/LAN benchmark + ping
- Many nodes on same machine
- Change parameters across runs
- Set transaction size to zero
- Preconfigure nodes with txs

 очков

- Send a single burst of transactions
- Benchmark for a few seconds
- Start timer in the batch maker
- Evaluate latency w/ only the first tx
- Separate latency and throughput
- Only benchmark happy path
Evaluation
Typical mistake

send 50k txs (once)

Benchmark client → Narwhal mempool

Ordered transactions → output after 400 ms

 😫 TPS = 50k / 400ms = 125k tx/s 😫
Evaluation
Instrument the codebase

bench_start_time
sample_tx_id -> send_time
Evaluation
Instrument the codebase

batch_digest -> sample_tx_id
batch_digest -> batch_bytes

bench_start_time
sample_tx_id -> send_time
Evaluation
Instrument the codebase

- `batch_digest` -> `sample_tx_id`
- `batch_digest` -> `batch_bytes`
- `block_digest` -> `batch_digest`

Benchmark client

Batch Maker

Proposer

Tusk

Ordered transactions

- `bench_start_time`
- `sample_tx_id` -> `send_time`
Evaluation
Instrument the codebase

- batch_digest $\rightarrow$ sample_tx_id
- batch_digest $\rightarrow$ batch_bytes
- block_digest $\rightarrow$ batch_digest
- bench_start_time
- sample_tx_id $\rightarrow$ send_time
- Tusk
- Proposer
- Batch Maker
- Batch
- Narwhal mempool
- Ordered transactions
- block_digest $\rightarrow$ commit_time
Evaluation
Compute throughput

declare bench_start_time

sample_tx_id -> send_time

Benchmark client to Batch Maker to Proposer to Tusk

Narwhal mempool

batch_digest -> sample_tx_id
batch_digest -> batch_bytes

block_digest -> batch_digest
block_digest -> commit_time

total_time = last_commit_time - bench_start_time

BPS = total_bytes / total_time

TPS = BPS / transaction_size
samples = commit_time - send_time
latency = average(samples)
Evaluation

Typical mistakes

😊 Forgo persistent storage
😊 Do not sanitise messages
😊 Local/LAN benchmark + ping
😊 Many nodes on same machine
😊 Change parameters across runs
😊 Set transaction size to zero
😊 Preconfigure nodes with txs

😊 Send a single burst of transactions
😊 Benchmark for a few seconds
😊 Start timer in the batch maker
😊 Evaluate latency w/ only the first tx
😊 Separate latency and throughput
😊 Only benchmark happy path
Evaluation
Throughput latency graph
Evaluation
Throughput latency graph

Change only
input rate
Evaluation
Throughput latency graph
Evaluation
Throughput latency graph
Evaluation
Throughput latency graph

 Longer benchmarks
Evaluation
Throughput latency graph

Breaking point!
Evaluation

Typical mistakes

خدام Forgo persistent storage
خدام Do not sanitise messages
خدام Local/LAN benchmark + ping
خدام Many nodes on same machine
خدام Change parameters across runs
خدام Set transaction size to zero
خدام Preconfigure nodes with txs

خدام Send a single burst of transactions
خدام Benchmark for a few seconds
خدام Start timer in the batch maker
خدام Evaluate latency w/ only the first tx
خدام Separate latency and throughput
خدام Only benchmark happy path
Evaluation
Throughput latency graph
Evaluation
Throughput latency graph
Evaluation
Scalability
Evaluation
Scalability
Evaluation

Performance under faults
Evaluation
Typical mistakes

🤖 Forgo persistent storage
🤖 Do not sanitise messages
🤖 Local/LAN benchmark + ping
🤖 Many nodes on same machine
🤖 Change parameters across runs
🤖 Set transaction size to zero
🤖 Preconfigure nodes with txs

🤖 Send a single burst of transactions
🤖 Benchmark for a few seconds
🤖 Start timer in the batch maker
🤖 Evaluate latency w/ only the first tx
🤖 Separate latency and throughput
🤖 Only benchmark happy path
Evaluation
Still many caveats

- Perfect load balance
- Transaction deduplication
- Synthetic load
- No Byzantine adversary
- No network adversary
- Only AWS network
Conclusion

Narwhal & Tusk

- Separate consensus and data dissemination for high performance
- Scalable design, egalitarian resource utilisations

- **Code:** [https://github.com/asonnino/narwhal](https://github.com/asonnino/narwhal)
Acknowledgements

George Danezis
Lefteris Kokoris-Kogias
Alexander Spiegelman
Alberto Sonnino

Work done at Facebook Novi
Future Works
Come talk to us!

• Performance under DDoS attack?
• How to implement scalable execution?
alberto@mystenlabs.com

Alberto Sonnino