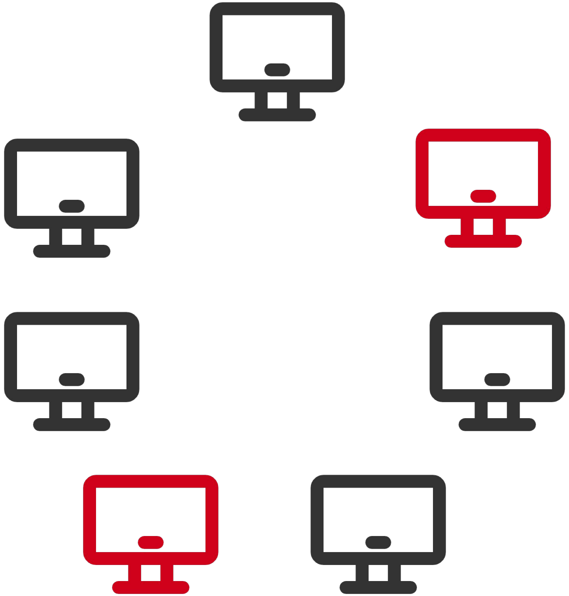
Dumbo: Faster Asynchronous BFT Protocols

Presenters: Yifeng He, Boqi Zhao, Zijin Cui, Jicheng Wang

BFT Protocols: The Atomic Broadcast Problem



Consistency

All honest nodes maintain identical (order and content) logs of delivered messages.

Liveness

If a message is broadcast by an honest node, it is eventually delivered by all honest nodes.

Timing Assumptions in Distributed Protocols

Total: all messages are delivered within a given time bound *d*
Partial: there is still a time bound *d*, but *d* is unknown
Asynchronous: all messages will be delivered, eventually

*NOTE: This chart is not exhaustive, and not all mentioned protocols can tolerant byzantine faults.

Why Asynchronous?

Tuning the timeout parameter

- d is as small as possible, then re-electing new leaders
- d is large, then the system recovers very slow

Responsiveness

Synchronous and partial synchronous protocols may hang under some DoS attacks with the timeout parameter, whereas asynchronous protocol will eventually proceed.

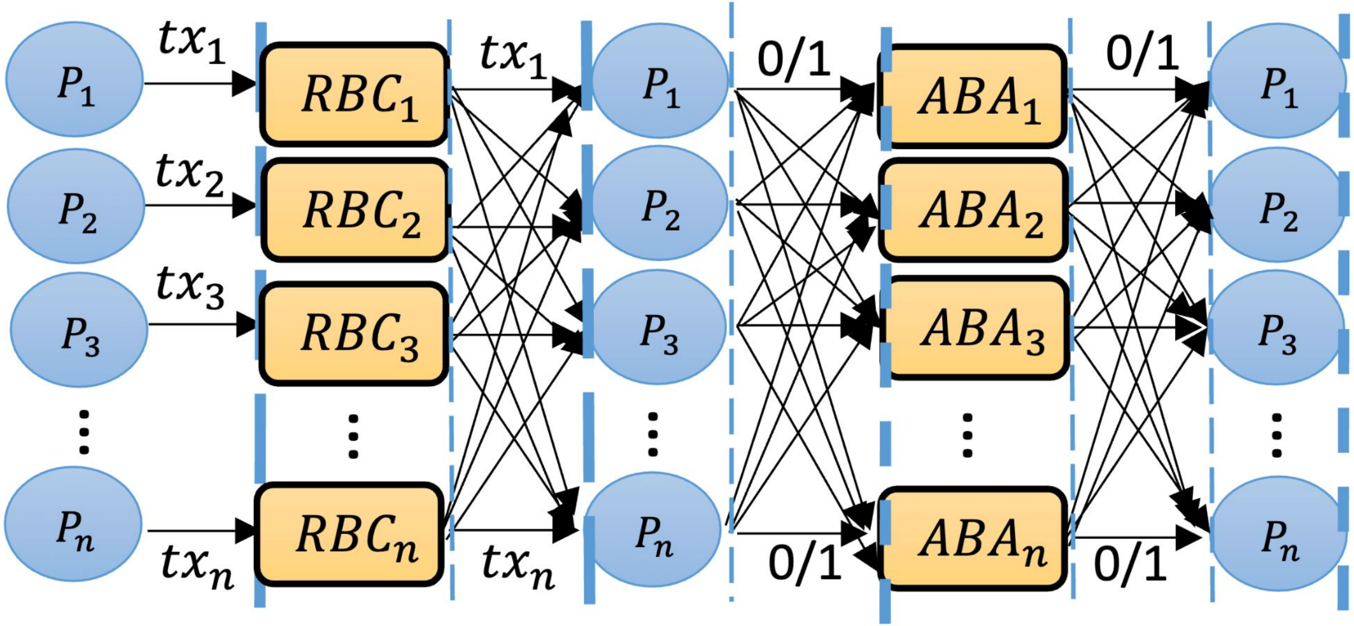
Asynchronous Common Subset (ACS) Protocol

Definition (Ben-Or et al., 1994): A common ACS protocol is built from two sub-protocols: reliable broadcast (RBC) and asynchronous binary agreement (ABA).

Each node invokes an RBC to broadcast its input value, and participates in n instances of the ABA protocol to agree on which subset of inputs to include.

ACS is good for batching in HoneyBadgerBFT.

ACS in HoneyBadgerBFT

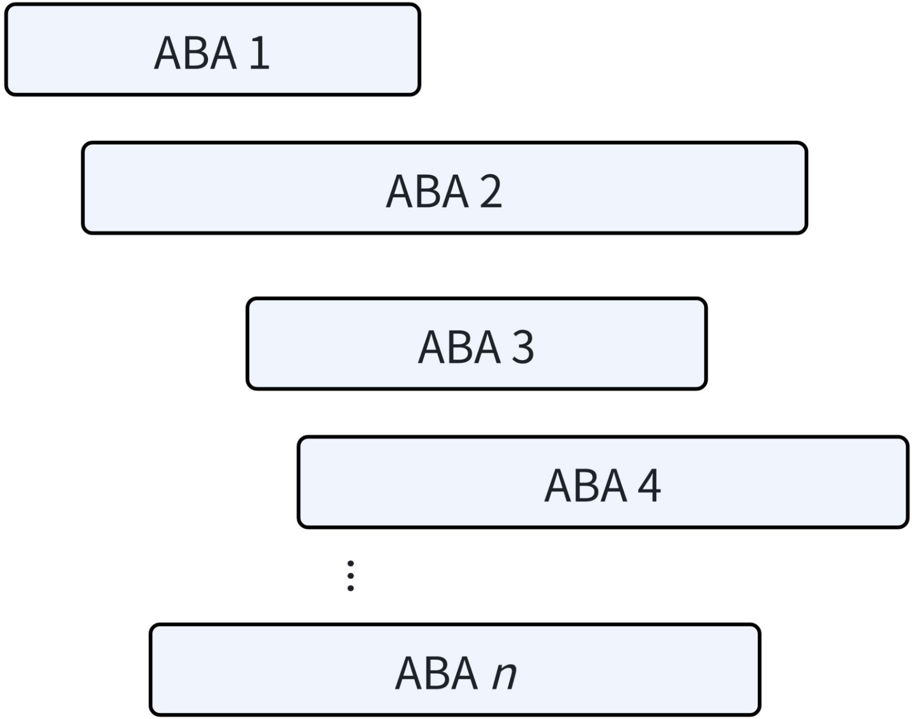


Every node participates in in a binary agreement to vote for each individual transaction (*n* in total). Everyone needs to participate in *n* transactions to do *n* votes.

ABA is the slower part in ACS

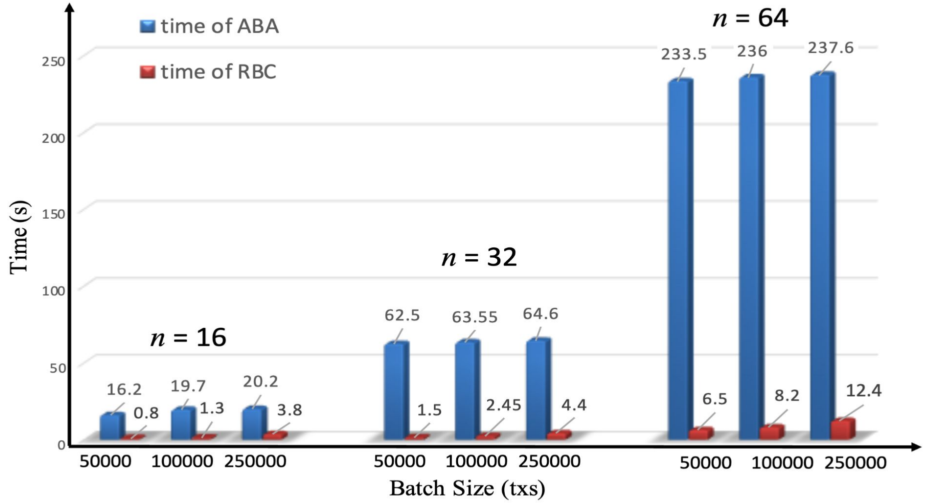
All protocol for asynchronous consensus are randomized.

Concurrent execution of a large number of randomized algorithms will hang *at the slowest instance*.



Validating Previous Observation on HoneyBadgerBFT

1. not all instances start at the same time, some of the instances may start later waiting for RBC.
2. normal node also has an efficiency degradation facing large scale concurrent execution (not enough CPU cores etc).



The slowest ABA instance determines the running time of the ACS of HoneyBadgerBFT.

Research Question:

How do the authors optimize the required number of instances of ABA in a new ACS protocol?

Dumbo1: reducing # instances of ABA to independent of n

1. HoneyBadgerBFT

    requires **n** ABA instances → High Communication Cost

    full data broadcast during RBC → Bandwidth Intensitivity

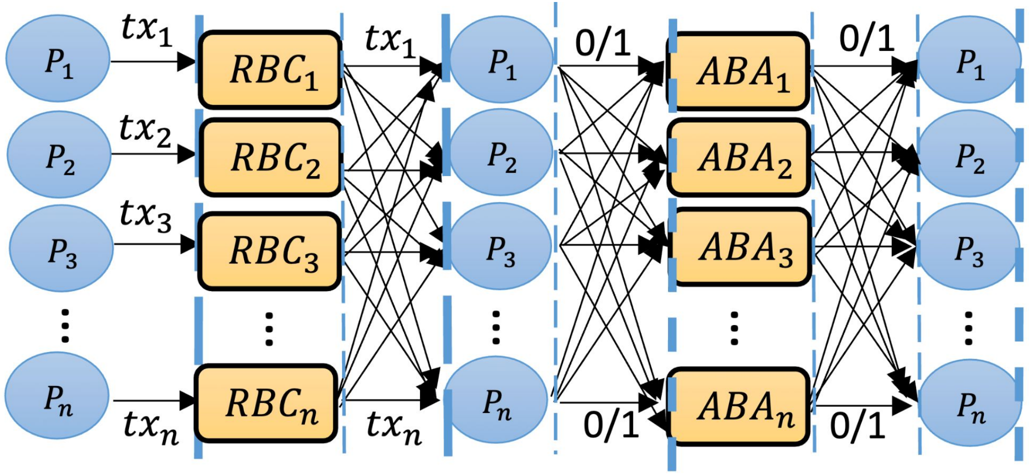
2. Dumbo1

    reduces ABA instances significantly (**k<<n**)

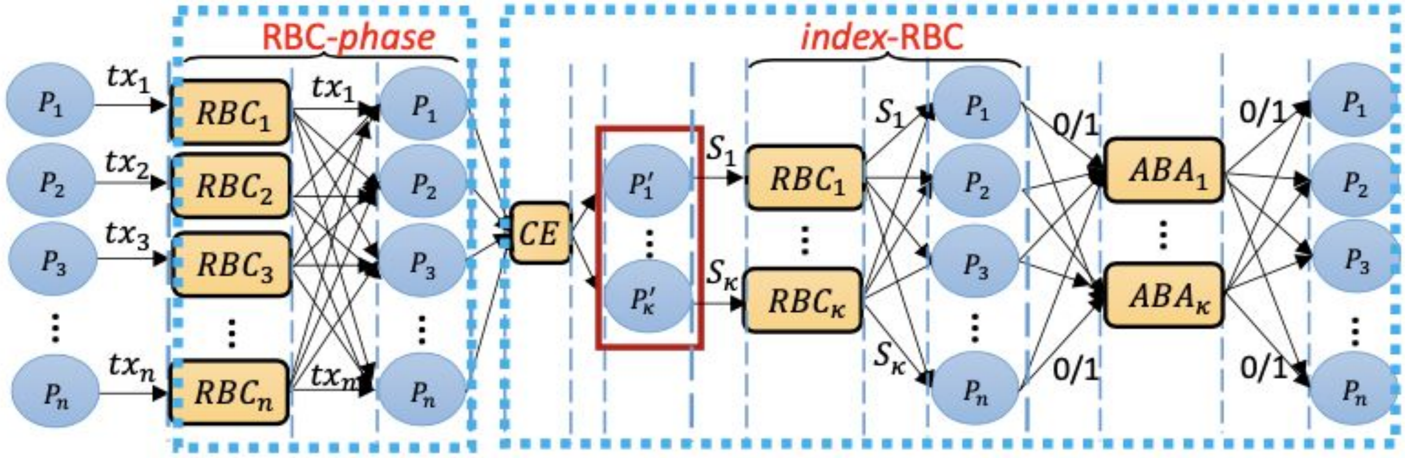
    uses **index broadcasts** to reduce bandwidth

    thus offers better scalability (without compromising on security or liveness)

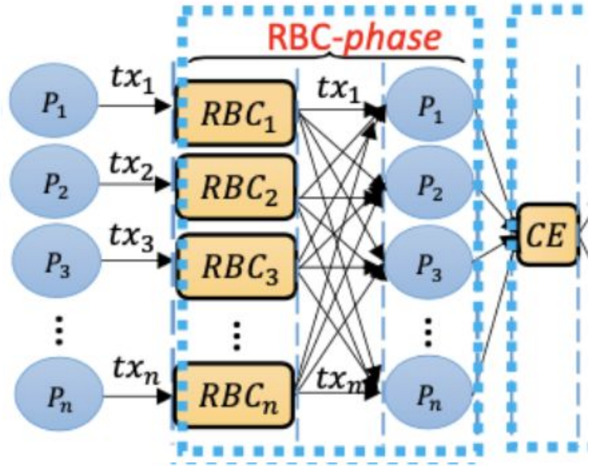
HoneyBadgerBFT



Dumbo1



Dumbo1: Execution Pipeline – RBC-phase and CE



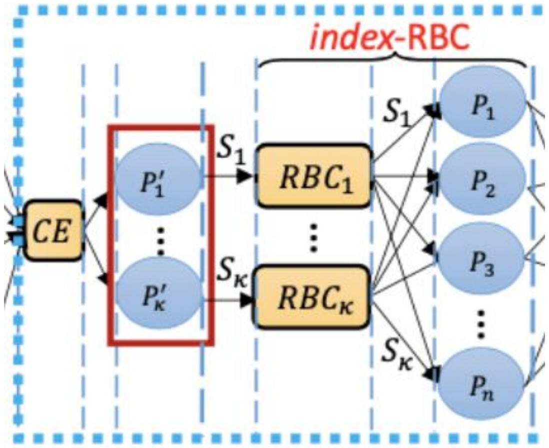
Premise: n = 3f + 1, where n is the number of all nodes, f is the number of Byzantine nodes.

1. Nodes broadcast their input values.
2. Select k nodes as leaders, where k ≤ f.
3. Committee members wait until they have received n−f valid input values.

P is the probability of CE containing no honest nodes.

$$p = \frac{\binom{f}{\kappa}}{\binom{n}{\kappa}} = \frac{f!(n-\kappa)!}{(f-\kappa)!n!} \leqslant \left(\frac{1}{3}\right)^{\kappa}$$

Dumbo1: Execution Pipeline – CE and Index-RBC

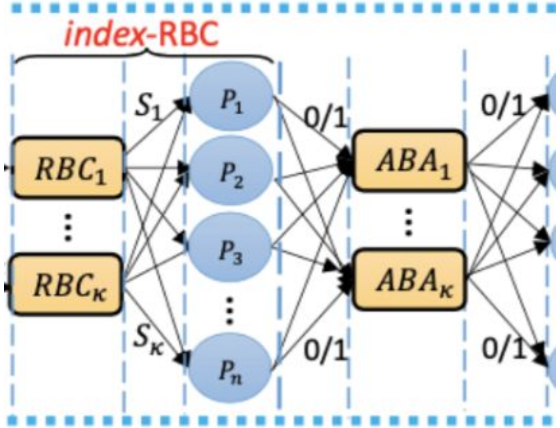


4. The committee members (P'_1 to P'_k) have received n-f Value message from distinct RBC instances.

5. Each of the committee members (P'_1 to P'_k) creates a set (S_1 to S_k), where the indices of nodes received are stored.

6. Each committee member broadcast its set to all nodes P_i.

Dumbo1: Execution Pipeline – ABA



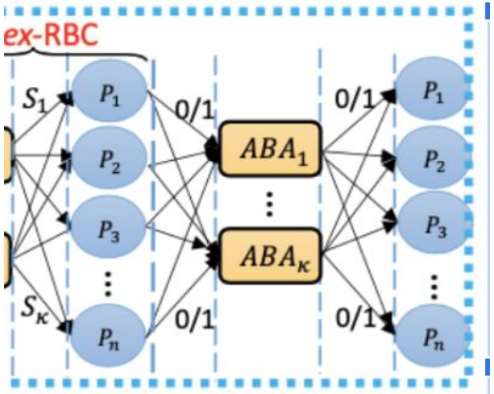
7. For each node P_i, it inputs **1** into ABA_i, if it has received all the values corresponding to S_i during data-RBC phase. Otherwise, it inputs **0.**

8. The ABA instance output **1** if all honest nodes agree to include S_i, otherwise to not include.

Early Termination Policy:

Once at least n-f ABA instances output 1, all others that have not completed are terminated by inputting 0.

Dumbo1: Execution Pipeline – Output



8. The ABA instance output **1** if all honest nodes agree to include S_i, otherwise to not include.

9. Each node collects the index sets S_i for which ABA_i outputs 1.

10. Nodes retrieve the actual inputs values corresponding to the indices in S_i by querying the data-RBC instances.

11. All instances from the previous step form the output value set.

Dumbo2: MVBA + Constant ABA Iteration✨

Dumbo2 Premise/Intuition🧐:

Dumbo1 reduced number of ABA iterations to *k*, but we can do better!

1) Prepare each peer with a vector of inputs from enough peer nodes
2) Find a way to identify and output one of them

   MVBA only needs on expectation three consecutive ABA instances.

What if indexed to a bad replica🤔?

Provable Reliable Broadcast (PRBC) to preserve integrity of the broadcast 😇

- Agreement: each honest replica's output are consistent
- Totality: any node with a valid pair (***id***, ***σ***) implies all honest nodes will output the same valid pair
- Validity: if sender is legit, all honest replicas reply with the same valid pair
- Succinctness: length of σ is invariant to the length of ***v***

PRBC Protocol🤓:

- Value broadcast phase: sender inputs value to RBC protocol
- Output value phase: receiving honest nodes send threshold shared signature *id* to everyone
- Output signature phase: upon receiving ***f + 1*** legit signature from ☝️ , combine these signatures into a threshold signature ***σ*** of ***id*** and output it 💨

Dumbo2: PRBC(Provable Reliable BroadCast) Protocol 📡

Algorithm 2 The PRBC_{id} protocol with epoch r (for party P_i, where the sender is P_s and $id = \langle r, s \rangle$)

```
Let RBC_id refer to the instance of the reliable broadcast protocol, where P_s is the sender of RBC_id; {DS_s} = ∅.
if P_i = P_s then
    upon receiving input value v_s do
        input {Value, v_s} to RBC_id;
upon receiving Value message {Value, v} from RBC_id do
    σ_is ← SigShare_{f+1} (sk_i, id);
    multicast (Done, id, σ_is);
upon receiving a Done message (Done, id, σ_js) from node P_j for the first time do
    if ShareVerify_{f+1}(id, (j, σ_js)) =1 then
        DS_s ← DS_s ∪ {j, σ_js};
upon |DS_s| = f + 1 do
    σ_s ← Combine_{f+1}(id, DS_s);
    return (Finish, id, σ_s).
```

Algorithm 2 PRBV Protocol at Epoch🌝 r

👉

Algorithm 3 Dumbo2-ACS Protocol

👉

Algorithm 3 The Dumbo2-ACS protocol (for party P_i) in consecutive epoch r

```
Let {PRBC_⟨r,j⟩}_n refer to n instance of provable reliable broadcast protocol, where P_j is the sender of PRBC_⟨r,j⟩, and the Q be the following predicate:
Q_r[{(s_1, σ_1), (s_2, σ_2), ··· , (s_n, σ_n)}] ≡ (at least n − f distinct i satisfy s_i ≠ ⊥ and Verify_{f+1}(⟨r, s_i⟩ , σ_i) = 1).
Initial: W = {(s_1, σ_1), (s_2, σ_2), ··· , (s_n, σ_n)}, where (s_j, σ_j) ← (⊥, ⊥) for all 1 ⩽ j ⩽ n; FS = 0.
upon receiving input value v_i do
    input {Value, v_i} to PRBC_⟨r,i⟩;
upon receiving a Finish message (Finish, ⟨r, j⟩ , σ_j) do
    (s_j, σ_j) ← (j, σ_j);
    FS = FS + 1;
upon FS = n − f do
    propose W for the MVBA_r;
    wait the MVBA_r to return W̄ = {(s̄_1, σ̄_1), (s̄_2, σ̄_2)), ··· , (s̄_n, σ̄_n)}
Let S ⊂ [n] be the set of s̄_j ≠⊥ for 1 ⩽ j ⩽ n.
Wait until receive v_j from PRBC_⟨r,j⟩ for all j ∈ S.
Finally output ∪_{j∈S} v_j.
```

Dumbo2: ACS Protocol🤖

Dumbo2 ACS Premise/Intuition🤨:

Provable Reliable BroadCast + Multi-Valued Validated Byzantine Agreement

W = {(s_1,σ_1), (s_2,σ_2), …, (s_n,σ_n)} are the inputs into the $MVBA_r$ for each node

Dumbo2-ACS Phases🌊:

- Value Broadcast phase: All node input value ***v*** into PRBC and wait for ***n - f*** **Finish**es
- MVBA phase: after receiving ***n - f*** **Finish**es, do MVBA to get W
- Output phase: all honest nodes wait for Value from PRBC using W

Dumbo2-ACS Efficiency👍:

- Message exchanges happen in when all replicas engage in n PRBC runs and second in MVBA runs
- Time **O(1)** due to lack of iterations, each of PRBC and MVBA takes constant time
- Msg **O(n^3)** due to PRBC taking O(n^2) for each instance
- Comm **O(n^2|m| + λn^3log(n))** because of PRBC bottleneck similar to Dumbo1

Table 1. Detailed performance metrics of ACS.

	Complexity‡		
Protocol	Time	Communication	Message
HB-BFT/BEAT0	$\mathcal{O}(\log n)$	$\mathcal{O}(n^2\|m\| + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$
BEAT1/BEAT2	$\mathcal{O}(\log n)$	$\mathcal{O}(n^3\|m\| + \lambda n^3)$	$\mathcal{O}(n^3)$
Dumbo1	$\mathcal{O}(\log \kappa)$	$\mathcal{O}(n^2\|m\| + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$
Dumbo2	$\mathcal{O}(1)$	$\mathcal{O}(n^2\|m\| + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$

EFFICIENCY ANALYSIS

Table 1: Detailed performance metrics of ACS.

	Complexity‡		
Protocol	Time	Communication	Message
HB-BFT/BEAT0	$O(\log n)$	$O(n^2\lvert m\rvert + \lambda n^3 \log n)$	$O(n^3)$
BEAT1/BEAT2	$O(\log n)$	$O(n^3\lvert m\rvert + \lambda n^3)$	$O(n^3)$
Dumbo1	$O(\log \kappa)$	$O(n^2\lvert m\rvert + \lambda n^3 \log n)$	$O(n^3)$
Dumbo2	$O(1)$	$O(n^2\lvert m\rvert + \lambda n^3 \log n)$	$O(n^3)$

Efficiency Goals: Improve the performance of asynchronous BFT protocols by reducing the number of ABA (Asynchronous Binary Agreement) instances.

Experimental Environment

Platform: Conducted on 100 Amazon EC2 t2.medium instances.

Deployment: Instances were distributed across 10 different regions globally.

Compared Protocols: Dumbo1, Dumbo2, and HoneyBadgerBFT.

Performance Metrics: Focused on latency and throughput

Testing Scenarios: Evaluated performance with varying batch sizes

Latency

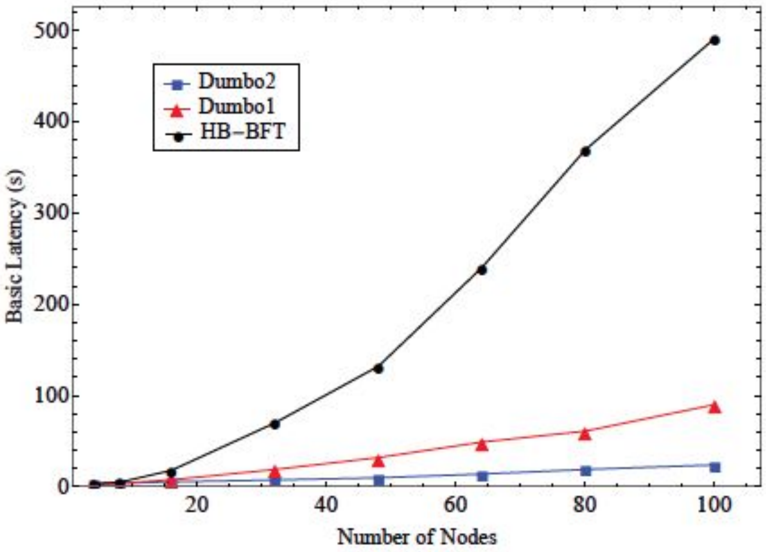


Figure 6: Basic latency of Dumbo1/2 and HoneyBadgerBFT

The reduction in ABA instances in Dumbo protocols significantly reduces latency, particularly for larger network sizes

Throughput

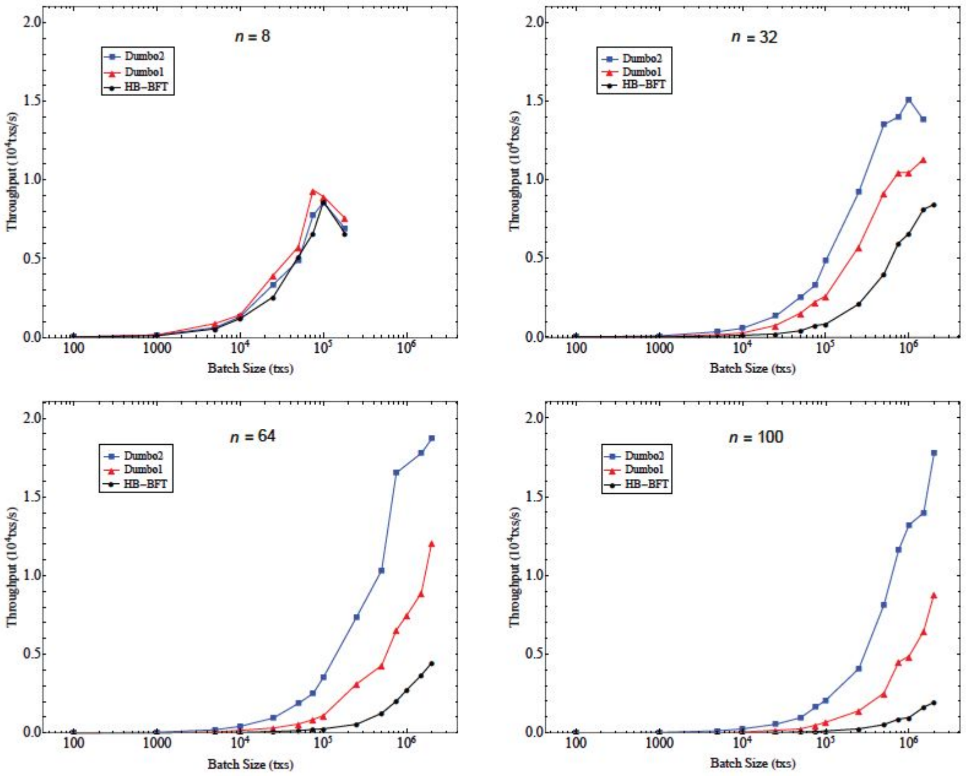


Figure 7: Throughput of Dumbo1/2 and HoneyBadgerBFT

The throughput of protocols get larger with the increase of batch size if the bandwidth and computing resources are sufficient.

Trades off between Latency and Throughput

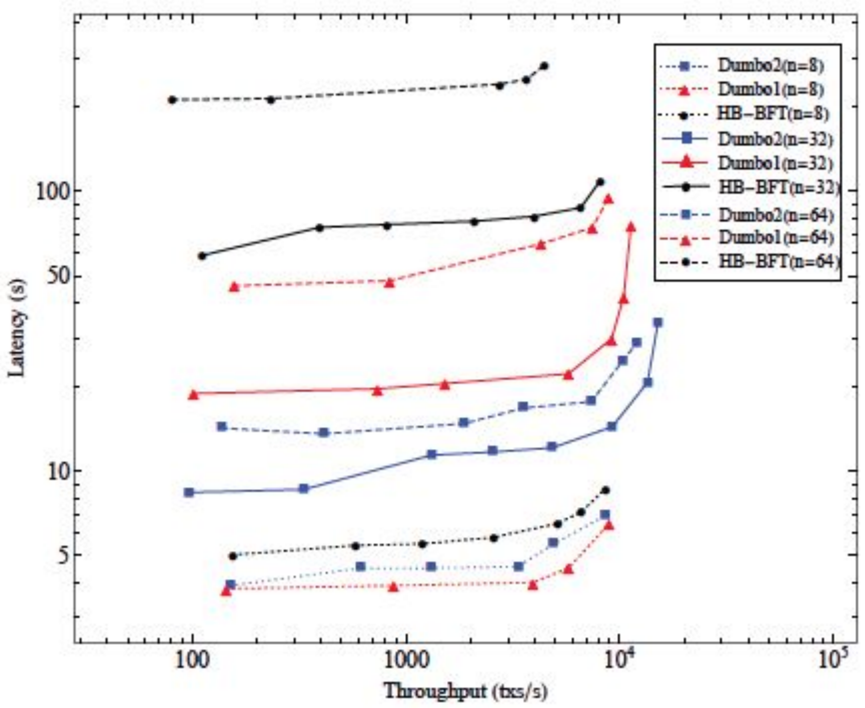


Figure 8: Throughput vs. Latency

For all protocols, latency grows with the increase of throughput, but the growth rate is obviously accelerating.

Conclusion

Dumbo1 and Dumbo2 present significant improvements over HoneyBadgerBFT by optimizing the number of ABA instances.

Dumbo1 achieves better efficiency through selective ABA, while Dumbo2 pushes the boundary further by leveraging MVBA to reduce runtime to constant.

Practical Impact: Both protocols achieve notable improvements in latency and throughput, especially in larger network scales.

Future Work

Further Optimization: Explore combining BEAT protocols' component optimizations with Dumbo protocols.

Adversarial Networks: Investigate robustness and performance in highly adversarial network conditions.