UCDAVIS

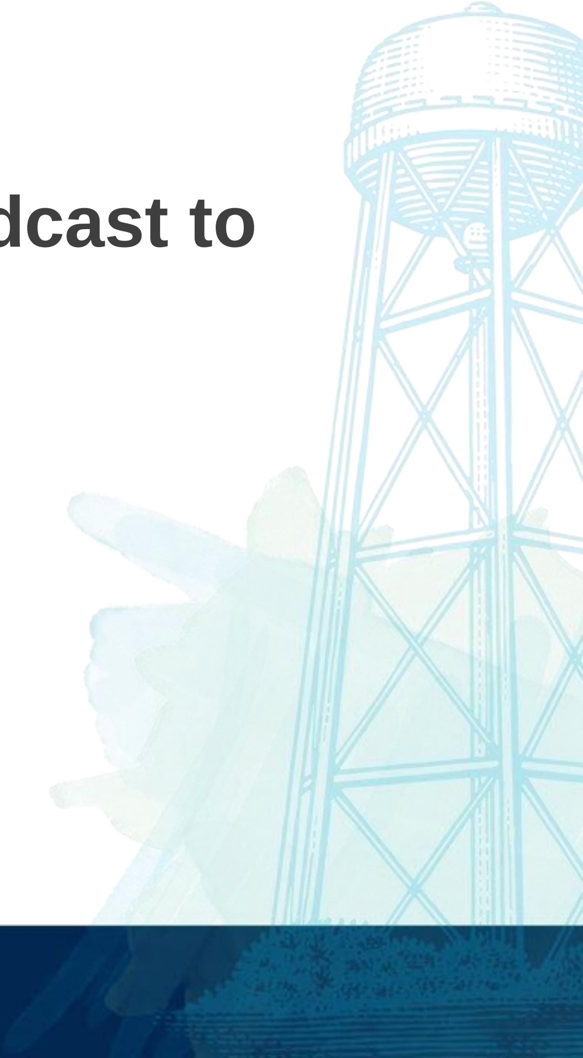
Chop Chop: Byzantine Atomic Broadcast to the Network Limit

Authors

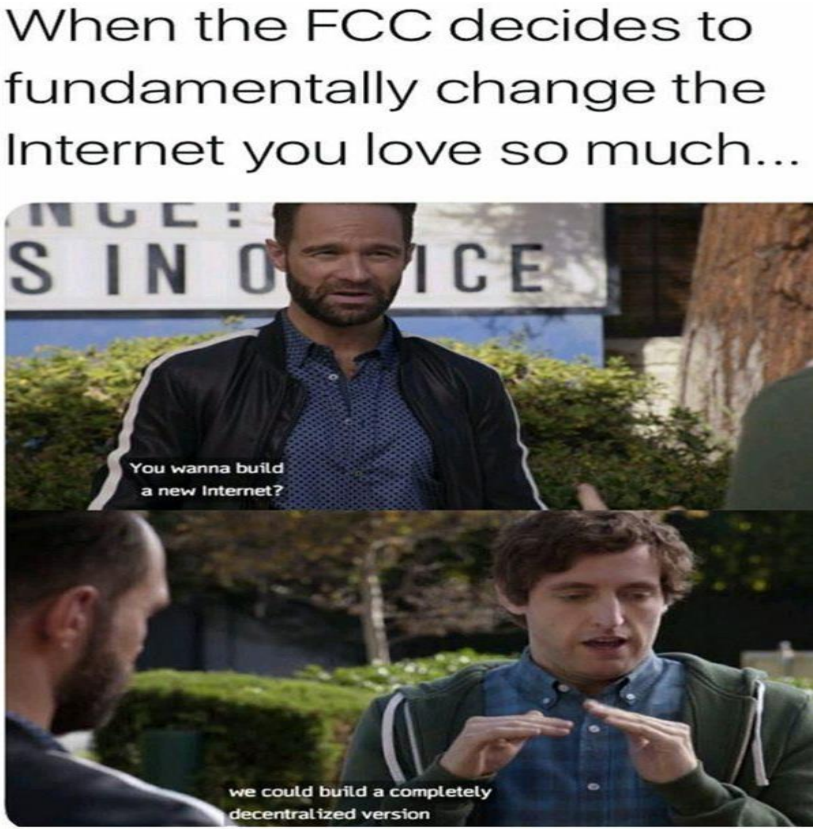
Martina Camaioni, Rachid Guerraoui, Matteo Monti, Pierre-Lois Roman, Manuel Vidigueira, Gaitheir Voron

Presenters

Aunsh Bandivadekar and Srishti Singh

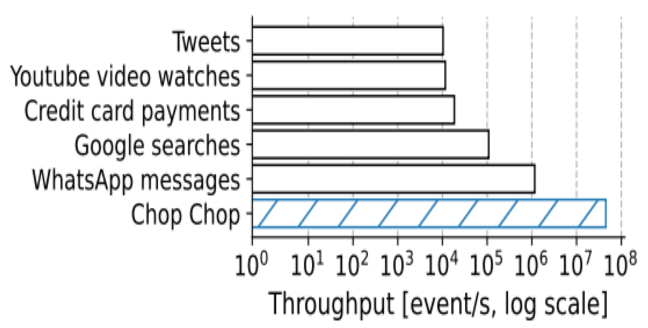


Introduction


When the FCC decides to fundamentally change the Internet you love so much...
S IN O ICE
You wanna build a new Internet?
we could build a completely decentralized version

Some Problems:

- Performance
- Scalability
- Consistency
- Attack Resistance
- Complexity
- Operational Management



Introduction

Atomic Broadcast

A fundamental communication primitive that orders, authenticates, and deduplicates messages.

Properties:

- Agreement: Correct servers deliver the same messages in the same order
- Validity: Messages from correct clients are eventually delivered.
- Integrity: Spurious messages cannot be attributed to correct clients.
- No Duplication: No message is delivered more than once.

Cost of Atomic Broadcast

Integrity : Digital Signatures

No duplication: Sequence Number

Overload on the application layer

Batching

For each message: 1 public key (32 B [Ed25519]) + Signature (64B) + Sequence Number (8B) + Message (4B) ~ 91% overhead

Narwhal-Bullshark

- Not handle authentication natively
- Trusted workers become bottlenecks
- No sequence number guarantees

Narwhal-Bullshark-sig

- Workers verify signatures but still workers are linked to server groups

UCDAVIS

Chop Chop

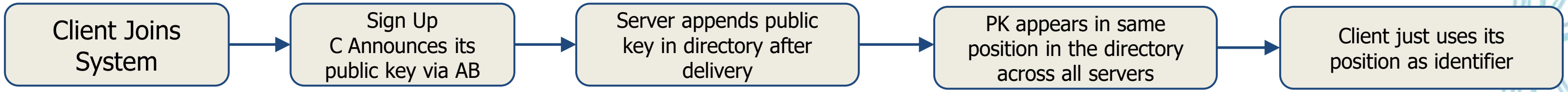
- A Byzantine Atomic Broadcast system that amortizes the cost of ordering, authenticating and deduplicating messages, achieving “line rate”.
- As a mempool, Chop Chop uses an underlying instance of Atomic Broadcast to order batches, aggregating messages to amortize costs.
- Chop Chop addresses the shortcoming of authentication and deduplications with a new form of batches: distilled batches.
- Chop Chop integrates the two following techniques to reduce the bandwidth and CPU cost of authentication.
  - Short identifiers
  - Pooled signature verification

Mitigations

Reduce BW and CPU cost of authentication

Short Identifiers

Repeated Public Keys consume a lot of server comm. budget



Client Joins System

Sign Up
C Announces its public key via AB

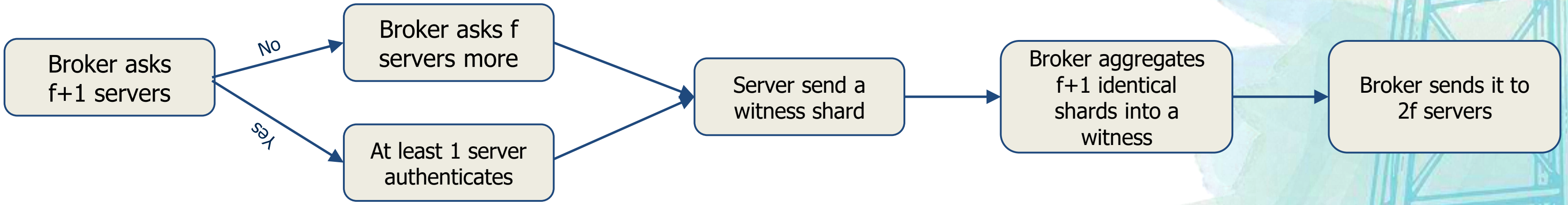
Server appends public key in directory after delivery

PK appears in same position in the directory across all servers

Client just uses its position as identifier

Pooled signature verification*

Authenticating a batch by verifying its signatures is a computationally intensive task for a server



Broker asks f+1 servers

No

Broker asks f servers more

Yes

At least 1 server authenticates

Server send a witness shard

Broker aggregates f+1 identical shards into a witness

Broker sends it to 2f servers

Result: In a 3f+1 server system, only f+1 servers verify the signatures. 2f servers avoid operation. 66% overhead reduced.

*Not all servers need to authenticate the signatures as shown by Red Belly and Mir

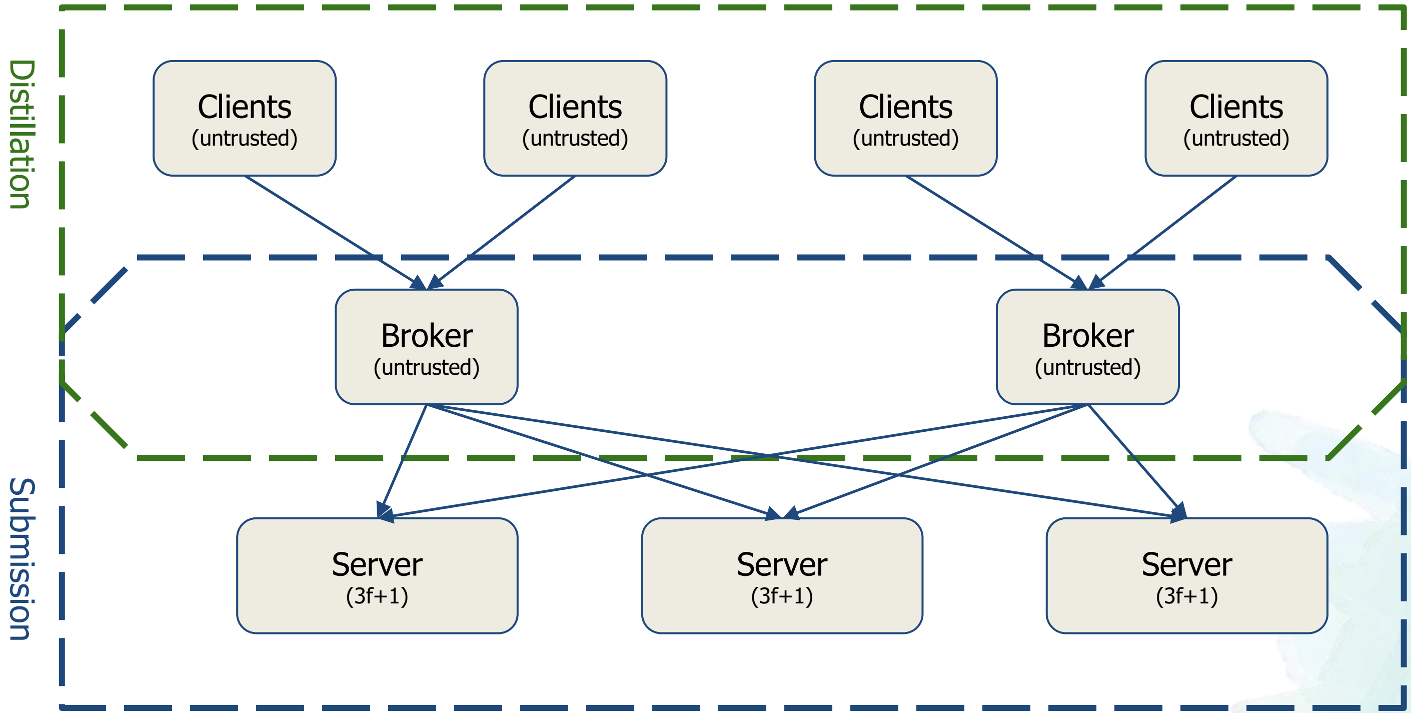
Architecture

Chop Chop involves three types of processes :

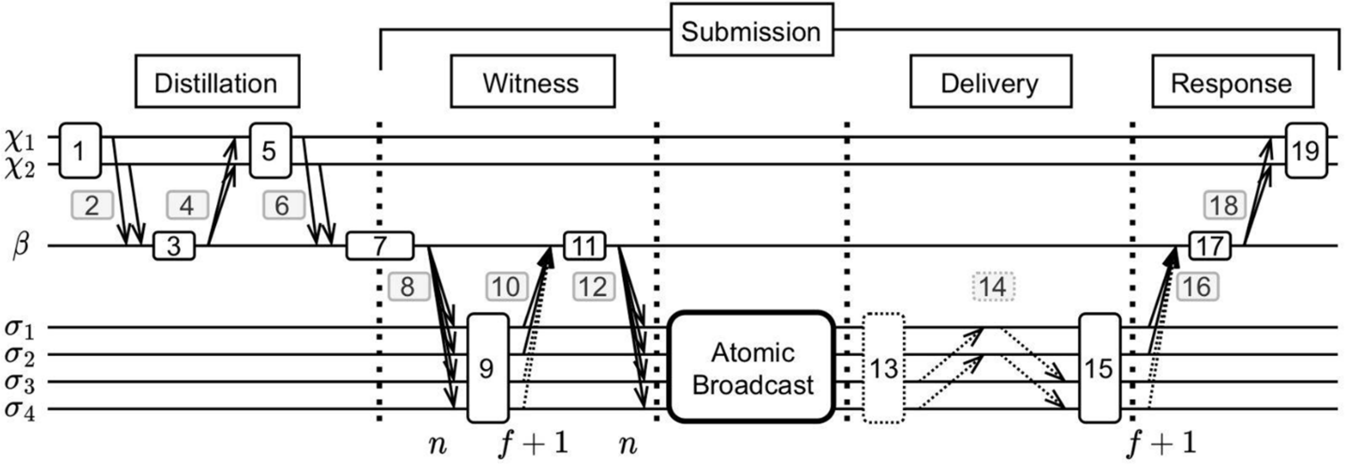
- Broadcasting clients
- Delivering servers
- A layer of broadcast-facilitating brokers

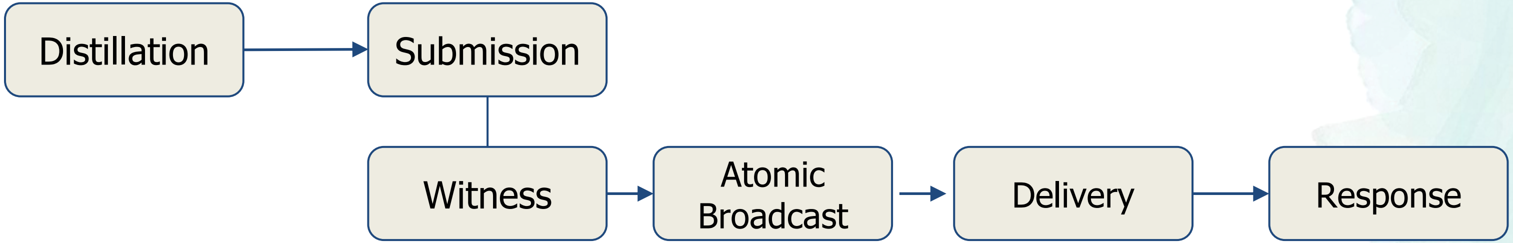
Chop Chop’s protocol unfolds in two phases:

- Distillation
- Submission


Distillation
Submission
Clients
(untrusted)
Clients
(untrusted)
Clients
(untrusted)
Clients
(untrusted)
Broker
(untrusted)
Broker
(untrusted)
Server
(3f+1)
Server
(3f+1)
Server
(3f+1)

Overview: Chop Chop protocol




Distillation

Submission

Witness

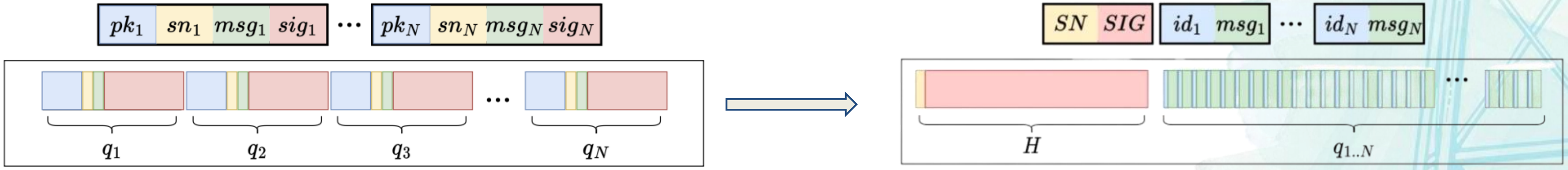
Atomic Broadcast

Delivery

Response

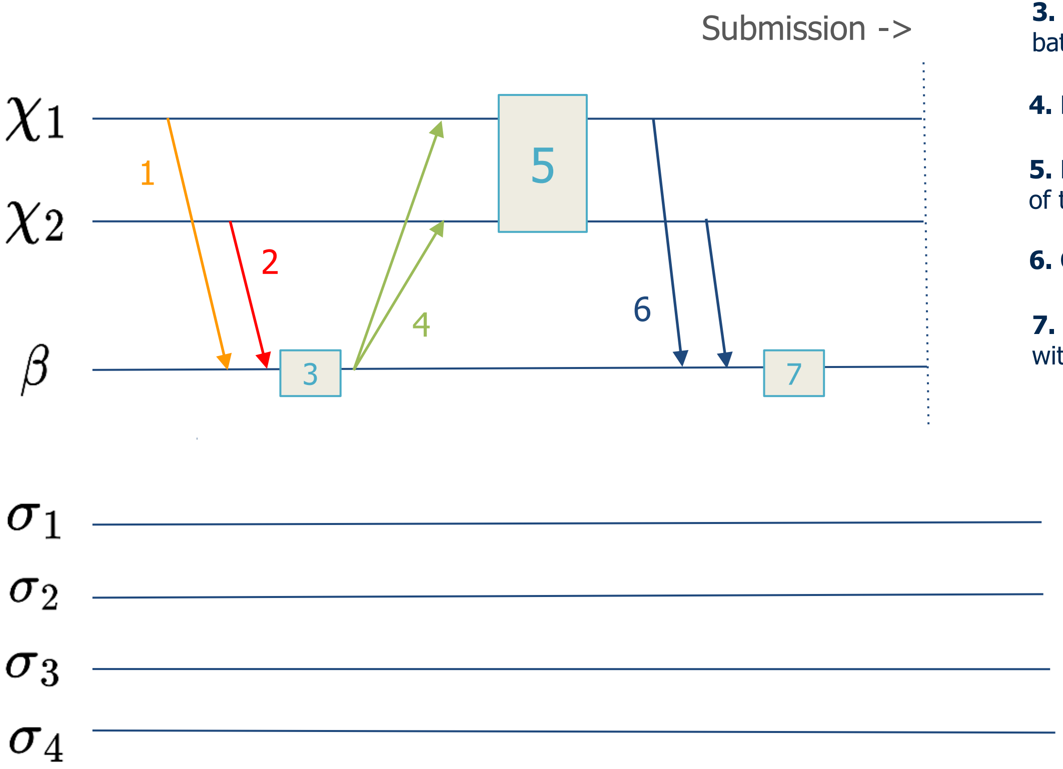
Distillation

- In the distillation phase, clients interact with a broker to gather their messages in a distilled batch
- With classic authentication and sequencing, each payload q_i contains a public key pk_i , a sequence number sn_i , a message msg_i and a signature s_ig_i.
- In the fully distilled case, each q_i reduces to just id_i and msg_i: one header H, composed of one aggregate sequence number SN and one aggregate signature SIG, is sufficient for the entire batch.



pk_1 sn_1 msg_1 sig_1 ··· pk_N sn_N msg_N sig_N
q_1
q_2
q_3
···
q_N
SN SIG id_1 msg_1 ··· id_N msg_N
···
H
$q_{1..N}$

Distillation



1 and 2. Clients send messages (m_1,….m_b) with sequence numbers (k_1,…..k_b) to the broker β

3. Broker β identifies the max sequence number k and builds a batch proposal B = [(x_1,k,m_1),…..(x_b,k,m_b)]

4. Broker sends the batch proposal back to the clients

5. Each client X_i produces a multi-signature s_i on H(B) [Sign the root of tree]

6. Clients send the signed batch back to Broker

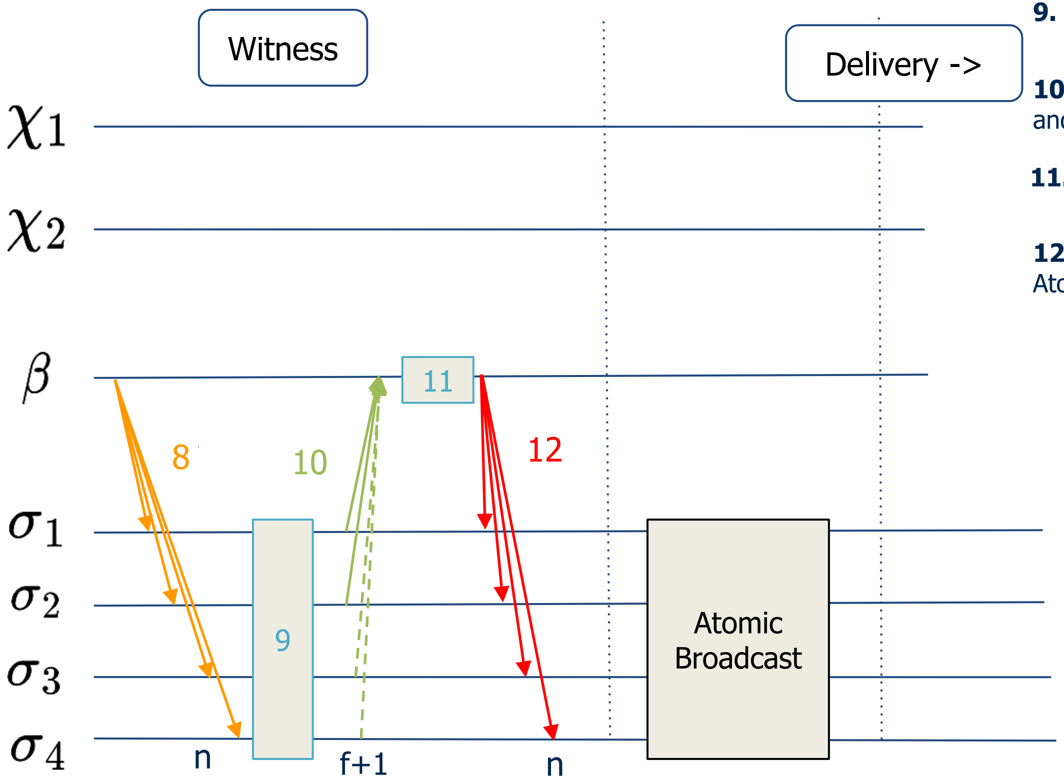
7. Broker produces a distilled batch B~ = [s, k, ((x_1,m_1),…..(x_b,m_b))] with aggregated signature s and sequence number k

χ_1	Client 1
χ_2	Client 2
β	Broker
σ_1	Servers (1…n)

Challenges during Distillation

1. What if a broker forges messages?
2. Can a broker avoid sending the entire batch?
3. What if a client does not multi-sign? (Fault-tolerant distillation)
4. What if a client broadcasts too frequently?
5. What if a client submits the largest possible sequence number?(Legitimacy Numbers | Legitimacy Proofs)
6. What if a broker crashes?

Submission: Witness & Atomic Broadcast



8. In order to collect a witness for $B^{\sim}$, β sends $B^{\sim}$ to all servers. β asks only f+1 servers to sign a witness shard

9. Upon receiving $B^{\sim}$, a correct server σ stores $B^{\sim}$.

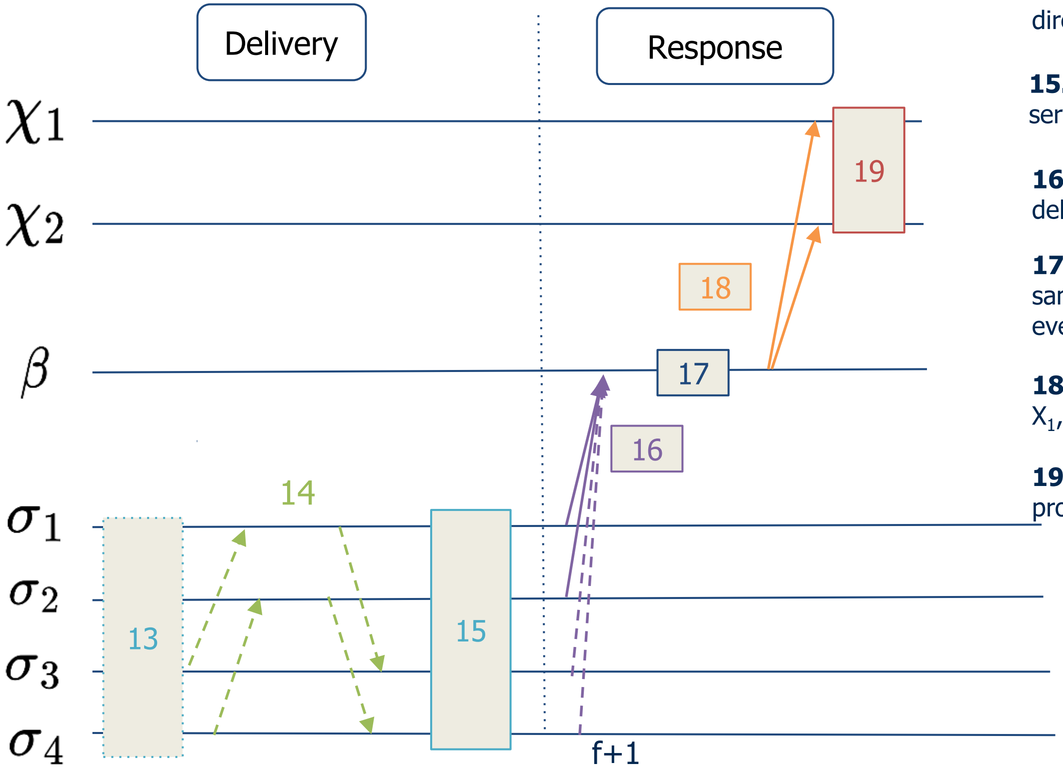
10. If asked to witness $B^{\sim}$, σ checks that $B^{\sim}$ is well-formed and sends back to β its witness shard for $B^{\sim}$

11. β collects and aggregates f +1 shards into a witness for B

12. β then submits $B^{\sim}$'s hash and witness to the server-run Atomic Broadcast

$B^{\sim}$	Distilled Batch
χ_1	Client 1
χ_2	Client 2
β	Broker
σ_1	Servers (1…n)

Submission: Delivery & Response



13. σ deliver B˜'s hash and witness from Atomic Broadcast

14. A correct server σ retrieves B˜, either from its local storage (if it directly received B˜ from β at #8) or from another server

15. As B˜ is retrievable, σ is guaranteed to eventually find a server to pull B˜ from which it retrieves

16. σ signs a delivery certificate, listing the messages in B˜ that σ delivered and sends its signature back to β

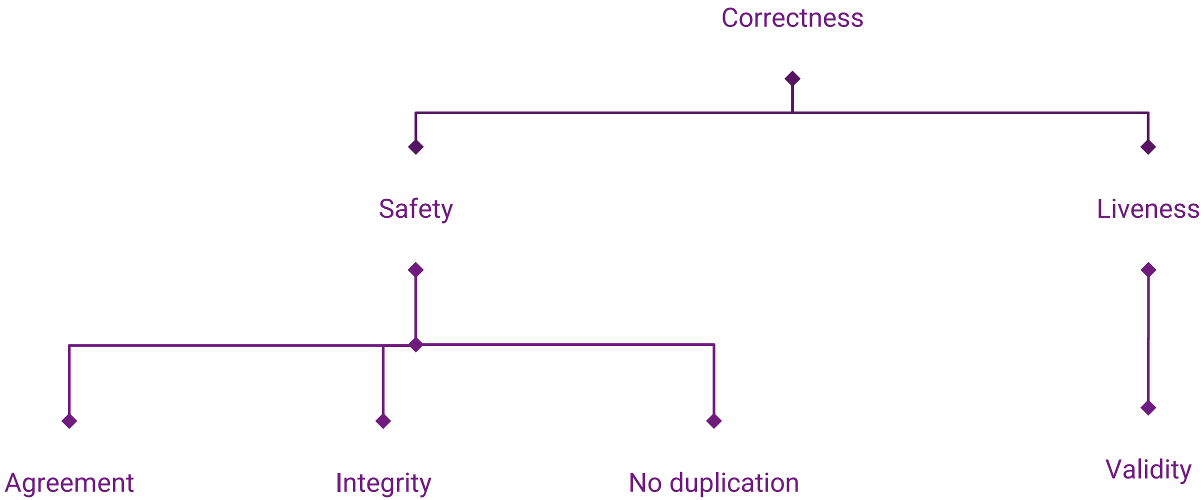
17. By agreement of Atomic Broadcast, all correct servers deliver the same subset of messages in B˜. As such, β is guaranteed to eventually collect f +1 signatures on the same delivery certificate

18. Upon doing so, β distributes a copy of B˜'s delivery certificate to $X_1,\dots,X_b$

19. Armed with B˜'s delivery certificate, a correct X_i can publicly prove the delivery of m_i and safely broadcast its next message

χ_1	Client 1
χ_2	Client 2
β	Broker
σ_1	Servers (1…n)

Correctness



Implementation

- Implemented in Rust, totalling 8900 lines of code.

- Main libraries

1. *tokio* for an asynchronous, event-based runtime
2. *rayon* for worker based parallel computation
3. *serde* for serialization and deserialization
4. *blake3* for cryptographic hashes
5. *ed25519-dalek* for EdDSA signatures
6. *blst* multi-signatures on the BLS12-381 curve
7. In-house libraries *talk* - for basic distributed computing and high-level networking and cryptography
8. *zebra* Merkel-tree based data structures

Evaluation

Servers:

- 64 c6i.8xlarge AWS machines
- 32 vCPUs, 16 physical cores per machine
- Spread across 14 regions globally

Batches:

- 65,536 messages per batch
- 8 bytes per message
- 736 KB per batch

Resilience:

- Tolerates up to 21 out of 64 faulty servers

Clients:

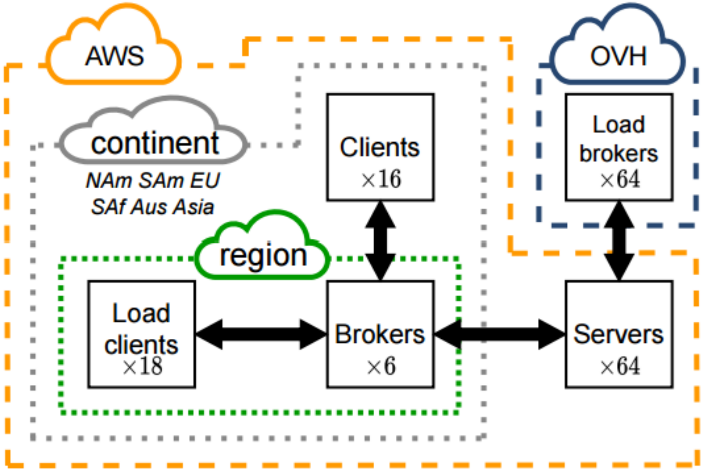
- 16 small T3 AWS machines across regions
- Produce load up to 257 million clients

Load Clients & Brokers:

- 320 large AWS & 64 OVH machines
- Simulate high load on servers and brokers

Networking:

- Geo-distributed
- High bandwidth connections
- TCP tuned for high throughput



Evaluation

Let’s evaluate Chop Chop

1. What workload can Chop Chop sustain?

1. What are the benefits of Chop Chop’s distillation?

1. How does Chop Chop scale to different numbers of servers?

1. How efficiently does Chop Chop use resources overall?

1. How does Chop Chop perform under adverse conditions, such as server failures?

1. What performance can applications achieve using Chop Chop?

Eval: Load Handling

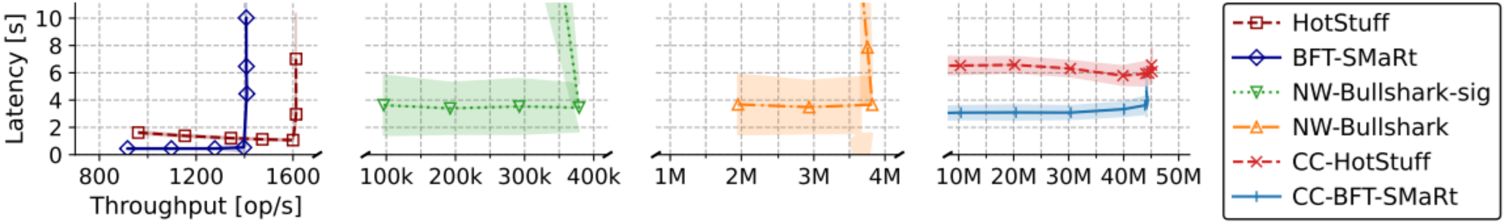


Fig. shows Latency v Throughput for various input rates of 8B messages

Baselines:

- BFT-SMaRt and HotStuff peak at 1.6K ops/sec
- Narwhal-Bullshark reaches 3.8M ops/sec
- Narwhal-Bullshark-sig peaks at 382K ops/sec

Chop Chop achieves:

- 43.6M ops/sec with BFT-SMaRt
- 44M ops/sec with HotStuff
- Latency between 3 and 6 seconds

Trade Offs:

- Chop Chop trades higher latency for maximum throughput
- Batching contributes to latency
- Still comparable to Narwhal's latency

Eval: Distillation

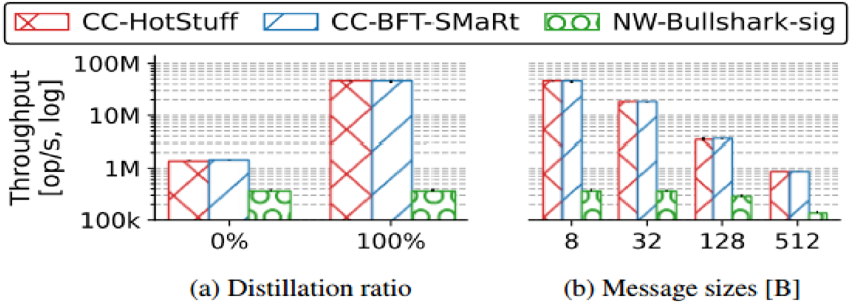


Fig a. Throughput with 0% and 100% Distillation
Fig b. Throughput with varying message sizes

Distillation vs Mitigation

- 1.5M ops/sec with mitigation techniques
- 44M ops/sec with full distillation
- 29x speedup from distillation

Distillation for Larger Messages

- Chop Chop sustains line-rate for varying message sizes
- Throughput scales linearly with message size
- Consistently faster than baseline

Eval: Distillation (Line Rate)

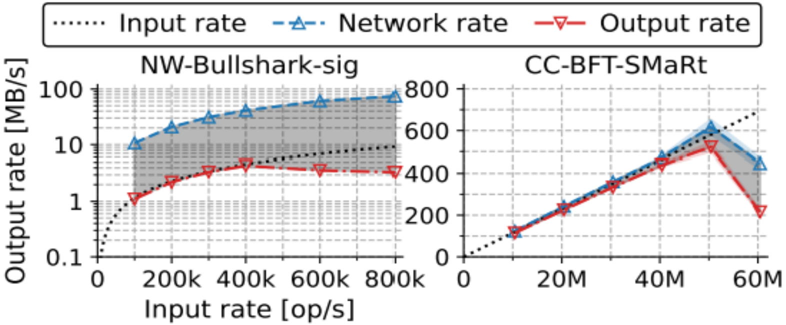


Fig. Illustrates Chop Chop's near line-rate network use by depicting its input, network and output rates:

Input Rate:

- Measures total rate of useful information (messages) entering brokers/servers fr clients
- Reflects throughput of atomic broadcast usage

Network Rate:

- Measures all information (including protocol overhead) ingress servers receive
- Input rate + overhead from cryptography, ordering, etc.

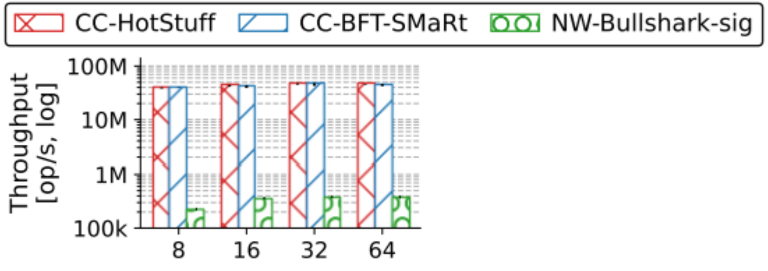
Output Rate:

- The "goodput" - useful application-level throughput delivered by servers
- Messages per second finally delivered to receiving applications

Line Rate

- Under 8% overhead beyond raw network usage
- Baseline has over 90% overhead
- Distillation achieves effective line rate

Eval: Number of Servers

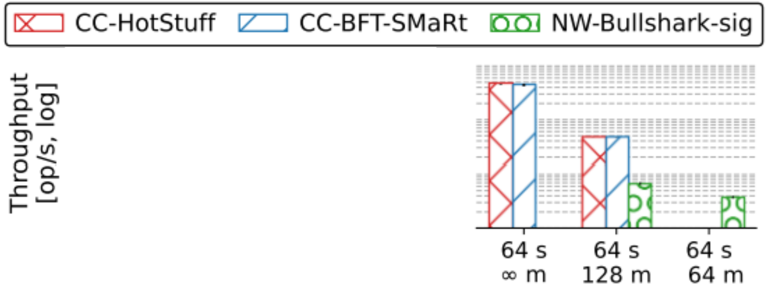


(a) System sizes

Fig a. Throughput varying system sizes

- System sizes evaluated: 8, 16, 32 and 64 servers
- Both Chop Chop and Narwhal-Bullshark-sig scale well upto 64 servers
- Narwhal-Bullshark-sig only scales vertically. If a server or workers are faulty the entire server group is compromised
- Chop Chop also scales horizontally with more brokers

Eval: Overall Efficiency

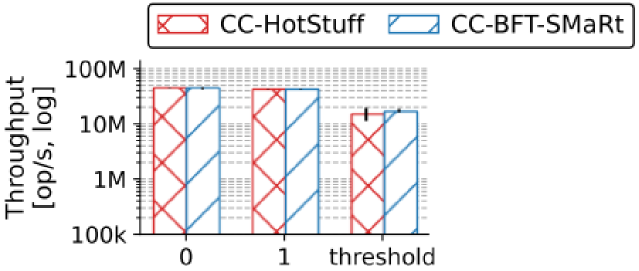


(b) Matching resources

Fig a. Throughput with varying number of machines when resources are kept constant

- Hardware resources matched between systems
- Chop Chop: 64 servers & 64 brokers
- Narwhal-Bullshark-sig: 128 workers
- Chop Chop achieves 4.6M ops/sec
- Narwhal-Bullshark-sig reaches 679K ops/sec
- Chop Chop has higher throughput with same resources
- Untrusted brokers provide additional scalability. Trusted workers contribute to its own server group.

Eval: Chop Chop Under Failures



(a) Server failures

Fig a. Throughput with various server failures (when failing 30s into the run)

Failures Evaluated:

- Crash of servers
- Clients not participating in distillation

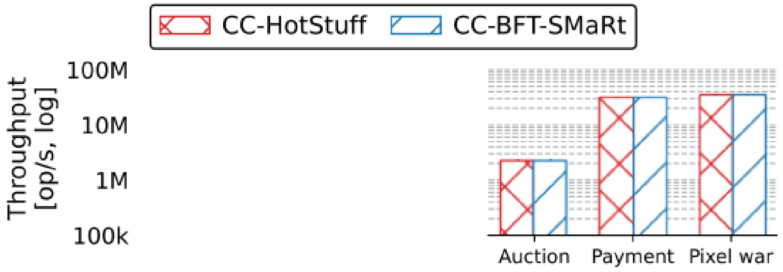
Effect of Server Crashes:

- 43M ops/sec with no crashes
- Marginal drop to 42M ops/sec with 1 crash
- 66% reduction to 15M ops/sec with max failures

Effect of No Distillation:

- Max throughput drops from 44M ops/sec to 1.5M ops/sec
- Worst case without distillation

Eval: Application Use Cases



(b) Applications

Fig b. Throughput with various application use cases

Apps:

- Payment: 32 M op/s
- Auction: 2.3 M op/s
- Pixel War: 35 M op/s

Features:

- Bottleneck is the application itself
- CC has sufficient capacity for high, single-application throughput
- Can support many separate high-throughput apps
- Applications only handle core logic, Chop Chop provides messaging fabric

Future Work

- Chop Chop's rapid throughput makes memory management difficult: servers quickly fill their memory if they are unable to garbage-collect under severe load.

- All Chop Chop servers are known at startup, and it is unknown whether its performance would be maintained when distributed on hundreds of servers.

- Future research could focus on sharding to obtain even higher throughput by operating many, independent, coordinated instances of Chop Chop, as well as delegating more jobs to the brokers, such as public key aggregation.

Conclusion

- Chop Chop accelerates the throughput of atomic broadcast protocols by orders of magnitude via a novel distillation technique and untrusted brokers
- Distilled batches amortize expensive authentication and deduplication over large message sets by aggregating signatures and sequence numbers.
- An untrusted layer of brokers performs heavy cryptographic operations like distillation outside the trust boundary. This prevents wasting scarce and expensive trusted resources.
- Together these innovations enable a geo-distributed deployment to achieve 43 million operations per second with just 64 servers. This gets within 8% of theoretical line rate.
- The raw throughput unlocks new potential decentralized applications in finance, communication, gaming etc. that need global scale.
- Compared to its closest competitors, Chop Chop improves throughput by up to 100x.

THANK YOU