MiniCast: Minimizing the Communication Complexity of Reliable Broadcast

Victor Shoup (Offchain Labs)

Joint work with *Thomas Locher* (Dfinity)

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

One of $P_1, \ldots, P_n$ is the *designated sender* S

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

One of $P_1, \ldots, P_n$ is the *designated sender* S

Protocol should S allow to broadcast a *single* message m to $P_1, \ldots, P_n$

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

One of $P_1, \ldots, P_n$ is the *designated sender* S

Protocol should S allow to broadcast a *single* message m to $P_1, \ldots, P_n$

Correctness property:

> *All honest parties that output a message must output the same message.*

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

One of $P_1, \ldots, P_n$ is the *designated sender* S

Protocol should S allow to broadcast a *single* message m to $P_1, \ldots, P_n$

Correctness property:

> *All honest parties that output a message must output the same message. Moreover, if S is honest, that message is the one input by S.*

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

One of $P_1, \ldots, P_n$ is the *designated sender* S

Protocol should S allow to broadcast a *single* message m to $P_1, \ldots, P_n$

Correctness property:

> *All honest parties that output a message must output the same message. Moreover, if S is honest, that message is the one input by S.*

Completeness property:

> *If any honest party outputs a message*

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

One of $P_1, \ldots, P_n$ is the *designated sender* S

Protocol should S allow to broadcast a *single* message m to $P_1, \ldots, P_n$

Correctness property:

> *All honest parties that output a message must output the same message. Moreover, if S is honest, that message is the one input by S.*

Completeness property:

> *If any honest party outputs a message or an honest sender S inputs a message,*

Reliable Broadcast

We have n parties $P_1, \ldots, P_n$ connected by *authenticated* point-point-channels

Communication network is *asynchronous* and controlled by the adversary

At most $t < n/3$ parties are corrupt (Byzantine)

One of $P_1, \ldots, P_n$ is the *designated sender* S

Protocol should S allow to broadcast a *single* message m to $P_1, \ldots, P_n$

Correctness property:

> *All honest parties that output a message must output the same message. Moreover, if S is honest, that message is the one input by S.*

Completeness property:

> *If any honest party outputs a message or an honest sender S inputs a message, then* eventually *all honest parties output a message.*

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \log n\ )$

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n\ +\ \lambda \cdot n^2 \log n\ )$

[$\lambda =$ hash output length]

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \log n \ )$

[$\lambda =$ hash output length]

Das, Xiang & Ren [2021]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \ )$

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \log n \ )$

[$\lambda =$ hash output length]

Das, Xiang & Ren [2021]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \ )$

Abraham & Asharov [2022]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \ )$

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n\ +\ \lambda \cdot n^2 \log n\ )$

[$\lambda =$ hash output length]

Das, Xiang & Ren [2021]: $O(\ |m| \cdot n\ +\ \lambda \cdot n^2\ )$

Abraham & Asharov [2022]: $O(\ |m| \cdot n\ +\ \lambda \cdot n^2\ )$ [statistically secure]

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \log n \ )$

[$\lambda =$ hash output length]

Das, Xiang & Ren [2021]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \ )$

Abraham & Asharov [2022]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \ )$ [statistically secure]

Long messages: worrying about big-O constants

All of the above cost at least $3 \cdot |m| \cdot n + \ldots$

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \log n\ )$

[$\lambda =$ hash output length]

Das, Xiang & Ren [2021]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2\ )$

Abraham & Asharov [2022]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2\ )$ [statistically secure]

Long messages: worrying about big-O constants

All of the above cost at least $3 \cdot |m| \cdot n + \ldots$

Locher [2024]: $2 \cdot |m| \cdot n + O(\lambda \cdot n^2 \log n)$

Message and communication complexity

Message complexity: total number of *messages* sent from all honest parties to any party

Bracha: $O(n^2)$

Communication complexity: total number of *bits* sent from all honest parties to any party

Bracha: $O(|m| \cdot n^2)$

Cachin & Tessaro [2005]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2 \log n\ )$

[$\lambda =$ hash output length]

Das, Xiang & Ren [2021]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2\ )$

Abraham & Asharov [2022]: $O(\ |m| \cdot n \ + \ \lambda \cdot n^2\ )$ [statistically secure]

Long messages: worrying about big-O constants

All of the above cost at least $3 \cdot |m| \cdot n + \ldots$

Locher [2024]: $2 \cdot |m| \cdot n + O(\lambda \cdot n^2 \log n)$

This work: $1.5 \cdot |m| \cdot n + O(\lambda \cdot n^2 \log n)$

Where does this 3× "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$**-erasure code**:

encode a message m as a vector of "fragments" $f_1, \ldots, f_n$

Where does this $3\times$ "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$-**erasure code**:

- encode a message m as a vector of "fragments" $f_1, \ldots, f_n$
- any $n-2t$ fragments can be used to recover m

Where does this 3× "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$**-erasure code**:

- encode a message m as a vector of "fragments" $f_1, \ldots, f_n$
- any $n-2t$ fragments can be used to recover m
- fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

Where does this 3× “blowup” come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$-**erasure code**:

- encode a message m as a vector of “fragments” $f_1, \ldots, f_n$
- any $n-2t$ fragments can be used to recover m
- fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

Where does this 3× "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$-**erasure code**:

- encode a message m as a vector of "fragments" $f_1, \ldots, f_n$
- any $n-2t$ fragments can be used to recover m
- fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

- S encodes m as $f_1, \ldots, f_n$ and sends each party P_i its fragment f_i

Where does this 3× "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$**-erasure code**:

- encode a message m as a vector of "fragments" $f_1, \ldots, f_n$
- any $n-2t$ fragments can be used to recover m
- fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

- S encodes m as $f_1, \ldots, f_n$ and sends each party P_i its fragment f_i
- Each party P_i broadcasts its fragment f_i

Where does this 3× “blowup” come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$-**erasure code**:

- encode a message m as a vector of “fragments” $f_1, \ldots, f_n$
- any $n-2t$ fragments can be used to recover m
- fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

- S encodes m as $f_1, \ldots, f_n$ and sends each party P_i its fragment f_i
- Each party P_i broadcasts its fragment f_i
- Each party outputs a message m when

Where does this 3× “blowup” come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$-**erasure code**:

- encode a message m as a vector of “fragments” $f_1, \ldots, f_n$
- any $n-2t$ fragments can be used to recover m
- fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

- S encodes m as $f_1, \ldots, f_n$ and sends each party P_i its fragment f_i
- Each party P_i broadcasts its fragment f_i
- Each party outputs a message m when
  - it receives $n-2t$ fragments, and

Where does this 3× "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$-**erasure code**:

- encode a message m as a vector of "fragments" $f_1, ..., f_n$
- any $n-2t$ fragments can be used to recover m
- fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

- S encodes m as $f_1, ..., f_n$ and sends each party P_i its fragment f_i
- Each party P_i broadcasts its fragment f_i
- Each party outputs a message m when
  - it receives $n-2t$ fragments, and
  - $n-t$ parties tell it they are holding fragments

Where does this 3× "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$**-erasure code**:

encode a message m as a vector of "fragments" $f_1, \ldots, f_n$

any $n-2t$ fragments can be used to recover m

fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

S encodes m as $f_1, \ldots, f_n$ and sends each party P_i its fragment f_i

Each party P_i broadcasts its fragment f_i

Each party outputs a message m when

- it receives $n-2t$ fragments, and
- $n-t$ parties tell it they are holding fragments
    - *of which maybe only $n-2t$ are honest and actually broadcast their fragment*

Where does this 3× "blowup" come from?

All of the earlier (pre-Locher) work uses an $(n, n-2t)$**-erasure code**:

encode a message m as a vector of "fragments" $f_1, \ldots, f_n$

any $n-2t$ fragments can be used to recover m

fragments are of size $\approx |m|/(n-2t) < 3|m|/n$

The idea:

S encodes m as $f_1, \ldots, f_n$ and sends each party P_i its fragment f_i

Each party P_i broadcasts its fragment f_i

Each party outputs a message m when

- it receives $n-2t$ fragments, and
- $n-t$ parties tell it they are holding fragments
  - *of which maybe only $n-2t$ are honest and actually broadcast their fragment*
  - **this is why we need a reconstruction threshold of only $n-2t$ to maintain completeness**

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Problem: how to maintain completeness?

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Problem: how to maintain completeness?

Locher [2024] gives a protocol that uses an $(n, n-t)$ erasure code

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Problem: how to maintain completeness?

Locher [2024] gives a protocol that uses an $(n, n-t)$ erasure code

... *but* the logic to maintain completeness pushes the blowup to $2\times$

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Problem: how to maintain completeness?

Locher [2024] gives a protocol that uses an $(n, n-t)$ erasure code

...but the logic to maintain completeness pushes the blowup to $2\times$

This work (*very* rough idea):

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Problem: how to maintain completeness?

Locher [2024] gives a protocol that uses an $(n, n-t)$ erasure code

... but the logic to maintain completeness pushes the blowup to $2\times$

This work (*very* rough idea):

- use an $(n, n-t)$ erasure code to encode the message as a vector of fragments

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Problem: how to maintain completeness?

Locher [2024] gives a protocol that uses an $(n, n-t)$ erasure code

. . . but the logic to maintain completeness pushes the blowup to $2\times$

This work (*very* rough idea):

- use an $(n, n-t)$ erasure code to encode the message as a vector of fragments
- use an $(n, n-2t)$ erasure code to encode each fragment as a vector of "mini-fragments"

Reducing blowup using a higher reconstruction threshold

If we use an $(n, n-t)$ erasure code, we can (hopefully) *reduce the blowup to* $1.5\times$

Problem: how to maintain completeness?

Locher [2024] gives a protocol that uses an $(n, n-t)$ erasure code

. . . but the logic to maintain completeness pushes the blowup to $2\times$

This work (*very* rough idea):

- use an $(n, n-t)$ erasure code to encode the message as a vector of fragments
- use an $(n, n-2t)$ erasure code to encode each fragment as a vector of "mini-fragments"
- using mini-fragments, it is possible to arrange that the honest parties help each other obtain their respective fragments *with little communication complexity*

Other features and follow-up work

Other features and follow-up work

- The communication of our protocol is *very well balanced*:

Other features and follow-up work

- The communication of our protocol is *very well balanced*:
  - each party, *including the sender*, transmits $\approx 1.5 \cdot |m| \cdot n$ bits

Other features and follow-up work

- The communication of our protocol is *very well balanced*:
  - each party, *including the sender*, transmits $\approx 1.5 \cdot |m| \cdot n$ bits
  - critical in *asymmetric* protocols, where only one sender broadcasts at a time (e.g., PBFT-like protocols), and where bandwidth is limited

Other features and follow-up work

- The communication of our protocol is *very well balanced*:
  - each party, *including the sender*, transmits $\approx 1.5 \cdot |m| \cdot n$ bits
  - critical in *asymmetric* protocols, where only one sender broadcasts at a time (e.g., PBFT-like protocols), and where bandwidth is limited
- The cost of erasure coding (and other overheads) is insignificant (compared to typical network speeds)

Other features and follow-up work

- The communication of our protocol is *very well balanced*:
  - each party, *including the sender*, transmits $\approx 1.5 \cdot |m| \cdot n$ bits
  - critical in *asymmetric* protocols, where only one sender broadcasts at a time (e.g., PBFT-like protocols), and where bandwidth is limited
- The cost of erasure coding (and other overheads) is insignificant (compared to typical network speeds)
  - erasure coding algorithms and software have come a long way!

    (https://github.com/AndersTrier/reed-solomon-simd)

Other features and follow-up work

- The communication of our protocol is *very well balanced*:
  - each party, *including the sender*, transmits $\approx 1.5 \cdot |m| \cdot n$ bits
  - critical in *asymmetric* protocols, where only one sender broadcasts at a time (e.g., PBFT-like protocols), and where bandwidth is limited
- The cost of erasure coding (and other overheads) is insignificant (compared to typical network speeds)
  - erasure coding algorithms and software have come a long way!

    `(https://github.com/AndersTrier/reed-solomon-simd)`
- The (good case) round complexity is 4 (which is sub-optimal)

Other features and follow-up work

- The communication of our protocol is *very well balanced*:
  - each party, *including the sender*, transmits $\approx 1.5 \cdot |m| \cdot n$ bits
  - critical in *asymmetric* protocols, where only one sender broadcasts at a time (e.g., PBFT-like protocols), and where bandwidth is limited
- The cost of erasure coding (and other overheads) is insignificant (compared to typical network speeds)
  - erasure coding algorithms and software have come a long way!

    (`https://github.com/AndersTrier/reed-solomon-simd`)
- The (good case) round complexity is 4 (which is sub-optimal)
  - Nevertheless, it can be adapted to PBFT-like protocols to have an effective round complexity of 2 (with caveats)

    (“Sing a Song of Simplex”, Shoup [2023])

Other features and follow-up work

- The communication of our protocol is *very well balanced*:
  - each party, *including the sender*, transmits $\approx 1.5 \cdot |m| \cdot n$ bits
  - critical in *asymmetric* protocols, where only one sender broadcasts at a time (e.g., PBFT-like protocols), and where bandwidth is limited
- The cost of erasure coding (and other overheads) is insignificant (compared to typical network speeds)
  - erasure coding algorithms and software have come a long way!

    (`https://github.com/AndersTrier/reed-solomon-simd`)
- The (good case) round complexity is 4 (which is sub-optimal)
  - Nevertheless, it can be adapted to PBFT-like protocols to have an effective round complexity of 2 (with caveats)

    (“Sing a Song of Simplex”, Shoup [2023])
  - In follow-up joint work with Locher, we reduce round complexity to 3 (and sometimes 2, with caveats), but with a modest communication imbalance

    (“Improving the Round Complexity of MiniCast”, Locher & Shoup [2025])