Byzantine-Tolerant Reliable Broadcast in the Presence of Silent Churn

Timothé Albouy, Davide Frey,
Michel Raynal, François Taïani

Univ Rennes, IRISA, CNRS, Inria
Rennes, France

Table of contents

Initial motivation

process disconnection and local state reconciliation
in money transfer applications

Basic Model and Byzantine Reliable Broadcast (BRB)

Basic computing model

- Computing entities:
  - ⋆ n asynchronous sequential processes p_1, ..., p_n
  - ⋆ Up to t processes can be Byzantine
- Communication:
  - ⋆ Fully connected point-to-point network
  - ⋆ Asynchronous
  - ⋆ reliable

Byzantine reliable broadcast (1)

- Introduced with a formal definition:
  Toueg (PODC 1984),
  Bracha and Toueg (JACM 85),
  Bracha (I&C 1987)
- Ensure that
  - ⋆ correct processes: deliver the same set of messages
  - ⋆ This set includes all the messages they rb-broadcast

Byzantine reliable broadcast (2)

Bracha's algorithm (I&C 1987)

- Resilient-optimal: $t < n/3$
- An application message gives rise to
  - $\star$ 3 sequential communication steps
  - $\star$ $(n-1)(2n+1)$ implementation messages

Around Bracha's BRB algorithm

- Versatility

  - Hirt M., Kastrato A., and Liu-Zhang C.-D., Multi-threshold asynchronous reliable broadcast and consensus. *OPODIS'20*, LIPICs Vol. 184, Article 6, 16 pages (2020)
  - Raynal M., On the versatility of Bracha's Byzantine reliable broadcast algorithm. *Parallel Processing Letters*, 31(3), 9 pages (2021)

- Efficiency

  - Imbs D. and Raynal M., Trading t-resilience for efficiency in asynchronous Byzantine reliable broadcast. *Parallel Processing Letters*, Vol. 26(4), 8 pages (2016)

- Scalability

  - Guerraoui G., Kuznetsov P., Monti M., Pavlovic M., and Seredinschi D.-A., Scalable Byzantine reliable broadcast. *DISC'19)*, LIPICs Vol. 146, 16 pages (2019)

- Dynamicity

  - Guerraoui G., Komatovic J., Kuznetsov P., Pignolet P.A., Seredinschi D.-A., and Tonkikh A., Dynamic Byzantine reliable broadcast. *(OPODIS'20)*, LIPICs Vol. 184, 18 pages (2020)

Silent Churn Model and Message Adversary

Santoro-Widmayer's notion of message adversary

- Introduced in the context of synchronous networks:

  - Santoro N. and Widmayer P., Time is not a healer. *(STACS'89)*, Springer LNCS 349, pp. 304-316 (1989)

  -Santoro N. and Widmayer P., Agreement in synchronous networks with ubiquitous faults. *Theoretical Computer Science*, 384(2-3): 232-249 (2007)

- A message adversary is a (constrained) daemon that, at the network level, eliminates messages sent by processes

Message adversary: definition

- To broadcast an (implementation) message at the network level, a (correct) process invokes the macro-operation broadcast msg(v), which is a shorthand for

  "**for all** $i \in \{1, \cdots, n\}$ **do** send msg(v) **to** p_j **end for**"

- For each message msg(v) broadcast by a process, the adversary can eliminate up to d copies of msg(v)
- Reminder: a Byzantine process does not necessarily use this base macro-operation
- $d = 0$: no message adversary

Message adversary: example

- Let D be a set of at most d processes
- During some period of time the adversary suppresses all the messages sent to the processes of D, so that these processes are input-disconnected
- The size and the content of the set D can arbitrarily vary over time
- This kind of churn is $silent$ in the sense that the processes of D do not notify their input-disconnections
- There is no notion of attendance list: no process has information on the status of the other processes

Byzantine failures vs message adversary

- Byzantine failures concern processes (appl. level)
- Message adversary concern messages (network level)

	Byz. processes	Message adv.
Concerned messages	sent by Byz.	all
Msg falsification	yes	no
Message losses with respect to	yes all the processes	yes at most d proc.

Cannot be compared

Silent churn Byzantine-Tolerant Reliable Broadcast(SC-BRB)

Definition

Vocabulary

- Communication abstraction that is built:
  - ⋆ Two operations: scb_broadcast() and scb_deliver()
  - ⋆ Appl. messages: *scb-broadcast/scb-delivered*
- at the network level:
  - ⋆ implementation messages are *broadcast/received*

Safety roperties

- SCB-Validity (no spurious message):
  If a correct process p_i scb-delivers a message m from a correct process p_j with sequence number sn, then p_j scb-broadcast m with sequence number sn

- SCB-No-duplication:
  A correct process p_i scb-delivers at most one message m from a process p_j with sequence number sn

- SCB-No-duplicity:
  If a correct process p_i scb-delivers a message m from a process p_j with sequence number sn, then no correct process scb-delivers another message $m' \neq m$ from p_j with sequence number sn

Liveness properties

- SCB-Local-delivery:
  If a correct process p_i scb-broadcasts a message m with sequence number sn, then at least one correct process p_j eventually scb-delivers m from p_i with sequence number sn
- SCB-Global-delivery:
  If a correct process p_i scb-delivers a message m from a process p_j with sequence number sn, then at most d correct processes do not scb-deliver m from p_j with sequence number sn

  If $d = 0$: boils down to Bracha's specification

An SCB-BRB Algorithm

Signatures

- The algorithm assumes two cryptography-related operations
  - ⋆ sign(msg) creates and signs a digest of the implementation message msg,
  - ⋆ verify(msg, sig, i) returns $\top$ (true) if the signature sig of message msg is valid using the public key of p_i, otherwise it returns $\bot$ (false)
- Signatures are used to cope with the net effect of Byzantine processes and silent churn:
  in spite of the unauthenticated nature of the point-to-point communication channels, signatures allow correct processes to verify the authenticity of messages that have not been directly received from their initial sender, but rather relayed through intermediary processes: they implement a *network-wide* non-repudiation mechanism
- Open problem: eliminate signatures ??

An algorithm

for $n > 3t + 2d$

Extreme: case $d = 0$ and case $t = 0$

Operation scb_broadcast(m)

operation scb_broadcast(m) **is**

$sn_i \leftarrow sn_i + 1$;
$sig_i \leftarrow \mathsf{sign}(\langle m, sn_i, i\rangle)$; % initial sender identity
$sig'_i \leftarrow \mathsf{sign}(\langle m, sn_i, i, sig_i, i\rangle)$; % forwarder identity

$echoes_i \leftarrow echoes_i \cup \{\langle m, sn_i, i, sig_i, i, sig'_i\rangle\}$;
broadcast echo$(m, sn_i, i, sig_i, i, sig'_i)$.

sig_i and sig'_i: signed fixed-size digests

$echoes_i$: set of the five-uplets representing the echoed implementation messages that have been received by p_i

Processing of an echo message

when echo(m, sn, j, sig, k, sig') **is** $received$ **do**
if $(\langle m, sn, j, sig, k, sig' \rangle \notin echoes_i \wedge$ verify$(\langle m, sn, j \rangle, sig, j))$ **then**
  if (verify$(\langle m, sn, j, sig, k \rangle, sig', k)$) **then**
    $echoes_i \leftarrow \{echoes_i \cup \langle m, sn, j, sig, k, sig' \rangle\}$ **end if**;
  if $(\langle m, sn, j, sig, i, - \rangle \notin echoes_i)$ **then**
    $sig'_i \leftarrow$ sign$(\langle m, sn, j, sig, i \rangle)$;
    $echoes_i \leftarrow echoes_i \cup \{\langle m, sn, j, sig, i, sig'_i \rangle\}$;
    broadcast echo$(m, sn, j, sig, i, sig'_i)$ **end if**;
  if $(|\{\langle m, sn, j, sig, -, - \rangle \in echoes_i\}| > \frac{n+t}{2})$ **then**
    $quorum_i \leftarrow \{\langle \ell, sig'' \rangle \mid \langle m, sn, j, sig, \ell, sig'' \rangle \in echoes_i\}$;
    broadcast quorum$(m, sn, j, sig, quorum_i)$ **end if**
end if.

$quorum_i$: set of pairs of signature/signing process id, proving that enough processes witnessed a given application message for it to be delivered

Processing of a quorum message

when quorum$(m, sn, j, sig, quorum)$ **is** $received$ **do**

$valid_i \leftarrow \{\langle k, sig' \rangle \in quorum$
such that verify$(\langle m, sn, j, sig, k \rangle, sig', k)\}$;

if $(|valid_i| > \frac{n+t}{2} \wedge \langle sn, j \rangle \notin delivered_i)$ **then**

broadcast quorum$(m, sn, j, sig, valid_i)$;

$delivered_i \leftarrow delivered_i \cup \{\langle sn, j \rangle\}$;

scb_delivery of m from p_j with seq nb sn

end if.

$valid_i$: set containing pairs of signature/signing process identity for which the verify() operation returned true with respect to the corresponding sender identity

$delivered_i$: set that contains the identifiers (proc. id, seq. number) of the application messages scb-delivered by p_i

Important properties used in the proof

Assumed $n > 3t + 2d$

- $\frac{n-t}{2} > t+d$ (from which follows that at least one echo is from a correct process, used to prove SCB-Validity)
- Any two sets of $> \frac{n+t}{2}$ different processes have at least on correct process in common (from which follows that no application message can be scb-delivered twice)
- $\frac{n+t}{2} < n - t - d$ (used to prove SCB-local delivery)

An important additional result (shown recently)

shown recently

Necessary and sufficient condition

$$\boxed{n > 3t + 2d}$$

Cost of the algorithm

- Cost due to signatures
- The scb-broadcast of an application message by a correct processes entails the sending of $O(n^2)$ implementation messages
- Let $t > 0$.

  For the values of d such that $d < n - t - \sqrt{\frac{n^2-t^2}{2}}$, scb-broadcast terminates in exactly three message rounds

Conclusion

Conclusion

- Model combining three adversaries
  - ⋆ Asynchrony
  - ⋆ Byzantine failures
  - ⋆ Silent churn
- A tight bound: $n > 3t + 2d$
- Open problem: eliminate signatures
- Application:
  local state reconciliation in money transfer