2025 23rd International Symposium on Network Computing and Applications (NCA) | 979-8-3315-7842-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/NCA67271.2025.00030

Rûm: Multivalued Loss-Tolerant Byzantine Consensus for Mobile Ad-Hoc Networks

João Pedro Leite
INESC-ID
Instituto Superior Técnico
Universidade de Lisboa
joao.p.leite@tecnico.ulisboa.pt

Guilherme Ramos
Instituto de Telecomunicações
and Instituto Superior Técnico
Universidade de Lisboa
guilherme.ramos@tecnico.ulisboa.pt

David R. Matos
INESC-ID
Instituto Superior Técnico
Universidade de Lisboa
david.r.matos@tecnico.ulisboa.pt

***Abstract*—We present Rûm, a randomized asynchronous multivalued byzantine consensus algorithm designed for mobile *ad-hoc* networks, operating under the message omission/loss model of Santoro-Widmayer. Rûm is optimal with respect to the byzantine fault bound, and makes progress in rounds where the number of message omissions is bounded, while always ensuring safety. Through network simulations conducted in ns-3, Rûm's performance is evaluated against Turquois and Ezhilchelvan et al.'s multivalued consensus algorithm, demonstrating its ability to achieve consensus in the presence of byzantine faults and message loss. The results indicate that Rûm can efficiently handle message omissions and byzantine faults, maintaining reliable decision-making processes, making it a suitable solution for consensus in mobile *ad-hoc* networks.**

***Index Terms*—Mobile Ad-Hoc Networks, MANET, Byzantine Fault Tolerance, Consensus.**

I. Introduction

Mobile Ad-Hoc Networks (MANET) have become integral to a varied array of applications in both military [16] and civilian settings. Their flexibility, decentralization, dynamism, openness, and ease of deployment allow for interesting new applications, which range from distributed peer-to-peer file-sharing [6], audience participation in musical performances [8], to emergency communication during blackouts and disaster scenarios [3], [12]. These networks, however, come with challenges in guaranteeing adversary-resilient and reliable communication, given the exposed and unreliable nature of the communication medium.

One of the cornerstone algorithms of distributed systems, upon which many other algorithms and applications rely is consensus. The consensus problem was first presented in the work of Lamport et al. [7], and since then, several different algorithms have been proposed [10], [17]. Nonetheless, many of them are unfit for mobile *ad-hoc* networks. These algorithms usually assume known bounds on message delivery times and the existence of reliable bilateral communication links, which are overly demanding assumptions in mobile networks [14], they rely on leader election, or they do not have an adversarial fault-model in mind. However, there are some consensus algorithms explicitly designed for *ad-hoc* networks, such as Turquois [9], and some consensus algorithms suited for *ad-hoc* networks, such as Ben-Or's 1983 consensus algorithm [2], and Ezhilchelvan et al. 2001 multivalued-consensus algorithm [4], whose broadcast based nature makes them suitable for these networks. Of these candidates, however, none features, simultaneously, byzantine fault-tolerance, message loss tolerance, multivalued proposals and asynchronicity.

In this paper, we introduce Rûm, a novel byzantine fault-tolerant multivalued asynchronous consensus algorithm specifically designed for MANETs, by being tolerant to message loss and not assuming any bounds on message delivery times. Rûm can be employed in various settings and applications within mobile *ad-hoc* networks, such as file-sharing, instant messaging, distributed computation, state-replication, database management, and can also serve as the backbone for local peer-to-peer, consistent, communication in a mobile network, without relying on pre-existing infrastructure. Through simulations conducted in the ns-3 network simulator, Rûm demonstrates stable performance across a variety of network sizes and adversarial conditions. The algorithm consistently achieves consensus within a small number of rounds in typical scenarios, even in the presence of byzantine nodes and message omissions. When compared to existing protocols such as Turquois and Ezhilchelvan et al.'s multivalued consensus algorithm, Rûm shows comparable performance with Turquois, despite offering stronger fault tolerance and operating in a more challenging system model, and a better performance than Ezhilchelvan's. These results validate Rûm's practicality and effectiveness as a reliable and efficient foundation for distributed coordination in mobile *ad-hoc* networks.

We make the following key contributions: (i) Rûm, a novel multivalued byzantine consensus algorithm tailored for mobile ad-hoc networks, capable of operating under the message omission model of Santoro and Widmayer [15], by employing a round-based, leaderless, and broadcast-oriented protocol structure that tolerates message loss; and (ii) an evaluation of Rûm through simulations in ns-3, showing its performance compared to existing consensus protocols under byzantine conditions and message loss, and in scenarios with no faults.

Work supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025, UID/PRR/50021/2025, and 2024.07494.IACDC (WELL), and, when eligible, co-funded by EU funds under project/support UID/50008/2025 Instituto de Telecomunicações.

979-8-3315-7842-8/25/$31.00 ©2025 IEEE
DOI 10.1109/NCA67271.2025.00030

II. System Model and Assumptions

In this section, we present Rûm's system model, state our assumptions, formally specify our problem definition, and which properties a candidate solution needs to satisfy.

A. Problem Definition

Rûm solves the multivalued byzantine consensus problem in an asynchronous network. In this problem, a set of n processes wishes to decide on a common value from a set of proposed values, executing an agreed-upon algorithm to do so. In contrast, some of those processes can be faulty and are allowed to arbitrarily, and even maliciously, deviate from the algorithm. A process that correctly follows the algorithm is termed a correct process, otherwise it is called a *byzantine* process.

Formally, an algorithm that solves the multivalued byzantine consensus problem must satisfy the following properties ($\mathcal{P}1$) to ($\mathcal{P}3$):

($\mathcal{P}1$) **Validity**: If all correct processes propose the same value $\mathcal{V}$, then any correct process that decides, decides $\mathcal{V}$.

($\mathcal{P}2$) **Agreement**: No two correct processes decide differing values.

($\mathcal{P}3$) **(Probabilistic) Termination**: Each correct process eventually decides a value, with probability 1.

Property ($\mathcal{P}3$) requires only termination with probability 1 to account for non-deterministic algorithms[1].

1) Terminology and Notation: In this paper, we adopt the following notation: we denote the number of processes in the system as n, the maximum number of faulty processes as f, and the process with identifier i as p_i. The phase of a process p_i is denoted as ϕ_i, its estimated value as v_i, and any other internal state variable of a process p_i will have the process's identifier in subscript.

A quorum of processes is any set of processes of size greater than $\frac{n+f}{2}$. Any two quorums intersect in at least one correct process, and in a quorum there is a majority of correct processes. In this paper, we call a majority that which is most frequent in a multiset.

We denote messages and their contents between angle brackets, such as $\langle i, \phi_i, v_i, status_i \rangle$. Some messages may carry a tag to identify the type of message being received, such as $\langle DEC, *, * \rangle$, which, in this case, identifies the message as a decision message. Whenever a message contains an asterisk ($*$) we mean that any value may be substituted for the asterisk.

B. System Model and Assumptions

We model the system as a set of n nodes, of which at most $f < \frac{n}{3}$ of those nodes can arbitrarily deviate from a correct execution of the algorithm. As is fit for *ad-hoc* mobile networks, the underlying communication in the network is assumed to be asynchronous – without bounds on the delivery of messages – and an arbitrary amount of message omissions are allowed, with safety, but not termination, being assured. To ensure termination ($\mathcal{P}3$), the number of message omission faults must be bounded, and, with a particular bound of up to $\sigma < \lfloor \frac{n+f}{2} \rfloor \left(n - t - \lfloor \frac{n+f}{2} \rfloor \right)$ messages lost per round, where $t \leq f$ is the actual number of byzantine processes, termination is guaranteed. We note that, even if the omissions in a round are not bounded, the safety properties ($\mathcal{P}1$) and ($\mathcal{P}2$) are always guaranteed. We also impose fairness on the omission of decision messages – messages of the form $\langle DEC, *, * \rangle$ – so that decision messages are eventually received by all correct processes, if broadcast infinitely often.

Byzantine processes, or adversaries, are assumed to have access to the internal state of every process, and are allowed to drop, replay, alter, and delay any messages.

We assume that processes have access to a message authentication scheme, be it via the existence of a public-key infrastructure or some symmetrical cryptographic scheme, and that cryptographic primitives cannot be broken by an adversary. We make the additional assumption that all processes are able to randomly choose an element from a finite set.

III. Proposed solution: Rûm

Rûm is an asynchronous multivalued byzantine consensus algorithm, i.e., it allows a set of processes to agree on a value among those proposed by each process, even when some of them can arbitrarily deviate from the algorithm's correct execution and there are no bounds on how long it takes for messages to be delivered. Rûm uses randomness to sidestep the FLP impossibility result [5], by randomizing the proposal values of processes, allowing for the system, at any time, to randomly select a configuration favorable for termination. In this section, we detail Rûm's design. Rûm was designed with a few key goals in mind:

1) **Peer-to-peer**: all nodes should behave equally, without a leader;
2) **Broadcast-based**: all communication should occur via broadcasting messages;
3) **Asynchronicity**: the algorithm should not assume any bound on message delivery.
4) **Loss-tolerance**: the algorithm should tolerate message loss.
5) **Locality**: the algorithm's state transitions should only rely on received messages and local randomness.

To reach these goals, we build Rûm as a randomized round-based broadcast consensus algorithm, which is leaderless, omission-tolerant, and asynchronous, adding semantic and authenticity validation procedures in order to curb the power of byzantine processes.

A. Algorithm Description

A process p_i correctly executing Rûm periodically broadcasts messages of the form $\langle i, \phi_i, v_i, status_i \rangle$ and upon state transitions. Each round in Rûm consists of three sequential phases, which are called CONVERGE, LOCK, and DECIDE (see Fig. 1 for a flowchart of the state transitions between phases, and Algorithm 1 for Rûm's pseudocode).

[1]For a probabilistic algorithm to terminate with probability 1, the probability of termination must converge to 1 as the algorithm progresses.

In the CONVERGE phase (Line 26), processes broadcast their decision estimates and select as their new estimates the majority value from the messages they received. This phase not only guarantees ($\mathcal{P}1$) but also allows Rûm to terminate earlier in specific cases [11].

In the LOCK phase (Line 28), processes broadcast their estimates again, hoping to find a quorum – a set of more than $\frac{n+f}{2}$ messages – with the same value. If no such value is found, then processes select a special value $\perp$ that signals a lack of preference, or failure to acquire an estimate. After the LOCK phase, every process sets $\perp$ as its estimate or the same value v, a property which we denote as "quasi-agreement".

In the DECIDE phase (Line 33), processes broadcast their estimates to determine if a quorum of processes locked onto the same value. If such a value was locked in the previous phase, then it is decided either in that round's DECIDE phase or is carried over to the next round. If no value was locked, then processes randomly select a new estimate and begin the next round.

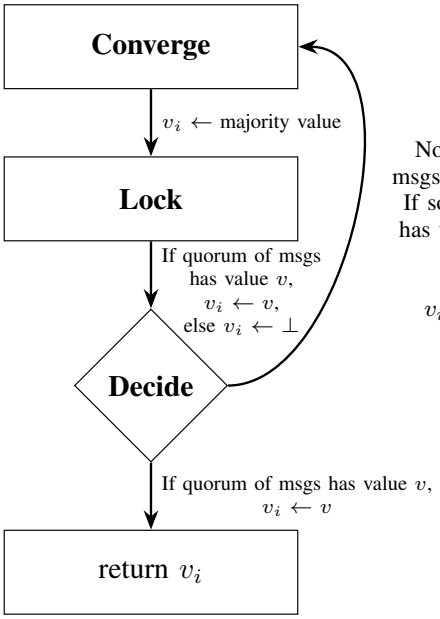


Fig. 1: Flowchart depiction of Rûm's pseudocode.

For ease of comprehension, we provide a message delivery diagram of two consecutive rounds of Rûm in Fig. 2. In the first round (Fig. 2a), correct processes propose differing values, and an adversary schedules the delivery of messages so that the correct processes are prevented from deciding in that round. However, some correct processes carry their proposal to the DECIDE phase, and onto the next round. The remaining correct processes will have to randomly select their new proposal values among values in messages received in the LOCK phase. In Fig. 2b, the remaining process randomly selects its proposal value in accordance with the remaining correct processes, resulting in an unanimous proposal round, where the correct processes can always decide despite byzantine interference. For a complete description of Rûm, we present Rûm's pseudocode in Algorithm 1.

Algorithm 1: Rûm: Multivalued Byzantine Consensus

Input: Initial arbitrary proposal value $proposal_i$
Output: Decision value proposed by some process

1 $\phi_i \leftarrow 1$; $v_i \leftarrow proposal_i$; $V_i \leftarrow \emptyset$
2 $status_i \leftarrow undecided$; $terminated_i \leftarrow false$
3 **when** $m = \langle DEC, v, \mathcal{M} \rangle$ *is received* **do** // TERMINATION
4 **if** m *is valid* **then**
5 $v_i \leftarrow v$
6 $status_i \leftarrow decided$
7 $terminated \leftarrow true$
8 **when** *local clock tick* $\wedge$ not $terminated_i$ **do**
9 broadcast $(\langle i, \phi_i, v_i, status_i \rangle)$
10 **when** $m = \langle j, \phi_j, v_j, status_j \rangle$ *is received* **do**
11 **if** not $terminated_i$ **then**
12 $V_i \leftarrow V_i \cup \{m : m \text{ is valid}\}$
 // TERMINATION
13 **if** $\exists v : V = \{\langle *, *, v, decided \rangle \in V_i\} \wedge |V| > f$ **then**
14 $terminated \leftarrow true$
15 **if** $terminated_i$ **then**
16 $V \leftarrow \{\langle *, *, v_i, decided \rangle \in V_i\}$
17 broadcast $(\langle DEC, v_i, V \rangle)$
18 **if** $\exists \langle *, \phi, v, status \rangle \in V_i : \phi > \phi_i$ **then** // FAST-FORWARD
19 $\phi_i \leftarrow \phi$;
20 **if** $\phi \equiv 1 \pmod 3$ ***and*** $|\{\langle *, \phi - 1, \perp, * \rangle \in V_i\}| > \frac{n+f}{2}$ **then**
21 $v_i \leftarrow$ coin $(\{v : \langle *, \phi - 2, v, * \rangle \in V_i\})$
22 **else**
23 $v_i \leftarrow v$
24 $status_i \leftarrow status$;
25 **if** $|\langle *, \phi_i, *, * \rangle \in V_i| > \frac{n+f}{2}$ **then**
26 **if** $\phi_i \equiv 1 \pmod 3$ **then** // CONVERGE
27 $v_i \leftarrow$ majority value v in messages with phase $\phi = \phi_i$
28 **if** $\phi_i \equiv 2 \pmod 3$ **then** // LOCK
29 **if** $\exists v : |\{\langle *, \phi_i, v, * \rangle \in V_i|\} > \frac{n+f}{2}$ **then**
30 $v_i \leftarrow v$
31 **else**
32 $v_i \leftarrow \perp$
33 **if** $\phi_i \equiv 0 \pmod 3$ **then** // DECIDE
34 **if** $\exists v : |\{\langle *, \phi_i, v, * \rangle \in V_i|\} > \frac{n+f}{2}$ **then**
35 $status_i \leftarrow decided$
36 **if** $\exists v : |\{\langle *, \phi_i, v, * \rangle \in V_i|\} \geq 1$ **then**
37 $v_i \leftarrow v$
38 **else**
39 $v_i \leftarrow$ coin $(\{v : \langle *, \phi_i - 1, v, * \rangle \in V_i\})$
40 $\phi_i \leftarrow \phi_i + 1$
41 **when** $status_i = decided$ **do**
42 $decision_i \leftarrow v_i$

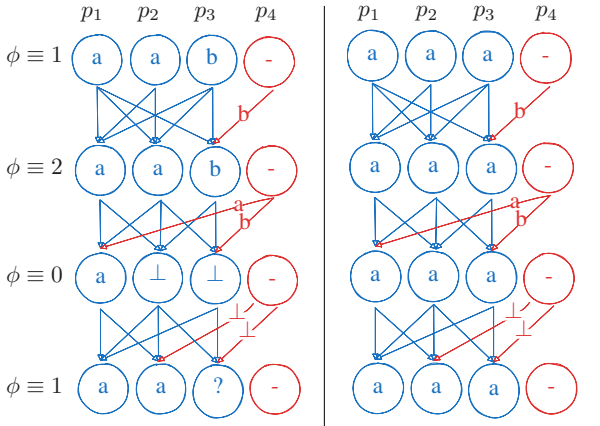
Theorem 1. *Rûm satisfies the consensus properties, ($\mathcal{P}1$) to ($\mathcal{P}3$), if the number of message omissions is bounded by* $\sigma < \lfloor \frac{n+f}{2} \rfloor \left(n - t - \lfloor \frac{n+f}{2} \rfloor \right)$.

Proof Sketch. Rûm satisfies validity ($\mathcal{P}1$), since if all correct processes propose the same value $\mathcal{V}$, then every quorum of messages has majority $\mathcal{V}$, so any estimates in the LOCK phase must also be $\mathcal{V}$, and so in the DECIDE phase, any process that decides must decide $\mathcal{V}$.

Agreement ($\mathcal{P}2$) is ensured by the quasi-agreement property

of the LOCK phase. In the DECIDE phase, after the LOCK phase, processes either estimate the same value $\mathcal{V}$ or $\perp$. If a process decides, in that DECIDE phase, it must decide $\mathcal{V}$, and so must any other process that decides in the same DECIDE phase. Once a process has decided, any other process that continues to the next round must decide $\mathcal{V}$, due to the fact that more than $\frac{n-f}{2}$ correct processes begin the next round with estimate $\mathcal{V}$.

Termination ($\mathcal{P}3$) is guaranteed if correct processes progressively increase their phase, and if they can all randomly select the same value from the DECIDE phase to the CONVERGE phase. Due to quorum intersections, in the LOCK phase, every correct processes receives a message common to every other correct process, and so, every correct process can select that common value on their coin. Eventually, by randomness, there will be some round where all correct processes will select that message's common value on their coin toss, and will then decide in the following round. Processes can always attempt another coin toss if they can always increase their phase. Processes can always increase their phase if a quorum of correct processes can always communicate, which is guaranteed if the number of message omissions is $\sigma < \lfloor \frac{n+f}{2} \rfloor \left(n - t - \lfloor \frac{n+f}{2} \rfloor \right)$, since, to partition the correct processes such that no quorum is formed, more message omissions are required. Therefore, processes can always increase their phase and eventually decide. □



(a) There is no decision, some values carry to the next round and one is randomized.

(b) Unanimous values lead to an immediate decision of the initial value a.

Fig. 2: Message Delivery diagram of executions of Rûm as a function of the initial values of correct processes, with byzantine interference. Correct processes are in blue, and byzantine processes in red. Randomized values are marked with a question mark (?).

B. Message Validation

To prevent byzantine processes from interfering with a correct execution of Rûm, received messages must be validated. There are two kinds of message validation: authenticity and integrity validation, and semantic validation. Authenticity and integrity validation is straightforward, with correct processes having access to cryptographic primitives and infrastructure that allows them to check if a message was created by a given process and has not been altered in transit. Semantic validation, on the other hand, serves to restrict the damage that byzantine processes can inflict, with correct processes validating incoming messages to guarantee that they do not conflict with a correct protocol execution or any previous valid message. For example, a byzantine process p_b might attempt to broadcast a message of the form $\langle b, 1, v_b, decided \rangle$ to mislead correct processes into thinking that p_b's proposal value should be decided, even if they should not. A correct process that receives this message will perform semantic validation and check that it is impossible for any process correctly executing the algorithm to have decided in the first phase, and therefore ignore the message. We present the semantic validation rules in Algorithm 2.

Algorithm 2: Rûm: Semantic Validation (process p_i)

Input: Message m.
Output: Boolean value indicating the validity of the input message.

if $m = \langle j, \phi_j, v_j, status_j \rangle$ **then** // Round Message
  // PHASE
  if not $\left(\phi_j \neq 1 \text{ and } |\langle *, \phi_j - 1, *, * \rangle \in V_i| > \frac{n+f}{2} \right)$ **then**
    return false
  if $\phi_j \equiv 2 \pmod 3$ **then** // VALUE
    if not $\exists \mathcal{M} \subset V_i^{\phi_j - 1}$ *such that* v_j *is a majority for* $\mathcal{M}$, *with* $|\mathcal{M}| \geq \frac{n+f}{2}$ **then**
      return false
  if $\phi_j \equiv 0 \pmod 3$ **then**
    if $v_j = \perp$ **then**
      if not $\exists \mathcal{M} \subset V_i^{\phi_j - 1}$ *with two different valued messages in* $\mathcal{M}$ **then**
        return false
    if $v_j \neq \perp$ **then**
      if not $|\{\langle *, \phi_j - 1, v_j, * \rangle \in V_i\}| > \frac{n+f}{2}$ **then**
        return false
  if $\phi_j \equiv 1 \pmod 3$ **and** $\phi_j > 1$ **then**
    if not $|\{\langle *, \phi_j - 2, v_j, * \rangle \in V_i\}| > \frac{n+f}{2}$ **and not** $|\{\langle *, \phi_j - 1, \perp, * \rangle \in V_i\}| > \frac{n+f}{2}$ **then**
      return false
  if $status = decided$ **then** // STATUS
    if $\phi_j \leq 3$ **then return** false
    if not $|\{\langle *, \phi, v_j, * \rangle \in V_i : \textit{highest received } \phi \textit{ such that } \phi \equiv 0 \pmod 3 \}| > \frac{n+f}{2}$ **then**
      return false
if $m = \langle DEC, v, \mathcal{M} \rangle$ **then** // Decision Message
  $\mathcal{M}' \leftarrow \{m' = \langle *, *, v_j, decided \rangle \in \mathcal{M} : m' \text{ is valid } \}$
  if not $|\mathcal{M}'| > f$ **then**
    return false
return true

Semantic validation is a powerful mechanism to restrict byzantine processes, but it does have its drawbacks. Each message requires validation, which introduces overhead. Perhaps more costly is the possible need for processes to broadcast justification messages due to asynchronicity and message

omissions. For a correct process p_i to consider a message $m = \langle *, \phi_j, *, * \rangle$ as valid, it must have received a quorum of messages with phase $\phi_j - 1$, except for $\phi_j = 1$, which serves as a justification for the phase of m. If, due to asynchronicity and message delays, p_i has not received such a quorum before receiving m, it must reject m. Therefore, p_j might have to supply p_i with the messages that led to p_j's state transition from phase $\phi_j - 1$ to phase ϕ_j, as justifications for its current message. This introduces some additional overhead to the protocol. Notwithstanding, justifications are not always required, if the messages needed to justify m have already been received. The justifications of m might themselves require justification, and so several justification messages might need to be sent, in a cascading effect. It is thus fitting for p_j to be optimistic, and to only send justifications for a message m if p_i rejects m, to minimize the number of justifications being sent, in a typical execution.

C. Broadcast Termination

For processes to stop broadcasting their internal state and transitioning to subsequent phases, we introduce a mechanism for correct processes to eventually stop broadcasting. Upon knowing that a quorum of nodes have decided, a correct process will "terminate", and stop broadcasting messages in rounds (Line 3). Instead, they will broadcast a decision message, which contains proof that a correct process has decided a given value, and for correctness, these decision messages must eventually be received by all correct processes. Upon receiving a decision message for a value v, a correct process will decide v and terminate, also broadcasting a decision message for v, or just echoing the one it received. In our implementation of Rûm, the proof contained in a decision message for a value v is a set of at least $f + 1$ messages with value v and a decided status.

IV. Evaluation

In this section, we go over our implementation and evaluation of Rûm in the ns-3 network simulator [13], and we compare it with our implementation of Turquois [9] and Ezhilchelvan et al.'s algorithm [4], which was implemented similarly and evaluated in ns-3, for fairness. The ns-3 network simulator allowed us to evaluate these algorithms with large numbers of processes – up to $n = 300$ – and with message loss as would be expected in a real network. We note that Ezhilchelvan et al.'s algorithm was inefficient compared to Rûm and Turquois, given that even in the smallest test case – with $n = 7$ processes – it would fail to terminate in under 60 minutes. The remaining algorithms used a timeout of 60 seconds. Therefore, we present only the results obtained in testing Rûm and Turquois.

A. Implementation Details

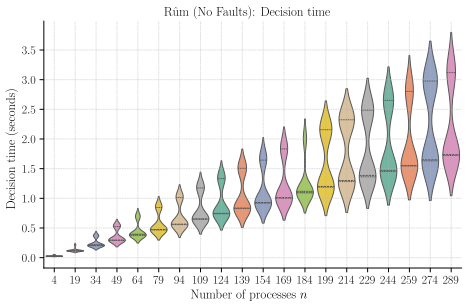
In every simulation of Rûm, the local clock period was set to $15n$ milliseconds, where n is the number of processes executing the Rûm consensus, and every broadcast is delayed by a random time below $1.1n$ milliseconds, which we refer to as the *broadcast jitter*. The broadcast jitter serves to prevent network congestion and packet collisions by staggering messages randomly, and the clock timer is adjusted linearly to allow for all messages sent by processes to arrive within a single clock period. We also apply linear clock periods and broadcast jitters in the simulation for Turquois, where they are, respectively, $80n$ milliseconds and n milliseconds. The authors chose these values since they provided better performance for each algorithm during the simulations. Every process in the simulation randomly chooses its proposal value.

For Rûm, each process chooses a random alphanumeric string of length 32, which makes it unlikely that any two processes have the same proposal. In this way, Rûm is tested in its worst-case configuration, in which all processes propose distinct values, since equal proposals accelerate termination in the CONVERGE phase. For Turquois, each process randomly selects either 0 or 1 as its proposal, which results in many processes (about half) having equal proposal values, which gives Turquois an advantage to terminate sooner, compared to Rûm. In the simulations, nodes are arranged in a two meter radius circle and are equipped with a transmission rate of 11 Mbps. All communication is performed via UDP broadcast to the networks broadcast address. We also introduce in our simulations a "simulated message loss" parameter, which is the probability that a packet is dropped on arrival, after being correctly received by a process. We did this to introduce some message loss beyond the network simulation. We calculate the effective message loss of a process p_i as $\frac{|\mathcal{B}| \times n - |\mathcal{R}_i|}{|\mathcal{R}_i|}$, where $\mathcal{B}$ is the set of all messages broadcast by processes, and $\mathcal{R}_i$ is the set of all messages that p_i received. which is always greater than the simulated message loss, and takes into account messages lost at the transport, network and physical layers. All signatures are EC25519 signatures.

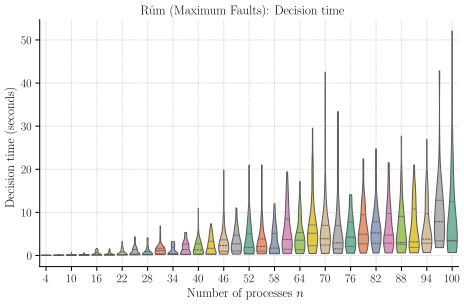
B. Experimental Results

In Fig. 3a, we present several violin plots that estimate the distribution of the decision time of Rûm with no byzantine faults, as a function of the number of participating processes. It is noteworthy that the estimated distributions appear to be bimodal, across process groups. This happens in instances in which sufficient messages are lost, due to message collisions and network congestion, resulting in processes having to trigger a clock tick after timing out. Interestingly, processes seem to only timeout at most once when no faults are present, with most executions finishing in one to three rounds. The decision time also seems to grow linearly in the number of processes, which is to be expected, since, in each round, 3 broadcasts are sent and $3n$ broadcasts are received, which results in $\mathcal{O}(n)$ processing per round.

1) Byzantine Faults: In our simulation, we modeled byzantine processes as processes that send arbitrary random values, distinct from the proposals of correct processes, in their broadcast messages, and deployed the maximum amount of byzantine processes, which is the fault threshold. This is a simple model of byzantine behaviour, which can't capture adversaries which collude and specifically target correct processes based

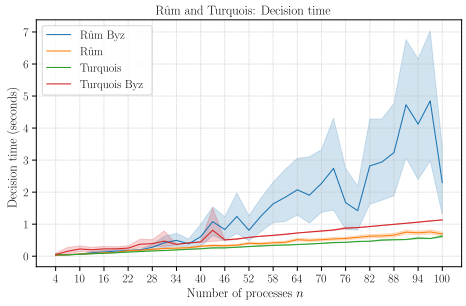


(a) No processes are faulty.

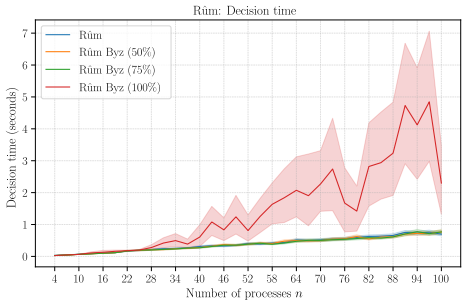


(b) Of the n processes, $f = \lfloor \frac{n-1}{3} \rfloor$ are faulty.

Fig. 3: Violin plot of the average decision time of processes as a function of the number of processes n, in varying fault scenarios.



(a) Comparison between Rûm and Turquois in scenarios with no faults and with maximum faults ($b = f = \lfloor \frac{n-1}{3} \rfloor$).



(b) Comparison between Rûm with no faults and Rûm with $b = f$, $b = 0.75\% \times f$, and $b = 0.50\% \times f$ faults, rounded down.

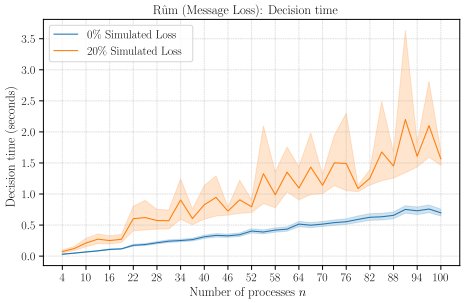
Fig. 4: Line plot of the average decision time of processes as a function of the number of processes n, in varying fault scenarios.

TABLE I: Average Decision Time (in ms) $\pm$ 95% confidence intervals, for Rûm and Turquois, with and without byzantine processes.

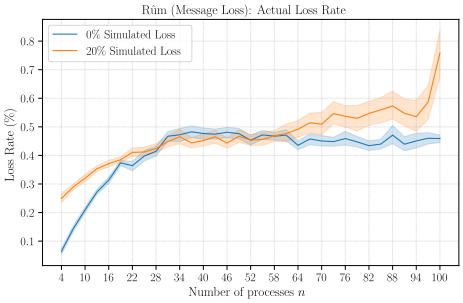
	Average Decision Time (ms)					
n	Turquois	Turquois Byz (100%)	Rûm	Rûm Byz (50%)	Rûm Byz (75%)	Rûm Byz (100%)
4	48.2 ± 35.0	51.3 ± 34.7	30.3 ± 2.4	29.0 ± 2.7	26.7 ± 1.9	26.4 ± 1.7
7	41.4 ± 1.7	154.9 ± 97.2	48.2 ± 3.5	43.3 ± 1.7	44.3 ± 2.0	45.7 ± 2.2
10	60.0 ± 3.7	225.2 ± 92.3	66.9 ± 4.5	65.0 ± 3.6	63.5 ± 3.0	63.3 ± 0.8
19	108.3 ± 1.4	230.3 ± 58.6	116.8 ± 4.1	116.4 ± 1.1	116.2 ± 1.2	160.0 ± 36.8
28	165.7 ± 7.9	391.6 ± 115.7	214.6 ± 18.5	203.1 ± 17.2	208.3 ± 17.1	278.2 ± 78.5
37	215.5 ± 7.6	415.1 ± 8.6	265.4 ± 22.2	260.7 ± 19.9	253.6 ± 18.2	385.9 ± 140.3
46	261.0 ± 2.4	509.3 ± 13.1	326.8 ± 26.1	351.9 ± 30.3	367.9 ± 31.8	832.4 ± 369.9
55	319.0 ± 11.0	624.2 ± 3.3	388.4 ± 32.0	412.8 ± 34.6	414.8 ± 35.6	1253.4 ± 558.2
64	364.6 ± 2.2	723.9 ± 3.1	515.7 ± 43.5	473.1 ± 40.5	494.4 ± 42.7	2072.9 ± 905.0
73	424.4 ± 15.0	817.9 ± 14.9	539.5 ± 44.6	522.4 ± 41.5	541.8 ± 45.6	2740.5 ± 1525.4
82	466.9 ± 2.6	926.5 ± 2.7	625.4 ± 53.9	586.4 ± 46.5	546.1 ± 37.7	2818.7 ± 1382.7
91	520.0 ± 2.5	1028.3 ± 3.4	751.9 ± 61.7	703.6 ± 60.8	700.1 ± 59.8	4728.8 ± 1851.0
100	620.9 ± 43.5	1132.1 ± 3.2	697.5 ± 55.4	768.9 ± 65.6	765.5 ± 65.2	2300.2 ± 1080.6

on global knowledge of their internal state. However, it is effective in demonstrating the semantic validation rules, and how invalid messages make the system more vulnerable to message loss.

In Fig. 3b, we can see that the introduction of faults has a noticeable effect, with Rûm being, at times, around 40 times slower than in the no-fault scenario of Fig. 3a. We attribute this slower decision time to correct processes not receiving a quorum of messages in a single clock period. This happens due to message loss, which is more likely given correct processes will need to receive every message from correct processes, since byzantine messages will be declared invalid, and only



(a) Average decision time of Rûm processes with a simulated loss rate of 0% and 20%.



(b) Effective loss rate with a simulated loss of 0% and 20%. The effective loss is grater than the simulated loss.

Fig. 5: Line plot of the average decision time and message loss rate of processes as a function of the number of processes n and of the induced simulated loss rate.

correct processes can generate a quorum of valid messages. The impact of byzantine faults, in typical network conditions, can be reduced by decreasing the clock period and performing congestion control.

We also ran Turquois in the same conditions as Rûm, with the timing parameters discussed in Section IV-A, to compare the two algorithms, with and without byzantine faults. We present our findings in Fig. 4a as a line plot, and in Table I in tabular form. Rûm's decision latency, in the fully faulty case, is significantly slower than the Turquois fully faulty scenario. This was expected, given that Turquois handles only binary input values, which makes convergence more likely, in the CONVERGE phase, even with the presence of byzantine faults. In fact, the latency gap between the byzantine and the non-byzantine executions of Turquois is mainly attributed to Turquois needing, on average, to receive more messages to reach a quorum of valid messages, since some (around half) of the messages from byzantine processes will be invalid. We highlight that Rûm achieves comparable performance to Turquois, in the scenario with no byzantine faults.

We tested Rûm with varying amounts of faulty processes, below the fault threshold. We found that, when the number of byzantine processes does not exactly match the fault threshold – i.e. the case with maximum faults – Rûm has almost equal latency as in the case with no faults. We tested Rûm with the additional cases of the number of byzantine processes b being equal to 50% and 75% of the fault threshold, rounded down, and present our findings in Fig. 4b and Table I. We conclude that, up to around 80% of the fault threshold, Rûm has performance as good as with no faults.

2) Message Loss: To test Rûm's resilience to message loss, besides byzantine fault tolerance, we introduced some simulated message loss into the simulations and measured how the decision latency would be affected by an increase in message loss. Rûm was designed to tolerate message loss, and, as we discussed in the proof of Theorem 1, can progress when the message losses are bounded in a broadcast round. We present our findings in Fig. 5. We highlight that the actual loss rate of messages increases with the simulated loss rate, beyond that which would be expected from the simulated rate, due to messages lost due to congestion. Lost messages lead to clock timeouts, which lead to more messages broadcast, along with their justifications, which leads to more message loss. However, we see that Rûm still functions under loss rates of 50%-80%, even though decision latency is affected.

V. Related Work

There is some work related to Rûm, such as the Turquois [9], Ben-Or's [2], and Ezhilchelvan et al.'s [4] algorithms. These algorithms are either binary consensus algorithms or not byzantine fault tolerant. Rûm differs from these algorithms by implementing simultaneously multivalued consensus, byzantine fault tolerance, and message loss tolerance.

Ben-Or's (1983) binary consensus algorithm [2] circumvented the FLP impossibility result [5] in an asynchronous system where processes may fail, by employing non-determinism. Ben-Or's algorithm works in rounds, where processes attempt to converge on a majority of proposals to decide said majority. If processes cannot converge on a value, then, in the following round, the processes randomly choose a new binary proposal value. This algorithm is conceptually simple. Nonetheless, it took 15 years for its correctness to be proven by Aguilera and Toueg [1]. Ben-Or's algorithm lacks performance, having an exponential complexity – $\mathcal{O}(2^n)$, where n is the number of processes – in the expected number of rounds until termination, and its system model admits only crash-stop faults. While theoretically interesting, due to operating asynchronously, Ben-Or's algorithm is unusable in a practical setting, due to its poor performance.

Ezhilchelvan et al. (2001) proposed a Randomized Multivalued Consensus Protocol [4], which functioned very similarly to Ben-Or's, but that extended it to the multivalued case. The algorithm introduced an initial broadcast phase, whereupon processes broadcast their initial proposal, and then store the received proposals in a local array. Whenever a new proposal value must be randomly selected, processes rely on

their local array of proposal values and randomly choose from this array a new proposal. In all other respects, Ezhilchelvan et al.'s algorithm functioned like Ben-Or's, but without strictly broadcasting binary values, and can be seen as a generalization of it. Like Ben-Or's algorithm, this algorithm operates in the crash-stop fault model, where processes can only fail by crashing mid-execution.

Turquois (2010) [9] is a byzantine fault-tolerant binary k-consensus algorithm, designed particularly for *ad-hoc* networks. Turquois operates in the message omission fault model of Santoro and Widmayer, like Rûm, and solves the k-Consensus problem in asynchronous networks, via non-determinism. This algorithm can be viewed as an extension of Ben-Or's algorithm to the byzantine fault model, inheriting much of the same structure. When viewed this way, Turquois presents a few improvements over Ben-Or's algorithm: (i) a new phase (CONVERGE) designed to expedite consensus [11]; (ii) an efficient hash-based symmetrical message authentication and integrity scheme; (iii) a semantic validation procedure to curb byzantine processes' ability to subvert the system.

Turquois is a flexible consensus algorithm, in the sense that it can tolerate a higher number of message omissions by reducing the required number of deciding processes. Formally, Turquois guarantees k-consensus as long as, per round, at most $\sigma < \lceil \frac{n-t}{2} \rceil (n-k-t) + k - 2$ messages are omitted, where n is the number of processes partaking in consensus, k is the required number of correct deciding processes, and t is the actual number of faulty processes, with $t \leq f$.

VI. Conclusion

In this paper, we presented Rûm, a novel asynchronous multivalued byzantine consensus algorithm operating in the message omission fault model of Santoro and Widmayer, and is specifically designed for use in wireless *ad-hoc* networks. This algorithm, via randomization, circumvents both FLP and Santoro and Widmayer's impossibility results for asynchronous networks, as long as message omissions are bounded by $\sigma < \lfloor \frac{n+f}{2} \rfloor \left(n - t - \lfloor \frac{n+f}{2} \rfloor \right)$, where n is the number of processes and f is the fault threshold. We simulated Rûm in ns-3 and established that it's performance scales linearly in the number of processes, with up to $n = 100$ nodes maintaining sub-second decision latency. Decision latency is affected by an increase in byzantine processes and message loss, but Rûm, nonetheless still terminated, during testing.

There is interesting further work to improving Rûm. Since Rûm's performance is mostly dictated by the parameters for the clock timeout and how Rûm handles message congestion, some form of congestion detection control and adaptive timeouts would be beneficial for practical implementations of the algorithm. By decreasing the clock timeout, correct processes retransmit lost messages faster, but they do so at the risk of introducing more congestion in the network, which can increase the number of lost messages. Error-correcting codes can also be applied to reduce the loss rate. Rûm is most useful for applications in mobile *ad-hoc* networks, where messages can be lost and agreement is required amongst devices, such as in messaging apps, information boards and file sharing applications. The fault model in our simulations is quite simple, and cannot capture the full spectrum of attacks allowed by our system model. Further simulations, with more intricate adversarial behaviour make for interesting and important future work.

References

[1] M. K. Aguilera and S. Toueg, "The correctness proof of ben-ors randomized consensus algorithm," *Distributed Computing*, vol. 25, no. 5, pp. 371–381, 2012.

[2] M. Ben-Or, "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '83. New York, NY, USA: Association for Computing Machinery, 1983, p. 2730.

[3] V. Dattana, A. Kush, R. Hasan, and S. Mahmood, "Ad hoc network as a solution in disaster management," in *2020 Global Conference on Wireless and Optical Technologies (GCWOT)*. IEEE, 2020, pp. 1–4.

[4] P. Ezhilchelvan, A. Mostefaoui, and M. Raynal, "Randomized multivalued consensus," in *Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISORC 2001*. IEEE, 2001, pp. 195–200.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[6] A. Klemm, C. Lindemann, and O. P. Waldhorst, "A special-purpose peer-to-peer file sharing system for mobile ad hoc networks," in *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No. 03CH37484)*, vol. 4. IEEE, 2003, pp. 2758–2763.

[7] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the works of Leslie Lamport*, 2019, pp. 203–226.

[8] S. W. Lee, G. Essl, and Z. M. Mao, "Distributing mobile music applications for audience participation using mobile ad-hoc network (manet)," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2014, pp. 533–536.

[9] H. Moniz, N. F. Neves, and M. Correia, "Turquois: Byzantine consensus in wireless ad hoc networks," in *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2010, pp. 537–546.

[10] G.-T. Nguyen and K. Kim, "A survey about consensus algorithms used in blockchain," *Journal of Information processing systems*, vol. 14, no. 1, pp. 101–128, 2018.

[11] M. Raynal, "A simple predicate to expedite the termination of a randomized consensus algorithm," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, pp. 106–111.

[12] D. Reina, J. M. L. Coca, M. Askalani, S. L. Toral, F. Barrero, E. Asimakopoulou, S. Sotiriadis, and N. Bessis, "A survey on ad hoc networks for disaster scenarios," in *2014 IEEE international conference on intelligent networking and collaborative systems*, 2014, pp. 433–438.

[13] G. Riley and T. Henderson, *The ns-3 Network Simulator*, 01 2010, pp. 15–34.

[14] D. Sakavalas and L. Tseng, "Delivery delay and mobile faults," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–8.

[15] N. Santoro and P. Widmayer, "Time is not a healer: Preliminary version," in *STACS 89: 6th Annual Symposium on Theoretical Aspects of Computer Science Paderborn, FRG, February 16–18, 1989 Proceedings 6*. Springer, 1989, pp. 304–313.

[16] R. Sivakami and G. K. Nawaz, "Secured communication for manets in military," in *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*. IEEE, 2011, pp. 146–151.

[17] W. Zhong, C. Yang, W. Liang, J. Cai, L. Chen, J. Liao, and N. Xiong, "Byzantine fault-tolerant consensus algorithms: A survey," *Electronics*, vol. 12, no. 18, p. 3801, 2023.