SOLUTION: BYZANTINE CLUSTER-SENDING IN EXPECTED CONSTANT COST AND CONSTANT TIME

Anonymous authors
Paper under double-blind review

Abstract

Traditional resilient systems operate on fully-replicated fault-tolerant clusters, which limits their scalability and performance. One way to make the step towards resilient high-performance systems that can deal with huge workloads is by enabling independent fault-tolerant clusters to efficiently communicate and cooperate with each other, as this also enables the usage of high-performance techniques such as sharding. Recently, such inter-cluster communication was formalized as the *Byzantine cluster-sending problem*. Unfortunately, existing worst-case optimal protocols for cluster-sending all have *linear complexity* in the size of the clusters involved.

In this paper, we propose *probabilistic cluster-sending techniques* as a solution for the cluster-sending problem with only an *expected constant message complexity*, this independent of the size of the clusters involved and this even in the presence of highly unreliable communication. Depending on the robustness of the clusters involved, our techniques require only *two-to-four* message round-trips (without communication failures). Furthermore, our protocols can support worst-case linear communication between clusters. Finally, we have put our techniques to the test in an in-depth experimental evaluation that further underlines the exceptional low expected costs of our techniques in comparison with other protocols. As such, our work provides a strong foundation for the further development of resilient high-performance systems.

1 Introduction

The promises of *resilient data processing*, as provided by private and public blockchains [14, 21, 27], has renewed interest in traditional consensus-based Byzantine fault-tolerant resilient systems [5, 6, 24]. Unfortunately, blockchains and other consensus-based systems typically rely on fully-replicated designs, which limits their scalability and performance. Consequently, these systems cannot deal with the ever-growing requirements in data processing [29, 30].

One way to improve on these limitations is by building complex system designs that consist of *independently-operating* resilient clusters that can cooperate to provide certain services. To illustrate this, one can consider a sharded resilient design. In a traditional resilient systems, resilience is provided by a fully-replicated consensus-based Byzantine fault-tolerant cluster in which all replicas hold all data and process all requests. This traditional design has only limited performance, even with the best consensus protocols, and lacks scalability. To improve on the design of traditional systems, one can employ the *sharded* design of Figure 1. In this sharded design, each cluster only holds part of the data. Consequently, each cluster only needs to process requests that affect data they hold. In this way, this sharded design improves performance by enabling *parallel processing* of requests by different clusters, while also improving storage scalability. To support *arbitrary general-purpose workloads* that can affect data in several clusters in such a sharded design, the clusters need to be able to *coordinate their operations*, however [1, 7, 15, 18].[1]

Central to such complex system designs is the ability to reliably and efficiently communicate between independently-operating resilient clusters. Recently, this problem of communication *between* Byzantine fault-tolerant clusters has been formalized as the *cluster-sending problem* [17, 19]. We believe that efficient solutions to this problem have a central role towards bridging *resilient* and *high-performance* data processing.

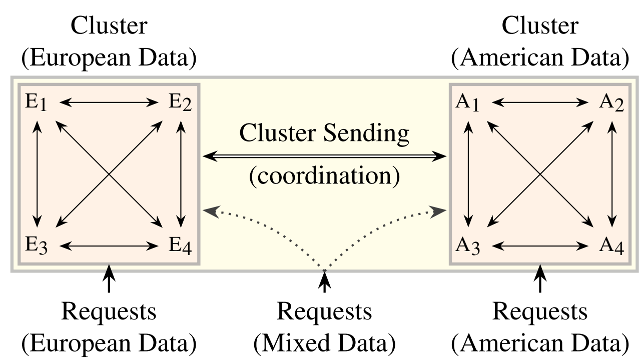

Figure 1: A *sharded* design in which each resilient cluster of four replicas holds only a part of the data. Local decisions within a cluster are made via *consensus* (⟷), whereas multi-shard coordination to process multi-shard transactions requires *cluster-sending* (⟺).

Although the cluster-sending problem has received some attention (e.g., as part of the design of AHL [7], BYSHARD [18], GEOBFT [15], and CHAINSPACE [1]), and

[1] Strict ordering as provided by consensus is necessary to support arbitrary general-purpose workloads. There are classes of operations for which strict consensus-based ordering of (sharding) steps is unnecessary, however. Examples include balance changes and, more generally, operations on CRDTs [31].

cluster-sending protocols that solve the cluster-sending problem with worst-case optimal complexity are known [17, 19], we believe there is still much room for improvement.

In this paper, we introduce a new solution to the cluster-sending problem: we introduce cluster-sending protocols that use *probabilistic cluster-sending* techniques and are able to provide low *expected-case* message complexity (at the cost of higher communication latencies, a good trade-off in systems where inter-cluster network bandwidth is limited). To simplify presentation, we first show how probabilistic cluster-sending works when communication is reliable and synchronous. Then, we generalize these synchronous solutions to practical environments in which communication can be unreliable and asynchronous. Our main contributions are as follows:

1. First, in Section 3, we introduce the cluster-sending step CS-STEP that attempts to send a value from a replica in the sending cluster to a replica in the receiving cluster in a verifiable manner and with a constant amount of inter-cluster communication. This step is guaranteed to perform cluster-sending if communication is reliable and the step is performed by non-faulty replicas.

2. Then, in Section 4, we illustrate the working of a *basic probabilistic cluster-sending protocol* by introducing the *Synchronous Probabilistic Cluster-Sending protocol* CSP. CSP uses CS-STEP with randomly selected sending and receiving replicas to provide cluster-sending in *expected constant* steps. In addition, we show how *pruned*-CSP (CSPP), a fine-tuned variant of CSP, can guarantee cluster-sending in *expected constant* steps while also guaranteeing termination.

3. Next, in Section 5, we propose the *Synchronous Probabilistic Linear Cluster-Sending protocol* CSPL. CSPL not only guarantees cluster-sending in *expected constant* steps, but also guarantees a *worst-case optimal* linear upper-bound on communication. To achieve this worst-case optimal upper-bound, we introuce a specialized randomized scheme via which CSPL selects replicas. To prove the complexity bounds of CSPL, we provide an in-depth analysis of the expected behavior of the randomized scheme we introduce.

4. Next, in Section 6, we generalize CSP, CSPP, and CSPL to practical environments in which communication can be *unreliable and asynchronous*.

5. Finally, in Section 7, we evaluate the behavior of the proposed probabilistic cluster-sending protocols via an in-depth evaluation. In this evaluation, we show that probabilistic cluster-sending protocols has exceptionally low communication costs in comparison with existing cluster-sending protocols, this even in the presence of communication failures.

A summary of our findings in comparison with existing techniques can be found in Figure 2. In Section 2, we introduce the necessary terminology and notation, in Section 8, we compare with related work, and in Section 9, we conclude on our findings.

2 The Cluster-Sending Problem

Before we present our probabilistic cluster-sending techniques, we first introduce all necessary terminology and notation. The formal model we use is based on the formalization of the cluster-sending problem provided by Hellings et al. [17, 19]. If S is a set of replicas, then $\mathsf{f}(S) \subseteq S$ denotes the *faulty replicas* in S, whereas $\mathsf{nf}(S) = S \setminus \mathsf{f}(S)$ denotes the *non-faulty replicas* in S. We write $\mathbf{n}_S = |S|$, $\mathbf{f}_S = |\mathsf{f}(S)|$, and $\mathbf{nf}_S = |\mathsf{nf}(S)| = \mathbf{n}_S - \mathbf{f}_S$ to denote the number of replicas, faulty replicas, and non-faulty replicas in S, respectively. A *cluster* $\mathcal{C}$ is a finite set of replicas. We consider clusters with *Byzantine replicas* that behave in arbitrary manners. In specific, if $\mathcal{C}$ is a cluster, then any malicious adversary can control the replicas in $\mathsf{f}(\mathcal{C})$ at any time, but adversaries cannot bring non-faulty replicas under their control.

Definition 2.1. Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. The *cluster-sending problem* is the problem of sending a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ such that

1. each non-faulty replica in $\mathrm{R} \in \mathsf{nf}(\mathcal{C}_2)$ decides RECEIVE for the value v, which indicates that v was received by R;

2. all non-faulty replicas in $\mathsf{nf}(\mathcal{C}_1)$ decide CONFIRM for the value v, which indicates that v was received by all non-faulty replicas in $\mathsf{nf}(\mathcal{C}_2)$; and

3. non-faulty replicas in $\mathsf{nf}(\mathcal{C}_2)$ only decide RECEIVE for value v if all non-faulty replicas in $\mathsf{nf}(\mathcal{C}_1)$ decided AGREE upon sending v.

We assume that there is no limitation on local communication within a cluster, while global communication between clusters is costly. This model is supported by practice, where communication between wide-area deployments of clusters is up-to-two orders of magnitude more expensive than communication within a cluster [7, 15].

We assume that each cluster can make *local decisions* among all non-faulty replicas, e.g., via a *consensus protocol* such as PBFT (when Byzantine fault tolerance is required) or PAXOS [6, 24] (when crash-fault tolerance suffices). Furthermore, we assume that the replicas in each cluster can certify such local decisions via a *signature scheme*. E.g., a cluster $\mathcal{C}$ can certify a consensus decision on some message m by collecting a set of signatures for m of $\mathbf{f}_{\mathcal{C}} + 1$ replicas in $\mathcal{C}$, guaranteeing one such signature is from a non-faulty replica (which would only signs values on which consensus is reached). We write $\langle m \rangle_{\mathcal{C}}$ to denote a message m certified by $\mathcal{C}$. To minimize the size of certified messages, one can

Figure 2: A comparison of *cluster-sending protocols* that send a value from cluster $\mathcal{C}_1$ with $\mathbf{n}_{\mathcal{C}_1}$ replicas, of which $\mathbf{f}_{\mathcal{C}_1}$ are faulty, to cluster $\mathcal{C}_2$ with $\mathbf{n}_{\mathcal{C}_2}$ replicas, of which $\mathbf{f}_{\mathcal{C}_2}$ are faulty. For each protocol P, *Protocol* specifies its name; *Robustness* specifies the conditions P puts on the clusters; *Message Steps* specifies the number of messages exchanges P performs; *Optimal* specifies whether P is worst-case optimal; and *Unreliable* specifies whether P can deal with unreliable communication.

	Protocol	Robustness[a]	Message Steps (expected-case)	(worst-case)	Optimal	Unreliable
	PBS-cs [17, 19]	$\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > \mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2}$	$\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2} + 1$		✔	✘
	PBS-cs [17, 19]	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	$\max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$		✔	✘
	GeoBFT [15]	$\mathbf{n}_{\mathcal{C}_1} = \mathbf{n}_{\mathcal{C}_2} > 3\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$	$\mathbf{f}_{\mathcal{C}_2} + 1$[b]	$\Omega(\mathbf{f}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2})$	✘	✔
	Chainspace [1]	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	$\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2}$		✘	✘
This Paper	CSPP	$\mathbf{n}_{\mathcal{C}_1} > 2\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 2\mathbf{f}_{\mathcal{C}_2}$	4	$(\mathbf{f}_{\mathcal{C}_1} + 1)(\mathbf{f}_{\mathcal{C}_2} + 1)$	✘	✔
	CSPP	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	$2\frac{1}{4}$	$(\mathbf{f}_{\mathcal{C}_1} + 1)(\mathbf{f}_{\mathcal{C}_2} + 1)$	✘	✔
	CSPL	$\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > \mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2}$	4	$\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2} + 1$	✔	✔
	CSPL	$\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 2(\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2})$	$2\frac{1}{4}$	$\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2} + 1$	✔	✔
	CSPL	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	3	$\max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$	✔	✔

[a]Protocols that have different message step complexities depending on the robustness assumptions have been included for each of the robustness assumptions.
[b]Complexity when the coordinating primary in $\mathcal{C}_1$ is non-faulty and communication is reliable.

utilize a threshold signature scheme [32]. To enable decision making and message certification, we assume, for every cluster $\mathcal{C}$, $\mathbf{n}_{\mathcal{C}} > 2\mathbf{f}_{\mathcal{C}}$, a minimal requirement [9, 25]. Lastly, we assume that there is a common source of randomness for all non-faulty replicas of each cluster, e.g., via a distributed fault-tolerant random coin [3, 4].

3 The Cluster-Sending Step

As the first step toward a probabilistic cluster-sending protocol, we introduce the *cluster-sending step* which tries to perform cluster-sending between a pair of replicas.

If communication is reliable and one knows non-faulty replicas $\mathrm{R}_1 \in \mathsf{nf}(\mathcal{C}_1)$ and $\mathrm{R}_2 \in \mathsf{nf}(\mathcal{C}_2)$, then cluster-sending a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ can be done via a straightforward *cluster-sending step*: one can simply instruct R_1 to send v to R_2. When R_2 receives v, it can disperse v locally in $\mathcal{C}_2$. Unfortunately, we do not know which replicas are faulty and which are non-faulty. Furthermore, it is practically impossible to reliably determine which replicas are non-faulty, as non-faulty replicas can appear faulty due to unreliable communication, while faulty replicas can appear well-behaved to most replicas, while interfering with the operations of only some non-faulty replicas.

To deal with faulty replicas when utilizing the above *cluster-sending step*, one needs a sufficient safeguards to detect *failure* of R_1, of R_2, or of the communication between them. To do so, we add receive and confirmation phases to the sketched cluster-sending step. During the *receive phase*, the receiving replica R_2 must construct a proof P that it received and dispersed v locally in $\mathcal{C}_2$ and then send this proof back to R_1. Finally, during the *confirmation phase*, R_1 can utilize P to prove to all other replicas in $\mathcal{C}_1$ that the cluster-sending step was successful. The pseudo-code of this *cluster-sending step protocol* CS-STEP can be found in Figure 3. We have the following:

Proposition 3.1. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters with $\mathrm{R}_1 \in \mathcal{C}_1$ and $\mathrm{R}_2 \in \mathcal{C}_2$. If $\mathcal{C}_1$ satisfies the pre-conditions of* CS-STEP(R_1, R_2, v)*, then execution of* CS-STEP(R_1, R_2, v) *satisfies the post-conditions and will exchange at most two messages between $\mathcal{C}_1$ and $\mathcal{C}_2$.*

Proof. We prove the three post-conditions separately. **(i)** We assume that communication is reliable, $\mathrm{R}_1 \in \mathsf{nf}(\mathcal{C}_1)$, and $\mathrm{R}_2 \in \mathsf{nf}(\mathcal{C}_2)$. Hence, R_1 sends message $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ to R_2 (Line 1 of Figure 3). In the receive phase (Lines 2–6 of Figure 3), replica R_2 *receives* message m from R_1. Replica R_2 uses local consensus on m to replicate m among all replicas $\mathcal{C}_2$ and, along the way, to constructs a *proof of receipt* $m_p := \langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$. As all replicas in $\mathsf{nf}(\mathcal{C}_2)$ participate in this local consensus, all replicas in $\mathsf{nf}(\mathcal{C}_2)$ will decide RECEIVE on v from $\mathcal{C}_1$. Finally, the proof m_p is returned to R_1. In the confirmation phase (Lines 7–10 of Figure 3), replica R_1 receives the proof of receipt m_p. Next, R_1 uses local consensus on m_p to replicate m_p among all replicas in $\mathsf{nf}(\mathcal{C}_1)$, after which all replicas in $\mathsf{nf}(\mathcal{C}_1)$ decide CONFIRM on sending v to $\mathcal{C}_2$

(ii) A replica in $\mathsf{nf}(\mathcal{C}_2)$ only decides RECEIVE on v after consensus is reached on a message $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ (Line 5 of Figure 3). This message m not only contains the value v, but also the identity of the recipient cluster $\mathcal{C}_2$. Due to the usage of certificates and the pre-condition, the message m cannot be created without the replicas in $\mathsf{nf}(\mathcal{C}_1)$ deciding AGREE on sending v to $\mathcal{C}_2$.

(iii) A replica in $\mathsf{nf}(\mathcal{C}_1)$ only decides CONFIRM on v after consensus is reached on a *proof of receipt* message $m_p := \langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$ (Line 10 of Figure 3). This consensus step will complete for all replicas in $\mathcal{C}_1$ whenever communication becomes reliable. Hence, all replicas in $\mathsf{nf}(\mathcal{C}_1)$ will eventually

Protocol CS-STEP(R_1, R_2, v), with $R_1 \in \mathcal{C}_1$ and $R_2 \in \mathcal{C}_2$**:**

Pre: Each replica in $\mathsf{nf}(\mathcal{C}_1)$ decided AGREE on sending v to $\mathcal{C}_2$ (and can construct $\langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$).

Post: **(i)** If communication is reliable, $R_1 \in \mathsf{nf}(\mathcal{C}_1)$, and $R_2 \in \mathsf{nf}(\mathcal{C}_2)$, then R_1 decides CONFIRM on v. **(ii)** If a replica in $\mathsf{nf}(\mathcal{C}_2)$ decides RECEIVE on v, then all replicas in $\mathsf{nf}(\mathcal{C}_1)$ decided AGREE on sending v to $\mathcal{C}_2$. **(iii)** If a replica in $\mathsf{nf}(\mathcal{C}_1)$ decides CONFIRM on v, then all replicas in $\mathsf{nf}(\mathcal{C}_2)$ decided RECEIVE on v and all replicas in $\mathsf{nf}(\mathcal{C}_1)$ eventually decide CONFIRM on v (whenever communication becomes reliable).

The *cluster-sending step* for R_1 and R_2:

1: Instruct R_1 to send $\langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ to R_2.

The *receive role* for $\mathcal{C}_2$:

2: **event** $R_2 \in \mathsf{nf}(\mathcal{C}_2)$ receives message $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ from $R_1 \in \mathcal{C}_1$ **do**
3: **if** R_2 does not have consensus on m **then**
4: Use *local consensus* on m and construct $\langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$.
5: {*Each replica in* $\mathsf{nf}(\mathcal{C}_2)$ *decides* RECEIVE *on* v.}
6: Send $\langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$ to R_1.

The *confirmation role* for $\mathcal{C}_1$:

7: **event** $R_1 \in \mathsf{nf}(\mathcal{C}_1)$ receives message $m_p := \langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$ with $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ from $R_2 \in \mathcal{C}_2$ **do**
8: **if** R_1 does not have consensus on m_p **then**
9: Use *local consensus* on m_p.
10: {*Each replica in* $\mathsf{nf}(\mathcal{C}_1)$ *decides* CONFIRM *on* v.}

Figure 3: The Cluster-sending step protocol CS-STEP(R_1, R_2, v). In this protocol, R_1 tries to send v to R_2, which will succeed if both R_1 and R_2 are non-faulty.

decide CONFIRM on v. Due to the usage of certificates, the message m_p cannot be created without cooperation of the replicas in $\mathsf{nf}(\mathcal{C}_2)$. The replicas in $\mathsf{nf}(\mathcal{C}_2)$ only cooperate in constructing m_p as part of the consensus step of Line 4 of Figure 3. Upon completion of this consensus step, all replicas in $\mathsf{nf}(\mathcal{C}_2)$ will decide RECEIVE on v. □

In the following sections, we show how to use the cluster-sending step in the construction of cluster-sending protocols. In Section 4, we introduce synchronous protocols that provide *expected constant message complexity*. Then, in Section 5, we introduce synchronous protocols that additionally provide *worst-case linear message complexity*, which is optimal. Finally, in Section 6, we show how to extend the presented techniques to asynchronous communication.

4 Probabilistic Cluster-Sending with Random Replica Selection

In the previous section, we introduced CS-STEP, the cluster-sending step protocol that succeeds whenever the participating replicas are non-faulty and communication is reliable. In this section, we introduce a basic probabilistic cluster-sending protocol that utilizes CS-STEP to perform cluster-sending with expected constant costs.

Using CS-STEP, we build a three-step protocol that cluster-sends a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$:

1. First, the replicas in $\mathsf{nf}(\mathcal{C}_1)$ reach agreement and decide AGREE on sending v to $\mathcal{C}_2$.
2. Then, the replicas in $\mathsf{nf}(\mathcal{C}_1)$ perform a *probabilistic cluster-sending step* by electing replicas $R_1 \in \mathcal{C}_1$ and $R_2 \in \mathcal{C}_2$ fully at random, after which CS-STEP(R_1, R_2, v) is executed.
3. Finally, each replica in $\mathsf{nf}(\mathcal{C}_1)$ waits for the completion of CS-STEP(R_1, R_2, v). If the waiting replicas decided CONFIRM on v during this wait, then cluster-sending is successful. Otherwise, we repeat the previous step.

To simplify presentation, we first present the above protocol assuming *synchronous* inter-cluster communication: in this sectiom, we assume that messages sent by non-faulty replicas will be delivered within some known bounded delay. *Synchronous* systems can be modeled by *pulses* [10, 11]:

Definition 4.1. A system is *synchronous* if all inter-cluster communication happens in *pulses* such that every message sent in a pulse will be received in the same pulse.

We refer to Section 6 on how to generalize the results of this section to practical environments with asynchronous and unreliable communication.

The pseudo-code of the resultant *Synchronous Probabilistic Cluster-Sending protocol* CSP can be found in Figure 4. Next, we prove that CSP is correct and has expected-case constant message complexity:

Protocol CSP($\mathcal{C}_1$, $\mathcal{C}_2$, v)**:**

Use *local consensus* on v and construct $\langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$.
{*Each replica in* $\mathsf{nf}(\mathcal{C}_1)$ *decides* AGREE *on* v.}
repeat
Choose replicas $(R_1, R_2) \in \mathcal{C}_1 \times \mathcal{C}_2$, fully at random.
CS-STEP(R_1, R_2, v)
Wait *three* global pulses.
until $\mathcal{C}_1$ reaches consensus on $\langle \texttt{proof} : \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1} \rangle_{\mathcal{C}_2}$.

Figure 4: The Synchronous Probabilistic Cluster-Sending protocol CSP($\mathcal{C}_1$, $\mathcal{C}_2$, v) that cluster-sends a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$.

Theorem 4.2. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. If communication is synchronous, then* CSP($\mathcal{C}_1$, $\mathcal{C}_2$, v) *results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$. The execution performs two local* *consensus steps in $\mathcal{C}_1$, one local consensus step in $\mathcal{C}_2$, and is expected to make $(\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2})/(\mathbf{nf}_{\mathcal{C}_1}\mathbf{nf}_{\mathcal{C}_1})$ cluster-sending steps.*[2]

Proof. Due to Lines 1–2 of Figure 4, CSP($\mathcal{C}_1$, $\mathcal{C}_2$, v) establishes the pre-conditions for any execution of CS-STEP(R_1, R_2, v) with $R_1 \in \mathcal{C}_1$ and $R_2 \in \mathcal{C}_2$. Using the correctness of CS-STEP (Proposition 3.1), we conclude that CSP($\mathcal{C}_1$, $\mathcal{C}_2$, v) results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$ whenever the replicas $(R_1, R_2) \in \mathcal{C}_1 \times \mathcal{C}_2$ chosen at Line 4 of Figure 4 are non-faulty. As the replicas $(R_1, R_2) \in \mathcal{C}_1 \times \mathcal{C}_2$ are chosen fully at random, we have probability $p_i = \mathbf{nf}_{\mathcal{C}_i}/\mathbf{n}_{\mathcal{C}_i}$, $i \in \{1,2\}$, of choosing $R_i \in \mathsf{nf}(\mathcal{C}_i)$. The probabilities p_1 and p_2 are independent of each other. Consequently, the probability of choosing $(R_1, R_2) \in \mathsf{nf}(\mathcal{C}_1) \times \mathsf{nf}(\mathcal{C}_2)$ is $p = p_1 p_2 = (\mathbf{nf}_{\mathcal{C}_1}\mathbf{nf}_{\mathcal{C}_2})/(\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2})$. As such, each iteration of the loop at Line 3 of Figure 4 can be modeled as an independent *Bernoulli trial* with probability of success p, and the expected number of iterations of the loop is $p^{-1} = (\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2})/(\mathbf{nf}_{\mathcal{C}_1}\mathbf{nf}_{\mathcal{C}_1})$.

Finally, we prove that each local consensus step needs to be performed only once. To do so, we consider the local consensus steps triggered by the loop at Line 3 of Figure 4. These are the local consensus steps at Lines 4 and 9 of Figure 3. The local consensus step at Line 4 can be initiated by a faulty replica R_2. After this single local consensus step reaches consensus on message $m := \langle \texttt{send} : v, \ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$, each replica in $\mathsf{nf}(\mathcal{C}_2)$ reaches consensus on m, decides RECEIVE on v, and can construct $m_p := \langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$, this independent of the behavior of R_2. Hence, a single local consensus step for m in $\mathcal{C}_2$ suffices, and no replica in $\mathsf{nf}(\mathcal{C}_2)$ will participate in future consensus steps for m. An analogous argument proves that a single local consensus step for m_p in $\mathcal{C}_1$, performed at Line 9 of Figure 3, suffices. □

Remark 4.3. Although Theorem 4.2 indicates local consensus steps in clusters $\mathcal{C}_1$ and $\mathcal{C}_2$, these local consensus steps typically come for *free* as part of the protocol that uses cluster-sending as a building block. To see this, we consider a multi-shard transaction τ processed by clusters $\mathcal{C}_1$ and $\mathcal{C}_2$.

The decision of cluster $\mathcal{C}_1$ to send a value v to cluster $\mathcal{C}_2$ is a consequence of the execution of τ in $\mathcal{C}_1$. Before the replicas in $\mathcal{C}_1$ execute τ, they need to reach consensus on the order in which τ is executed in $\mathcal{C}_1$. As part of this consensus step, the replicas in $\mathcal{C}_1$ can also construct $\langle \texttt{send} : v, \ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ without additional consensus steps. Hence, no consensus step is necessary in $\mathcal{C}_1$ to send value v. Likewise, if value v is received by replicas in $\mathcal{C}_2$ as part of some multi-shard transaction execution protocol, then the replicas in $\mathcal{C}_2$ need to perform their portion of the necessary transaction execution steps to execute τ as a *consequence* of receiving v. To do so, the replicas in $\mathcal{C}_2$ need to reach consensus on the order in which these transaction execution steps are performed. As part of this consensus step, the replicas in $\mathcal{C}_2$ can also constructing a proof of receipt for v.

[2]Throughout this paper, the *number of consensus steps* in the presented cluster-sending protocols refers to the *single* consensus step necessary to reach agreement in the sending cluster on sending a value v and all consensus steps performed in all invocations of CS-STEP by the protocol.

In typical fault-tolerant clusters, more than half of the replicas are non-faulty (e.g., in synchronous systems with Byzantine failures that use digital signatures, or in systems that only deal with crashes) or more than two-third of the replicas are non-faulty (e.g., asynchronous systems). In these systems, CSP is expected to only performs a few cluster-sending steps:

Corollary 4.4. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. If communication is synchronous, then the expected number of cluster-sending steps performed by* CSP($\mathcal{C}_1$, $\mathcal{C}_2$, v) *is upper bounded by 4 if $\mathbf{n}_{\mathcal{C}_1} > 2\mathbf{f}_{\mathcal{C}_1}$ and $\mathbf{n}_{\mathcal{C}_2} > 2\mathbf{f}_{\mathcal{C}_2}$; and by $2\frac{1}{4}$ if $\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}$ and $\mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$.*

In CSP, the replicas $(R_1, R_2) \in \mathcal{C}_1 \times \mathcal{C}_2$ are chosen fully at random and *with replacement*, as CSP does not retain any information on *failed* probabilistic steps. In the worst case, this prevents *termination*, as the same pair of replicas can be picked repeatedly. Furthermore, CSP does not prevent the choice of faulty replicas whose failure could be detected. We can easily improve on this, as the *failure* of a probabilistic step provides some information on the chosen replicas. In specific, we have the following technical properties:

Lemma 4.1. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. We assume synchronous communication and assume that each replica in $\mathsf{nf}(\mathcal{C}_1)$ decided* AGREE *on sending v to $\mathcal{C}_2$.*

1. *Let $(R_1, R_2) \in \mathcal{C}_1 \times \mathcal{C}_2$. If* CS-STEP($R_1$, R_2, v) *fails to cluster-send v, then either $R_1 \in \mathsf{f}(\mathcal{C}_1)$, $R_2 \in \mathcal{C}_2$, or both.*
2. *Let $R_1 \in \mathcal{C}_1$. If* CS-STEP(R_1, R_2, v) *fails to cluster-send v for $\mathbf{f}_{\mathcal{C}_2} + 1$ distinct replicas $R_2 \in \mathcal{C}_2$, then $R_1 \in \mathsf{f}(\mathcal{C}_1)$.*
3. *Let $R_2 \in \mathcal{C}_2$. If* CS-STEP(R_1, R_2, v) *fails to cluster-send v for $\mathbf{f}_{\mathcal{C}_1} + 1$ distinct replicas $R_1 \in \mathcal{C}_1$, then $R_2 \in \mathsf{f}(\mathcal{C}_2)$.*

Proof. The statement of this Lemma assumes that the pre-conditions for any execution of CS-STEP(R_1, R_2, v) with $R_1 \in \mathcal{C}_1$ and $R_2 \in \mathcal{C}_2$ are established. Hence, by Proposition 3.1, CS-STEP(R_1, R_2, v) will cluster-send v if $R_1 \in \mathsf{nf}(\mathcal{C}_1)$ and $R_2 \in \mathsf{nf}(\mathcal{C}_2)$. If the cluster-sending step fails to cluster-send v, then one of the replicas involved must be faulty, proving the first property. Next, let $R_1 \in \mathcal{C}_1$ and consider a set $S \subseteq \mathcal{C}_2$ of $\mathbf{n}_S = \mathbf{f}_{\mathcal{C}_2} + 1$ replicas such that, for all $R_2 \in S$, CS-STEP(R_1, R_2, v) fails to cluster-send v. Let $S' = S \setminus \mathsf{f}(\mathcal{C}_2)$ be the non-faulty replicas in S. As $\mathbf{n}_S > \mathbf{f}_{\mathcal{C}_2}$, we have $\mathbf{n}_{S'} \geq 1$ and there exists a $R_2' \in S'$. As $R_2' \notin \mathsf{f}(\mathcal{C}_2)$ and CS-STEP(R_1, R_2', v) fails to cluster-send v, we must have $R_1 \in \mathsf{f}(\mathcal{C}_1)$ by the first property, proving the second property. An analogous argument proves the third property. □

We can apply the properties of Lemma 4.1 to actively *prune* which replica pairs CSP considers (Line 4 of Figure 4). Notice that pruning via Lemma 4.1(1) simply replaces choosing replica pairs *with replacement*, as done by CSP, by choosing replica pairs *without replacement*, this without further reducing the possible search space. Pruning via Lemma 4.1(2) does reduce the search space, however, as each replica in $\mathcal{C}_1$ will only be paired with a subset of $\mathbf{f}_{\mathcal{C}_2}+1$ replicas in $\mathcal{C}_2$. Likewise, pruning via Lemma 4.1(3) also reduces the search space. We obtain the *Pruned Synchronous Probabilistic Cluster-Sending protocol* (CSPP) by applying all three prune steps to CSP. By construction, Theorem 4.2, and Lemma 4.1, we conclude:

Corollary 4.5. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. If communication is synchronous, then* CSPP($\mathcal{C}_1$, $\mathcal{C}_2$, v) *results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$. The execution performs two local consensus steps in $\mathcal{C}_1$, one local consensus step in $\mathcal{C}_2$, is expected to make less than $(\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2})/(\mathbf{nf}_{\mathcal{C}_1}\mathbf{nf}_{\mathcal{C}_1})$ cluster-sending steps, and makes worst-case $(\mathbf{f}_{\mathcal{C}_1}+1)(\mathbf{f}_{\mathcal{C}_2}+1)$ cluster-sending steps.*

5 Worst-Case Linear-Time Probabilistic Cluster-Sending

In the previous section, we introduced CSP and CSPP, two probabilistic cluster-sending protocols that can cluster-send a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ with expected constant cost. Unfortunately, CSP does not guarantee termination, while CSPP has a worst-case *quadratic complexity*. In this section, we improve on this by presenting a probabilistic cluster-sending protocol that has expected constant cost and guarantees termination with a worst-case *optimal linear complexity* [17,19]. We refer to Table 1 for an overview of the notation used in this section.

To improve on CSP and CSPP, we need to improve the scheme by which we select replica pairs $(\mathrm{R}_1, \mathrm{R}_2) \in \mathcal{C}_1 \times \mathcal{C}_2$ that we use in cluster-sending steps. The straightforward manner to guarantee a worst-case *linear complexity* is by using a scheme that can select only up-to-$n = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$ distinct pairs $(\mathrm{R}_1, \mathrm{R}_2) \in \mathcal{C}_1 \times \mathcal{C}_2$. To select n replica pairs from $\mathcal{C}_1 \times \mathcal{C}_2$, we will proceed in two steps.

1. We generate list S_1 of n replicas taken from $\mathcal{C}_1$ and list S_2 of n replicas taken from $\mathcal{C}_2$.

2. Then, we choose permutations $P_1 \in \mathsf{perms}(S_1)$ and $P_2 \in \mathsf{perms}(S_2)$ fully at random, and interpret each pair $(P_1[i], P_2[i])$. $0 \le i < n$, as one of the chosen replica pairs.

We use the first step to deal with any differences in the sizes of $\mathcal{C}_1$ and $\mathcal{C}_2$, and we use the second step to introduce sufficient randomness in our protocol to yield a low expected-case message complexity.

Next, we introduce some notations to simplify reasoning about the above list-based scheme. If R is a set of replicas, then $\mathsf{list}(R)$ is the list consisting of the replicas in R placed in a predetermined order (e.g., on increasing replica identifier). If S is a list of replicas, then we write $\mathsf{f}(S)$ to denote the faulty replicas in S and $\mathsf{nf}(S)$ to denote the non-faulty replicas in S, and we write $\mathbf{n}_S = |S|$, $\mathbf{f}_S = |\{i \mid (0 \le i < \mathbf{n}_S) \wedge S[i] \in \mathsf{f}(S)\}|$, and $\mathbf{nf}_S = \mathbf{n}_S - \mathbf{f}_S$ to denote the number of positions in S with replicas, faulty replicas, and non-faulty replicas, respectively. If (P_1, P_2) is a pair of equal-length lists of $n = |P_1| = |P_2|$ replicas, then we say that the i-th position is a *faulty position* if either $P_1[i] \in \mathsf{f}(P_1)$ or $P_2[i] \in \mathsf{f}(P_2)$. We write $\|P_1; P_2\|_{\mathbf{f}}$ to denote the number of *faulty positions* in (P_1, P_2). As faulty positions can only be constructed out of the $\mathbf{f}_{P_1}$ faulty replicas in P_1 and the $\mathbf{f}_{P_2}$ faulty replicas in P_2, we must have $\max(\mathbf{f}_{P_1}, \mathbf{f}_{P_2}) \le \|P_1; P_2\|_{\mathbf{f}} \le \min(n, \mathbf{f}_{P_1} + \mathbf{f}_{P_2})$.

Table 1: Overview of the notation used in Section 5.

Notation	Description
P_1, P_2	Permutation of a list of replicas from $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively.
m_1, m_2	Given a pair of lists of replicas (P_1, P_2), the number of faulty replicas in list P_1 and P_2, respectively.
b_1	The number 1-*faulty pairs* in a given pair of lists of replicas (P_1, P_2).
b_2	The number 2-*faulty pairs* in a given pair of lists of replicas (P_1, P_2).
$b_{1,2}$	The number of *both-faulty pairs* in a given pair of lists of replicas (P_1, P_2).
$\mathsf{list}(R)$	A list-representation of the replica set R.
$\mathsf{perms}(S)$	Permutation of list of replicas S.
$S^{:n}$	The first n elements in the list obtained by repeatedly concatenating list S.
$L\|_M$	Tthe list obtained from L by only keeping the values that also appear in list M.
$\mathbb{M}(v, w)$	The number of distinct ways in which two lists of v and w elements, respectively, can be merged together (without shuffling elements from their respective lists).
Φ	A list-pair function.
$\|P_1; P_2\|_{\mathbf{f}}$	The number of faulty positions in (P_1, P_2).
$\mathbb{F}(n, m_1, m_2, k)$	The number of permutations (P_1, P_2) with k faulty positions of two given lists of n replicas of which m_1 and m_2 replicas are faulty, respectively.
$\mathbb{E}(n, m_1, m_2)$	The non-faulty position trials problem with two lists of n replicas of which m_1 and m_2 replicas are faulty, respectively.

Example 5.1. Consider clusters $\mathcal{C}_1, \mathcal{C}_2$ with

$$S_1 = \mathsf{list}(\mathcal{C}_1) = [\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,5}], \quad \mathsf{f}(\mathcal{C}_1) = \{\mathrm{R}_{1,1}, \mathrm{R}_{1,2}\};$$
$$S_2 = \mathsf{list}(\mathcal{C}_2) = [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,5}], \quad \mathsf{f}(\mathcal{C}_2) = \{\mathrm{R}_{2,1}, \mathrm{R}_{2,2}\}.$$

The set $\mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ contains $5!^2 = 14400$ list pairs. Now, consider the list pairs $(P_1, P_2), (Q_1, Q_2), (R_1, R_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with

$$P_1[\underline{\mathrm{R}_{1,1}}, \mathrm{R}_{1,5}, \underline{\mathrm{R}_{1,2}}, \mathrm{R}_{1,4}, \mathrm{R}_{1,3}],$$
$$P_2[\underline{\mathrm{R}_{2,1}}, \mathrm{R}_{2,3}, \underline{\mathrm{R}_{2,2}}, \mathrm{R}_{2,5}, \mathrm{R}_{2,4}];$$

Protocol CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, Φ)**:**

```
1: Use local consensus on v and construct ⟨send : v, C2⟩_C1.
2: {Each replica in nf(C1) decides AGREE on v.}
3: Let (S1, S2) := Φ(C1, C2).
4: Choose (P1, P2) ∈ perms(S1) × perms(S2) fully at random.
5: i := 0.
6: repeat
7:    CS-STEP(P1[i], P2[i], v)
8:    Wait three global pulses.
9:    i := i + 1.
10: until C1 reaches consensus on ⟨proof : ⟨send : v, C2⟩_C1⟩_C2.
```

Figure 5: The Synchronous Probabilistic Linear Cluster-Sending protocol CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, Φ) that cluster-sends a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ using list-pair function Φ.

$$Q_1[\underline{\mathrm{R}_{1,1}}, \mathrm{R}_{1,3}, \mathrm{R}_{1,5}, \mathrm{R}_{1,4}, \underline{\mathrm{R}_{1,2}}],$$
$$Q_2[\mathrm{R}_{2,5}, \mathrm{R}_{2,4}, \mathrm{R}_{2,3}, \underline{\mathrm{R}_{2,2}}, \underline{\mathrm{R}_{2,1}}];$$
$$R_1[\mathrm{R}_{1,5}, \mathrm{R}_{1,4}, \mathrm{R}_{1,3}, \underline{\mathrm{R}_{1,2}}, \underline{\mathrm{R}_{1,1}}],$$
$$R_2[\underline{\mathrm{R}_{2,1}}, \underline{\mathrm{R}_{2,2}}, \mathrm{R}_{2,3}, \mathrm{R}_{2,4}, \mathrm{R}_{2,5}].$$

We have underlined the faulty replicas in each list, and we have $\|P_1;P_2\|_{\mathbf{f}} = 2 = \mathbf{f}_{S_1} = \mathbf{f}_{S_2}$, $\|Q_1;Q_2\|_{\mathbf{f}} = 3$, and $\|R_1;R_2\|_{\mathbf{f}} = 4 = \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$.

In the following, we will use a *list-pair function* Φ to compute the initial list-pair (S_1, S_2) of n replicas taken from $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. We build a cluster-sending protocol that uses Φ to compute S_1 and S_2, uses randomization to choose n replica pairs from $S_1 \times S_2$, and, finally, performs cluster-sending steps using only these n replica pairs. The pseudo-code of the resultant *Synchronous Probabilistic Linear Cluster-Sending protocol* CSPL can be found in Figure 5. Next, we prove that CSPL is correct and has a worst-case linear message complexity:

Proposition 5.1. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters and let Φ be a list-pair function with $(S_1, S_2) := \Phi(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. If communication is synchronous and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$, then* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, Φ) *results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$. The execution performs two local consensus steps in $\mathcal{C}_1$, one local consensus step in $\mathcal{C}_2$, and makes worst-case $\mathbf{f}_{S_1} + \mathbf{f}_{S_2} + 1$ cluster-sending steps.*

Proof. Due to Lines 1–2 of Figure 5, CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, Φ) establishes the pre-conditions for any execution of CS-STEP(R_1, R_2, v) with $\mathrm{R}_1 \in \mathcal{C}_1$ and $\mathrm{R}_2 \in \mathcal{C}_2$. Now let $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$, as chosen at Line 4 of Figure 5. As P_i, $i \in \{1, 2\}$, is a permutation of S_i, we have $\mathbf{f}_{P_i} = \mathbf{f}_{S_i}$. Hence, we have $\|P_1;P_2\|_{\mathbf{f}} \leq \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ and there must exist a position j, $0 \leq j < n$, such that $(P_1[j], P_2[j]) \in \mathsf{nf}(\mathcal{C}_1) \times \mathsf{nf}(\mathcal{C}_2)$. Using the correctness of CS-STEP (Proposition 3.1), we conclude that CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, Φ) results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$ in at most $\mathbf{f}_{S_1} + \mathbf{f}_{S_2} + 1$ cluster-sending steps. Finally, the bounds on the number of consensus steps follow from an argument analogous to the one in the proof of Theorem 4.2. □

Proposition 5.1 only shows that CSPL will perform cluster-sending when specific conditions are met on the list-pair function. Next, we proceed in two steps to arrive at practical list-pair functions for CSPL that can be used in combination with CSPL to guarantee an expected constant cost. First, in Section 5.1, we study the probabilistic nature of CSPL. Then, in Section 5.2, we propose practical list-pair functions and show that these functions yield instances of CSPL with expected constant message complexity.

5.1 The Expected-Case Complexity of CSPL

The expected-case analysis of CSP and CSPP was rather straightforward, as the sending and receiving replicas used by these protocols are chosen fully at random and independent of each other. Hence, the random choices made by both protocols can be modelled via well-known independent Bernoulli trials (see the proof of Theorem 4.2). In CSPL, the choice of sending and receiving replicas are dependent, as they are chosen from a list of possible replica pairs. As such, the random choices made by CSPL can no longer be modelled via independent Bernoulli trials. Hence, the expected-case analysis of CSPL requires a further analysis of the probabilistic nature of the randomized scheme used by CSPL.

As the first step toward this analysis, we solve the following abstract problem that captures the probabilistic argument at the core of the expected-case complexity of CSPL:

Problem 5.2 (non-faulty position trials)**.** Let S_1 and S_2 be lists of $|S_1| = |S_2| = n$ replicas. Choose permutations $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ fully at random. Next, we inspect positions in P_1 and P_2 fully at random (with replacement). The *non-faulty position trials problem* asks how many positions one expects to inspect to find the first non-faulty position.

Let S_1 and S_2 be lists of $|S_1| = |S_2| = n$ replicas. To answer the non-faulty position trials problem, we first look at the combinatorics of *faulty positions* in pairs $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. Let $m_1 = \mathbf{f}_{S_1}$ and $m_2 = \mathbf{f}_{S_2}$. By $\mathbb{F}(n, m_1, m_2, k)$, we denote the number of distinct pairs (P_1, P_2) one can construct that have exactly k faulty positions, hence, with $\|P_1;P_2\|_{\mathbf{f}} = k$. As observed, we have $\max(m_1, m_2) \leq \|P_1;P_2\|_{\mathbf{f}} \leq \min(n, m_1 + m_2)$ for any pair (P_1, P_2). Hence, we have $\mathbb{F}(n, m_1, m_2, k) = 0$ for all $k < \max(m_1, m_2)$ and $k > \min(n, m_1 + m_2)$.

Now consider the step-wise construction of any permutation $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with k faulty positions. First, we choose $(P_1[0], P_2[0])$, the pair at position 0, after which we choose pairs for the remaining $n-1$ positions. For $P_i[0]$, $i \in \{1, 2\}$, we can choose n distinct replicas, of which m_i are faulty. If we pick a non-faulty replica, then the remainder of P_i is constructed out of $n-1$ replicas, of which

m_i are faulty. Otherwise, the remainder of P_i is constructed out of $n-1$ replicas of which $m_i - 1$ are faulty. If, due to our choice of $(P_1[0], P_2[0])$, the first position is faulty, then only $k-1$ out of the $n-1$ remaining positions must be faulty. Otherwise, k out of the $n-1$ remaining positions must be faulty. Combining this analysis yields four types for the first pair $(P_1[0], P_2[0])$:

1. *A non-faulty pair* $(P_1[0], P_2[0]) \in \mathsf{nf}(P_1) \times \mathsf{nf}(P_2)$. We have $(n-m_1)(n-m_2)$ such pairs, and we have $\mathbb{F}(n-1, m_1, m_2, k)$ different ways to construct the remainder of P_1 and P_2.

2. *A* 1*-faulty pair* $(P_1[0], P_2[0]) \in \mathsf{f}(P_1) \times \mathsf{nf}(P_2)$. We have $m_1(n-m_2)$ such pairs, and we have $\mathbb{F}(n-1, m_1-1, m_2, k-1)$ different ways to construct the remainder of P_1 and P_2.

3. *A* 2*-faulty pair* $(P_1[0], P_2[0]) \in \mathsf{nf}(P_1) \times \mathsf{f}(P_2)$. We have $(n-m_1)m_2$ such pairs, and we have $\mathbb{F}(n-1, m_1, m_2-2, k-1)$ different ways to construct the remainder of P_1 and P_2.

4. *A both-faulty pair* $(P_1[0], P_2[0]) \in \mathsf{f}(P_1) \times \mathsf{f}(P_2)$. We have $m_1 m_2$ such pairs, and we have $\mathbb{F}(n-1, m_1-1, m_2-1, k-1)$ different ways to construct the remainder of P_1 and P_2.

Hence, for all k, $\max(m_1, m_2) \leq k \leq \min(n, m_1+m_2)$, $\mathbb{F}(n, m_1, m_2, k)$ is recursively defined by:

$$\begin{aligned}
\mathbb{F}(n, m_1, m_2, k) = {} & (n-m_1)(n-m_2)\mathbb{F}(n-1, m_1, m_2, k) \\
& \hfill \text{(non-faulty pair)} \\
& + m_1(n-m_2)\mathbb{F}(n-1, m_1-1, m_2, k-1) \\
& \hfill \text{(1-faulty pair)} \\
& + (n-m_1)m_2\mathbb{F}(n-1, m_1, m_2-1, k-1) \\
& \hfill \text{(2-faulty pair)} \\
& + m_1 m_2 \mathbb{F}(n-1, m_1-1, m_2-1, k-1), \\
& \hfill \text{(both-faulty pair)}
\end{aligned}$$

and the base case for this recursion is $\mathbb{F}(0,0,0,0) = 1$.

Example 5.3. Reconsider the list pairs (P_1, P_2), (Q_1, Q_2), and (R_1, R_2) from Example 5.1. In (P_1, P_2), we have both-faulty pairs at positions 0 and 2 and non-faulty pairs at positions 1, 3, and 4. In (Q_1, Q_2), we have a 1-faulty pair at position 0, non-faulty pairs at positions 1 and 2, a 2-faulty pair at position 3, and a both-faulty pair at position 4. Finally, in (R_1, R_2), we have 2-faulty pairs at positions 0 and 1, a non-faulty pair at position 2, and 1-faulty pairs at positions 3 and 4.

Using the combinatorics of faulty positions, we formalize an exact solution to the *non-faulty position trials problem*:

Lemma 5.1. *Let S_1 and S_2 be lists of $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$ replicas with $m_1 = \mathbf{f}_{S_1}$ and $m_2 = \mathbf{f}_{S_2}$. If $m_1 + m_2 < n$, then the non-faulty position trials problem $\mathbb{E}(n, m_1, m_2)$ has solution*

$$\frac{1}{n!^2}\left(\sum_{k=\max(m_1,m_2)}^{m_1+m_2} \frac{n}{n-k}\mathbb{F}(n, m_1, m_2, k)\right).$$

Proof. We have $|\mathsf{perms}(S_1)| = |\mathsf{perms}(S_2)| = n!$. Consequently, we have $|\mathsf{perms}(S_1) \times \mathsf{perms}(S_2)| = n!^2$ and we have probability $1/(n!^2)$ to choose any pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. Now consider such a pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. As there are $\|P_1; P_2\|_{\mathbf{f}}$ faulty positions in (P_1, P_2), we have probability $p(P_1, P_2) = (n - \|P_1; P_2\|_{\mathbf{f}})/n$ to inspect a non-faulty position. Notice that $\max(m_1, m_2) \leq \|P_1; P_2\|_{\mathbf{f}} \leq m_1 + m_2 < n$ and, hence, $0 < p(P_1, P_2) \leq 1$. Each of the inspected positions in (P_1, P_2) is chosen fully at random. Hence, each inspection is a *Bernoulli trial* with probability of success $p(P_1, P_2)$, and we expect to inspect a first non-faulty position in the $p(P_1, P_2)^{-1} = n/(n - \|P_1; P_2\|_{\mathbf{f}})$-th attempt. We conclude that the non-faulty position trials problem $\mathbb{E}(n, m_1, m_2)$ has solution

$$\frac{1}{n!^2}\left(\sum_{(P_1,P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)} \frac{n}{n - \|P_1; P_2\|_{\mathbf{f}}}\right).$$

Notice that there are $\mathbb{F}(n, m_1, m_2, k)$ distinct pairs $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $\|P_1'; P_2'\|_{\mathbf{f}} = k$ for each k, $\max(m_1, m_2) \leq k \leq m_1 + m_2 < n$. Hence, in the above expression for $\mathbb{E}(n, m_1, m_2)$, we can group on these pairs (P_1', P_2') to obtain the searched-for solution. ☐

To further solve the non-faulty position trials problem, we work towards a *closed form* for $\mathbb{F}(n, m_1, m_2, k)$. Consider any pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $\|P_1; P_2\|_{\mathbf{f}} = k$ obtained via the outlined step-wise construction. Let b_1 be the number of 1*-faulty pairs*, let b_2 be the number of 2*-faulty pairs*, and let $b_{1,2}$ be the number of *both-faulty pairs* in (P_1, P_2). By construction, we must have $k = b_1 + b_2 + b_{1,2}$, $m_1 = b_1 + b_{1,2}$, and $m_2 = b_2 + b_{1,2}$ and by rearranging terms, we can derive

$$b_{1,2} = (m_1 + m_2) - k, \quad b_1 = k - m_2, \quad b_2 = k - m_1.$$

Example 5.4. Consider

$$\begin{aligned}
S_1 &= [\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,5}], & \mathsf{f}(S_1) &= \{\mathrm{R}_{1,1}, \mathrm{R}_{1,2}, \mathrm{R}_{1,3}\}; \\
S_2 &= [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,5}], & \mathsf{f}(S_2) &= \{\mathrm{R}_{2,1}\}.
\end{aligned}$$

Hence, we have $n = 5$, $m_1 = \mathbf{f}_{S_1} = 3$, and $m_2 = \mathbf{f}_{S_2} = 1$. If we want to create a pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $k = \|P_1; P_2\|_{\mathbf{f}} = 3$ faulty positions, then (P_1, P_2) must have two non-faulty pairs, two 1-faulty pairs, no 2-faulty pairs, and one both-faulty pair. Hence, we have $n - k = 2$, $b_1 = 2$, $b_2 = 0$, and $b_{1,2} = 1$.

The above analysis only depends on the choice of m_1, m_2, and k, and not on our choice of (P_1, P_2). Next, we use this analysis to express $\mathbb{F}(n, m_1, m_2, k)$ in terms of the number of distinct ways in which one can *construct*

(A) lists of b_1 1-faulty pairs out of faulty replicas from S_1 and non-faulty replicas from S_2,

(B) lists of b_2 2-faulty pairs out of non-faulty replicas from S_1 and faulty replicas from S_2,

(C) lists of $b_{1,2}$ both-faulty pairs out of the remaining faulty replicas in S_1 and S_2 that are not used in the previous two cases, and

(D) lists of $n-k$ non-faulty pairs out of the remaining (non-faulty) replicas in S_1 and S_2 that are not used in the previous three cases;

and in terms of the number of distinct ways one can *merge* these lists. As the first step, we look at how many distinct ways we can merge two lists together:

Lemma 5.2. *For any two disjoint lists S and T with $|S| = v$ and $|T| = w$, there exist $\mathbb{M}(v, w) = (v+w)!/(v!w!)$ distinct lists L with $L|_S = S$ and $L|_T = T$, in which $L|_M$, $M \in \{S, T\}$, is the list obtained from L by only keeping the values that also appear in list M.*

Next, we look at the number of distinct ways in which one can construct lists of type A, B, C, and D. Consider the construction of a list of type A. We can choose $\binom{m_1}{b_1}$ distinct sets of b_1 faulty replicas from S_1 and we can choose $\binom{n-m_2}{b_1}$ distinct sets of b_1 non-faulty replicas from S_2. As we can order the chosen values from S_1 and S_2 in $b_1!$ distinct ways, we can construct $b_1!^2\binom{m_1}{b_1}\binom{n-m_2}{b_1}$ distinct lists of type A. Likewise, we can construct $b_2!^2\binom{n-m_1}{b_2}\binom{m_2}{b_2}$ distinct lists of type B.

Example 5.5. We continue from the setting of Example 5.4: we want to create a pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $k = \|P_1; P_2\|_{\mathbf{f}} = 3$ faulty positions. To create (P_1, P_2), we need to create $b_1 = 2$ pairs that are 1-faulty. We have $\binom{m_1}{b_1} = \binom{3}{2} = 3$ sets of two faulty replicas in S_1 that we can choose, namely the sets $\{\mathrm{R}_{1,1}, \mathrm{R}_{1,2}\}$, $\{\mathrm{R}_{1,1}, \mathrm{R}_{1,3}\}$, and $\{\mathrm{R}_{1,2}, \mathrm{R}_{1,3}\}$. Likewise, we have $\binom{n-m_2}{b_1} = \binom{4}{2} = 6$ sets of two non-faulty replicas in S_2 that we can choose. Assume we choose $T_1 = \{\mathrm{R}_{1,1}, \mathrm{R}_{1,3}\}$ from S_1 and $T_2 = \{\mathrm{R}_{2,4}, \mathrm{R}_{2,5}\}$ from S_2. The two replicas in T_1 can be ordered in $\mathbf{n}_{T_1}! = 2! = 2$ ways, namely $[\mathrm{R}_{1,1}, \mathrm{R}_{1,3}]$ and $[\mathrm{R}_{1,3}, \mathrm{R}_{1,1}]$. Likewise, the two replicas in T_2 can be ordered in $\mathbf{n}_{T_2}! = 2! = 2$ ways. Hence, we can construct $2 \cdot 2 = 4$ distinct lists of type A out of this single choice for T_1 and T_2, and the sequences S_1 and S_2 provide us with $\binom{m_1}{b_1}\binom{n-m_2}{b_1} = 18$ distinct choices for T_1 and T_2. We conclude that we can construct 72 distinct lists of type A from S_1 and S_2.

By construction, lists of type A and type B cannot utilize the same replicas from S_1 or S_2. After choosing $b_1 + b_2$ replicas in S_1 and S_2 for the construction of lists of type A and B, the remaining $b_{1,2}$ faulty replicas in S_1 and S_2 are all used for constructing lists of type C. As we can order these remaining values from S_1 and S_2 in $b_{1,2}!$ distinct ways, we can construct $b_{1,2}!^2$ distinct lists of type C (per choice of lists of type A and B). Likewise, the remaining $n-k$ non-faulty replicas in S_1 and S_2 are all used for constructing lists of type D, and we can construct $(n-k)!^2$ distinct lists of type D (per choice of lists of type A and B).

As the final steps, we merge lists of type A and B into lists of type AB. We can do so in $\mathbb{M}(b_1, b_2)$ ways and the resultant lists have size $b_1 + b_2$. Next, we merge lists of type AB and C into lists of type ABC. We can do so in $\mathbb{M}(b_1+b_2, b_{1,2})$ ways and the resultant lists have size k. Finally, we merge list of type ABC and D together, which we can do in $\mathbb{M}(k, n-k)$ ways. From this construction, we derive that $\mathbb{F}(n, m_1, m_2, k)$ is equivalent to

$$b_1!^2\binom{m_1}{b_1}\binom{n-m_2}{b_1}b_2!^2\binom{n-m_1}{b_2}\binom{m_2}{b_2}\cdot \mathbb{M}(b_1,b_2)b_{1,2}!^2\mathbb{M}(b_1+b_2,b_{1,2})(n-k)!^2\mathbb{M}(k,n-k),$$

which can be simplified to the following (see Appendix B):

Lemma 5.3. *Let* $\max(m_1, m_2) \leq k \leq \min(n, m_1+m_2)$ *and let $b_1 = k - m_2$, $b_2 = k - m_1$, and $b_{1,2} = (m_1+m_2) - k$. We have*

$$\mathbb{F}(n, m_1, m_2, k) = \frac{m_1!m_2!(n-m_1)!(n-m_2)n!}{b_1!b_2!b_{1,2}!(n-k)!}.$$

Proof. We write $f(n, m_1, m_2, k)$ for the closed form in the statement of this lemma and we prove the statement of this lemma by induction. First, the base case $\mathbb{F}(0,0,0,0)$. In this case, we have $n = m_1 = m_2 = k = 0$ and, hence, $b_1 = b_2 = b_{1,2} = 0$, and we conclude $f(0,0,0,0) = 1 = \mathbb{F}(0,0,0,0)$.

Now assume $\mathbb{F}(n', m_1', m_2', k') = f(n', m_1', m_2', k')$ for all $n' < n$ and all k' with $\max(m_1', m_2') \leq k' \leq \min(n', m_1' + m_2')$. Next, we prove $\mathbb{F}(n, m_1, m_2, k) = f(n, m_1, m_2, k)$ with $\max(m_1, m_2) \leq k \leq \min(n, m_1+m_2)$. We use the shorthand $\mathbb{G} = \mathbb{F}(n, m_1, m_2, k)$ and we have

$$\begin{aligned}\mathbb{G} = {} & (n-m_1)(n-m_2)\mathbb{F}(n-1, m_1, m_2, k) && \text{(non-faulty pair)}\\ & + m_1(n-m_2)\mathbb{F}(n-1, m_1-1, m_2, k-1) && \text{(1-faulty pair)}\\ & + (n-m_1)m_2\mathbb{F}(n-1, m_1, m_2-1, k-1) && \text{(2-faulty pair)}\\ & + m_1m_2\mathbb{F}(n-1, m_1-1, m_2-1, k-1). && \text{(both-faulty pair)}\end{aligned}$$

Notice that if $n = k$, then the non-faulty pair case does not apply, as $\mathbb{F}(n-1, m_1, m_2, k) = 0$, and evaluates to zero. Likewise, if $b_1 = 0$, then the 1-faulty pair case does not apply, as $\mathbb{F}(n-1, m_1-1, m_2, k-1) = 0$, and evaluates to zero; if $b_2 = 0$, then the 2-faulty pair case does not apply, as $\mathbb{F}(n-1, m_1, m_2-1, k-1) = 0$, and evaluates to zero; and, finally, if $b_{1,2} = 0$, then the both-faulty pair case does not apply, as $\mathbb{F}(n-1, m_1-1, m_2-1, k-1) = 0$, and evaluates to zero.

First, we consider the case in which $n > k$, $b_1 > 0$, $b_2 > 0$, and $b_{1,2} > 0$. Hence, each of the four cases apply and evaluate to non-zero values. We directly apply the induction hypothesis on $\mathbb{F}(n-1, m_1, m_2, k)$, $\mathbb{F}(n-1, m_1-1, m_2, k-1)$, $\mathbb{F}(n-1, m_1, m_2-1, k-1)$, and $\mathbb{F}(n-1, m_1-1, m_2-1, k-1)$, and obtain

$$\begin{aligned}
\mathbb{G} = {} & (n-m_1)(n-m_2) \cdot \\
& \quad \frac{m_1!m_2!(n-1-m_1)!(n-1-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-1-k)!} \\
& + m_1(n-m_2) \cdot \\
& \quad \frac{(m_1-1)!m_2!(n-m_1)!(n-1-m_2)!(n-1)!}{(b_1-1)!b_2!b_{1,2}!(n-1-(k-1))!} \\
& + (n-m_1)m_2 \cdot \\
& \quad \frac{m_1!(m_2-1)!(n-1-m_1)!(n-m_2)!(n-1)!}{b_1!(b_2-1)!b_{1,2}!(n-1-(k-1))!} \\
& + m_1m_2 \cdot \\
& \quad \frac{(m_1-1)!(m_2-1)!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!(b_{1,2}-1)!(n-1-(k-1))!}.
\end{aligned}$$

We apply $x! = x(x-1)!$ and further simplify and obtain

$$\begin{aligned}
\mathbb{G} = {} & \frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-1-k)!} \\
& + \frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{(b_1-1)!b_2!b_{1,2}!(n-k)!} \\
& + \frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!(b_2-1)!b_{1,2}!(n-k)!} \\
& + \frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!(b_{1,2}-1)!(n-k)!} \\
= {} & (n-k)\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\
& + b_1\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\
& + b_2\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\
& + b_{1,2}\frac{m-1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!}.
\end{aligned}$$

We have $k = b_1 + b_2 + b_{1,2}$ and, hence, $n = (n-k) + b_1 + b_2 + b_{1,2}$ and we conclude

$$\begin{aligned}
\mathbb{G} &= ((n-k) + b_1 + b_2 + b_{1,2}) \cdot \\
& \qquad \frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\
&= n\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\
&= \frac{m_1!m_2!(n-m_1)!(n-m_2)!n!}{b_1!b_2!b_{1,2}!(n-k)!}.
\end{aligned}$$

Next, in all other cases, we can repeat the above derivation while removing the terms corresponding to the cases that evaluate to 0. By doing so, we end up with the expression

$$\mathbb{G} = \frac{((\sum_{t \in T}\ t)m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!}.$$

in which T contains the term $(n-k)$ if $n > k$ (the non-faulty pair case applies), the term b_1 if $b_1 > 0$ (the 1-faulty case applies), the term b_2 if $b_2 > 0$ (the 2-faulty case applies), and the term $b_{1,2}$ if $b_{1,2} > 0$ (the both-faulty case applies). As each term $(n-k)$, b_1, b_2, and $b_{1,2}$ is in T whenever the term is non-zero, we have $\sum_{t \in T}\ t = (n-k) + b_1 + b_2 + b_{1,2} = n$. Hence, we can repeat the steps of the above derivation in all cases, and complete the proof. ☐

We combine Lemma 5.1 and Lemma 5.3 to conclude

Proposition 5.2. *Let S_1 and S_2 be lists of $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$ replicas with $m_1 = \mathbf{f}_{S_1}$, $m_2 = \mathbf{f}_{S_2}$, $b_1 = k - m_2$, $b_2 = k - m_1$, and $b_{1,2} = (m_1 + m_2) - k$. If $m_1 + m_2 < n$, then the non-faulty position trials problem $\mathbb{E}(n, m_1, m_2)$ has solution*

$$\frac{1}{n!^2}\left(\sum_{k=\max(m_1,m_2)}^{m_1+m_2} \frac{n}{n-k}\frac{m_1!m_2!(n-m_1)!(n-m_2)!n!}{b_1!b_2!b_{1,2}!(n-k)!}\right).$$

Finally, we use Proposition 5.2 to derive

Proposition 5.3. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters and let Φ be a list-pair function with $(S_1, S_2) := \Phi(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. If communication is synchronous and $\mathbf{f}_{S_1} + \mathbf{f}_{S_2} < n$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, Φ) *is less than $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$.*

Proof. Let $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. We notice that CSPL inspects positions in P_1 and P_2 in a different way than the non-faulty trials problem: at Line 7 of Figure 5, positions are inspected one-by-one in a predetermined order and not fully at random (with replacement). Next, we will argue that $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$ provides an upper bound on the expected number of cluster-sending steps regardless of these differences. Without loss of generality, we assume that S_1 and S_2 each have n distinct replicas. Consequently, the pair (P_1, P_2) represents a set R of n distinct replica pairs taken from $\mathcal{C}_1 \times \mathcal{C}_2$.

We notice that each of the $n!$ permutations of R is represented by a single pair $(P'_1, P'_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$.

Now consider the selection of positions in (P_1, P_2) fully at random, but without replacement. This process will yield a list $[j_0, \dots, j_{n-1}] \in \mathsf{perms}([0, \dots, n-1])$ of positions fully at random. Let $Q_i = [P_i[j_0], \dots, P_i[j_{n-1}]]$, $i \in \{1,2\}$. We notice that the pair (Q_1, Q_2) also represents R and we have $(Q_1, Q_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. Hence, by choosing a pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$, we choose set R fully at random and, at the same time, we choose the order in which replica pairs in R are inspected fully at random.

Finally, we note that CSPL inspects positions without replacement. As the number of expected positions inspected in the non-faulty position trials problem decreases if we choose positions without replacement, we have proven that $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$ is an upper bound on the expected number of cluster-sending steps. □

5.2 Practical Instances of CSPL

As the last step in providing practical instances of CSPL, we need to provide practical list-pair functions to be used in conjunction with CSPL. We provide two such functions that address most practical environments. Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters, let $n_{\min} = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$, and let $n_{\max} = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$. We provide list-pair functions

$$\Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2) \mapsto (\mathsf{list}(\mathcal{C}_1)^{:n_{\min}}, \mathsf{list}(\mathcal{C}_2)^{:n_{\min}}),$$
$$\Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2) \mapsto (\mathsf{list}(\mathcal{C}_2)^{:n_{\max}}, \mathsf{list}(\mathcal{C}_2)^{:n_{\max}}),$$

in which $L^{:n}$ denotes the first n values in the list obtained by repeating list L. Next, we illustrate usage of these functions:

Example 5.6. Consider clusters $\mathcal{C}_1, \mathcal{C}_2$ with

$$S_1 = \mathsf{list}(\mathcal{C}_1) = [\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,9}];$$
$$S_2 = \mathsf{list}(\mathcal{C}_2) = [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,4}].$$

We have

$$\Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2) = ([\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,4}], \mathsf{list}(\mathcal{C}_2));$$
$$\Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2) = (\mathsf{list}(\mathcal{C}_1), [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,4}, \mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,4}, \mathrm{R}_{2,1}]).$$

Next, we combine $\Phi_{\min}$ and $\Phi_{\max}$ with CSPL, show that in practical environments $\Phi_{\min}$ and $\Phi_{\max}$ satisfy the requirements put on list-pair functions in Proposition 5.1 to guarantee termination and cluster-sending, and use these results to determine the expected constant complexity of the resulting instances of CSPL.

Theorem 5.7. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters with synchronous communication.*

1. *If $n = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 2\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\min}$) *is upper bounded by* 4*. For every $(S_1, S_2) := \Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2)$, we have $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$, $n > 2\mathbf{f}_{S_1}$, $n > 2\mathbf{f}_{S_2}$, and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$*
2. *If $n = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 3\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\min}$) *is upper bounded by $2\frac{1}{4}$. For every $(S_1, S_2) := \Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2)$, we have $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$, $n > 3\mathbf{f}_{S_1}$, $n > 3\mathbf{f}_{S_2}$, and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$.*
3. *If $\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}$ and $\mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\max}$) *is upper bounded by* 3*. For every $(S_1, S_2) := \Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2)$, we have $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2} = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ and either we have $\mathbf{n}_{\mathcal{C}_1} \geq \mathbf{n}_{\mathcal{C}_2}$, $n > 3\mathbf{f}_{S_1}$, and $n > 2\mathbf{f}_{S_2}$; or we have $\mathbf{n}_{\mathcal{C}_2} \geq \mathbf{n}_{\mathcal{C}_1}$, $n > 2\mathbf{f}_{S_1}$, and $n > 3\mathbf{f}_{S_2}$.*

Each of these instance of CSPL *results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$.*

Proof. First, we prove the properties of $\Phi_{\min}$ and $\Phi_{\max}$ claimed in the three statements of the theorem. In the first and second statement of the theorem, we have $\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > c\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$, $c \in \{2,3\}$. Let $(S_1, S_2) := \Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. By definition of $\Phi_{\min}$, we have $n = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$, in which case S_i, $i \in \{1,2\}$, holds n distinct replicas from $\mathcal{C}_i$. Hence, we have $\mathbf{f}_{\mathcal{C}_i} \geq \mathbf{f}_{S_i}$ and, as $n > c\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2}) \geq c\mathbf{f}_{\mathcal{C}_i}$, also $n > c\mathbf{f}_{S_i}$. Finally, as $n > 2\mathbf{f}_{S_1}$ and $n > 2\mathbf{f}_{S_2}$, also $2n > 2\mathbf{f}_{S_1} + 2\mathbf{f}_{S_2}$ and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ holds.

In the last statement of the theorem, we have $\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}$ and $\mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$. Without loss of generality, we assume $\mathbf{n}_{\mathcal{C}_1} \geq \mathbf{n}_{\mathcal{C}_2}$. Let $(S_1, S_2) := \Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. By definition of $\Phi_{\max}$, we have $n = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) = \mathbf{n}_{\mathcal{C}_1}$. As $n = \mathbf{n}_{\mathcal{C}_1}$, we have $S_1 = \mathsf{list}(\mathcal{C}_1)$. Consequently, we also have $\mathbf{f}_{S_1} = \mathbf{f}_{\mathcal{C}_1}$ and, hence, $\mathbf{n}_{S_1} > 3\mathbf{f}_{\mathcal{C}_1}$. Next, we will show that $\mathbf{n}_{S_2} > 2\mathbf{f}_{S_2}$. Let $q = \mathbf{n}_{\mathcal{C}_1} \operatorname{div} \mathbf{n}_{\mathcal{C}_2}$ and $r = \mathbf{n}_{\mathcal{C}_1} \bmod \mathbf{n}_{\mathcal{C}_2}$. We note that $\mathsf{list}(\mathcal{C}_2)^{:n}$ contains q full copies of $\mathsf{list}(\mathcal{C}_2)$ and one partial copy of $\mathsf{list}(\mathcal{C}_2)$. Let $T \subset \mathcal{C}_2$ be the set of replicas in this partial copy. By construction, we have $\mathbf{n}_{S_2} = q\mathbf{n}_{\mathcal{C}_2} + r > q3\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T + \mathbf{nf}_T$ and $\mathbf{f}_{S_2} = q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T$ with $\mathbf{f}_T \leq \min(\mathbf{f}_{\mathcal{C}_2}, r)$. As $q > 1$ and $\mathbf{f}_{\mathcal{C}_2} \geq \mathbf{f}_T$, we have $q\mathbf{f}_{\mathcal{C}_2} \geq \mathbf{f}_{\mathcal{C}_2} \geq \mathbf{f}_T$. Hence, $\mathbf{n}_{S_2} > 3q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T + \mathbf{nf}_T > 2q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T + \mathbf{nf}_T \geq 2(q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T) + \mathbf{nf}_T \geq 2\mathbf{f}_{S_2}$. Finally, as $n > 3\mathbf{f}_{S_1}$ and $n > 2\mathbf{f}_{S_2}$, also $2n > 3\mathbf{f}_{S_1} + 2\mathbf{f}_{S_2}$ and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ holds.

Now, we prove the upper bounds on the expected number of cluster-sending steps for CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\min}$) with $\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 2\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$. By Proposition 5.3, the expected number of cluster-sending steps is upper bounded by $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$. In the worst case, we have $n = 2f + 1$ with $f = \mathbf{f}_{S_1} = \mathbf{f}_{S_2}$. Hence, the expected number of cluster-sending steps is upper bounded by $\mathbb{E}(2f+1, f, f)$, $f \geq 0$. We claim that $\mathbb{E}(2f+1, f, f)$ simplifies to $\mathbb{E}(2f+1, f, f) = 4 - 2/(f+1) - f!^2/(2f)!$. Hence, for all S_1 and S_2, we have $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2}) < 4$. An analogous argument can be used to prove the other upper bounds. □

Note that the third case of Theorem 5.7 corresponds to cluster-sending between arbitrary-sized resilient clusters that each operate using Byzantine fault-tolerant consensus protocols.

Remark 5.8. The upper bounds on the expected-case complexity of instances of CSPL presented in Theorem 5.7 match the upper bounds for CSP presented in Corollary 4.4. This does not imply that the expected-case complexity for these protocols is the same, however, as the probability distributions that yield these expected-case complexities are very different. To see this, consider a system in which all clusters have n replicas of which f, $n = 2f + 1$, are faulty. Next, we denote the expected number of cluster-sending steps of protocol P by $\mathbf{E}_P$, and we have

$$\mathbf{E}_{\text{CSP}} = \frac{(2f+1)^2}{(f+1)^2} = 4 - \frac{4f+3}{(f+1)^2};$$
$$\mathbf{E}_{\text{CSPL}} = \mathbb{E}(2f+1, f, f) = 4 - \frac{2}{(f+1)} - \frac{f!^2}{(2f)!}.$$

In Figure 6, we have illustrated this difference by plotting the expected-case complexity of CSP and CSPL for systems with equal-sized clusters. In practice, we see that the expected-case complexity for CSP is slightly lower than the expected-case complexity for CSPL.

5.3 Practical Considerations

The results in this paper address the worst-case use-case of cluster-sending: the exchange of a *single* value between clusters in complete isolation without *any* knowledge on the likelihood of specific replicas to be faulty. Practical use-cases typically provide additional knowledge that can be used to further fine-tune the cluster-sending protocols. E.g., if multiple values are to be exchanged in consecutive steps, then one can start the cluster-sending of the *next* value by first attempting to cluster-send via the previously-successful replica pair and by skipping any replica pairs that have failed (in preceding rounds). Likewise, if the likelihood of replicas to be faulty is known to be skewed, then one can incorporate the skew in the *fully at random* selection of replica pairs to maximize the likelihood of selection non-faulty replica pairs.

6 Asynchronous Communication

In the previous sections, we introduced CSP, CSPP, and CSPL, three probabilistic cluster-sending protocols with expected constant message complexity. To simplify presentation, we have presented their design with respect to a synchronous environment. Next, we consider their usage in environments with asynchronous inter-cluster communication due to which messages can get arbitrary delayed, duplicated, or dropped.

We notice that the presented protocols *only* depend on synchronous communication to minimize communication: at the core of the correctness of CSP, CSPP, and CSPL is the cluster-sending step performed by CS-STEP, which does not make any assumptions on communication (Proposition 3.1). Consequently, CSP, CSPP, and CSPL can easily be generalized to operate in environments with asynchronous communication.

First, we observe that message duplication and out-of-order delivery have no impact on the cluster-sending step performed by CS-STEP. Hence, we do not need to take precautions against such asynchronous behavior. Furthermore, if communication is asynchronous, but reliable (messages do not get lost, but can get duplicated, be delivered out-of-order, or get arbitrarily delayed), both CSPP and CSPL will be able to always perform cluster-sending in a finite number of steps.

If communication is asyncrhonous and unreliable (messages sent between non-faulty replicas can get lost and all cluster-sending steps can fail), then the presented *syncrhonous* protocols can fail and, hence, need to be adjusted to the asyncrhonous environment in which they are deployed. The best way in which a probabilistic cluster-sending solution can deal with *unreliable* asynchronous communication depends on the model of asynchronous communication one is optimizing for. If, for example, communication is expected to follow the fair-lossy link model (in which any communication step will succeed infinitely often if performed infinitely often), then replicas in $\mathcal{C}_1$ can simply continue cluster-sending steps until a step succeeds (CSP) or rerun the protocol until a step succeeds (CSPP, and CSPL), which will eventually happen. As a second example, one can consider the partial synchrony model often employed by primary-backup consensus protocols such as PBFT. In these models, *unreliable periods of communication* are followed by *sufficiently-long periods of reliable communication*, during which cluster-sending with the presented algorithms will always succeed. Hence, under this model, running the algorithms until successfull cluster-sending will assure success as soon as communication becomes reliable.

We note that if communication is asynchronous, then messages can get arbitrarily delayed. Fortunately, practical environments operate with large periods of reliable communication in which the majority of the messages arrive within some bounded delay unknown to $\mathcal{C}_1$ and $\mathcal{C}_2$. Hence, replicas in $\mathcal{C}_1$ can simply assume some delay δ. If this delay is too short, then a cluster-sending step can *appear to fail* simply because the proof of receipt is still under way. In this case, cluster-sending will still be achieved when the proof of receipt arrives, but spurious cluster-sending steps can be initiated in the meantime. To reduce the number of such spurious cluster-sending steps, all non-faulty replicas in $\mathcal{C}_1$ can use *exponential backoff* to increase the message delay δ toward some reasonable upper bound (e.g., 100 s).

Finally, asynchronous environments often necessitate rather high assumptions on the message delay δ. Consequently, the duration of a single failed cluster-sending step performed by CS-STEP will be high. Here, a trade-off can be made between *message complexity* and *duration* by starting several rounds of the cluster-sending step at once. E.g.,

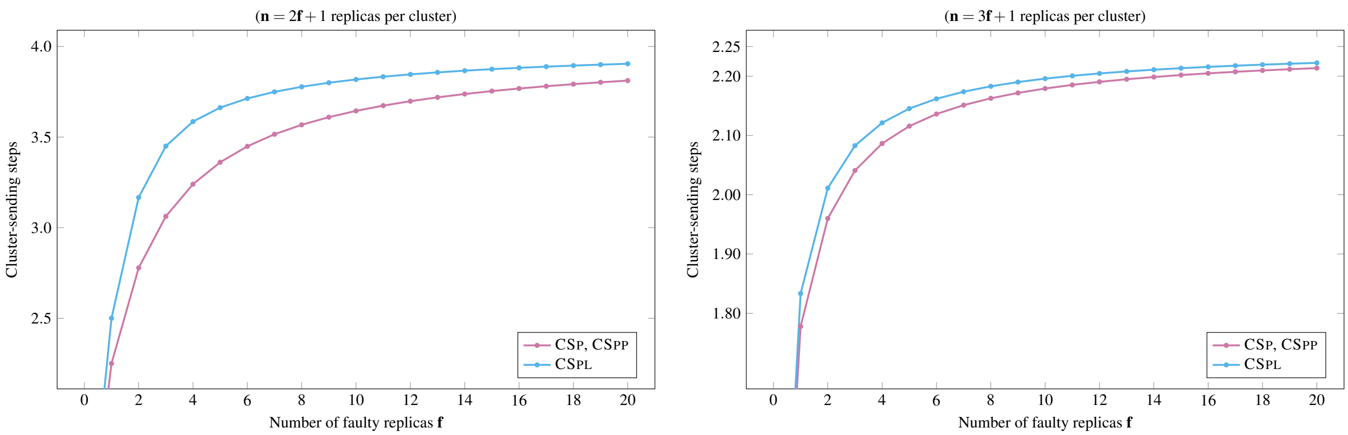

Figure 6: Comparison of the expected-case complexity of CSPL and CSP as a function of the number of faulty replicas.

when communication is sufficiently reliable, then all three protocols are expected to finish in four rounds or less, due to which starting four rounds initially will sharply reduce the duration of the protocol with only a constant increase in expected message complexity.

7 Performance evaluation

In the previous sections, we introduced probabilistic cluster-sending protocols with expected-case constant message complexity. To gain further insight in the performance attainable by these protocols, especially in environments with unreliable communication, we implemented these protocols in a simulated sharded resilient environment that allows us to control the faulty replicas and the message loss rates.[3] As a baseline of comparison, we also evaluated three cluster-sending protocols from the literature:

1. The *worst-case optimal cluster-sending protocol* PBS-CS of Hellings et al. [17, 19] that can perform cluster-sending using only $\mathbf{f}_{C_1} + \mathbf{f}_{C_2} + 1$ messages, which is worst-case optimal. This protocol requires reliable communication.

2. The *broadcast-based cluster-sending protocol* of CHAINSPACE [1] that can perform cluster-sending using $\mathbf{n}_{C_1}\mathbf{n}_{C_2}$ messages. This protocol requires reliable communication.

3. The *global sharing protocol* of GEOBFT [15], an optimistic cluster-sending protocol that assumes that each cluster uses a primary-backup consensus protocol (e.g., PBFT [6]) and optimizes for the case in which the coordinating primary of C_1 is non-faulty. In this optimistic case, GEOBFT can perform cluster-sending using only $\mathbf{f}_{C_2} + 1$ messages. To deal with faulty primaries and unreliable communication, GEOBFT employs a costly remote view-change protocol, however.

[3] The full implementation of this experiment is available at `anonymized`.

We refer to Figure 2 for an analytical comparison between these three cluster-sending protocols and our three probabilistic cluster-sending protocols.

In each experiment, we measured the number of messages exchanged in 10 000 runs of the cluster-sending protocol under consideration. In specific, in each run we measure the number of messages exchanged when sending a value v from a cluster C_1 to a cluster C_2 with $\mathbf{n}_{C_1} = \mathbf{n}_{C_2} = 3\mathbf{f}_{C_1} + 1 = 3\mathbf{f}_{C_2} + 1$, and we aggregate this data over 10 000 runs. The messages exchanged is an objective measure of the performance of the cluster-sending protocols under consideration that is independent of the environment (e.g., network bandwidth, message delays) and the application use-case for which cluster-sending is used. As we use equal-sized clusters, we have $\Phi_{\min}(C_1, C_2) = \Phi_{\max}(C_1, C_2)$ and, hence, we use a singe instance of CSPL.

Next, we detail the two experiments we performed and look at their results.

7.1 Performance of Cluster-Sending Protocols

In our first experiment, we measure the number of messages exchanged as a function of the number of faulty replicas. In this case, we assumed reliable communication, due to which we could include all six protocols. The results of this experiment can be found in Figure 7.

As is clear from the results, our probabilistic cluster-sending protocols are able to perform cluster-sending with only a constant number of messages exchanged. Furthermore, we see that the performance of our cluster-sending protocols matches the theoretical expected-case analysis in this paper and closely follows the expected performance illustrated in Figure 6 (note that Figure 6 plots cluster-sending steps and each cluster-sending step involves the exchange of *two* messages between clusters).

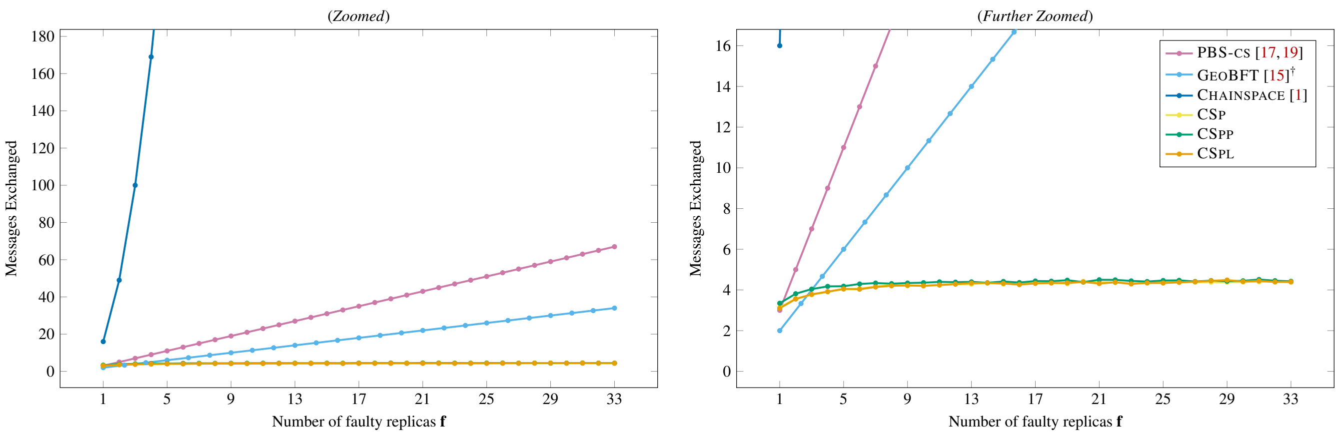

Figure 7: A comparison of the number of message exchange steps as a function of the number of faulty replicas in both clusters by our probabilistic cluster-sending protocols CSP, CSPP, and CSPL, and by three protocols from the literature. For each protocol, we measured the number of message exchange steps to send a value between two equally-sized clusters (average of 10 000 runs), each cluster having $n = 3\mathbf{f}+1$ replicas. †The results for GeoBFT are a plot of the best-case optimistic phase of that protocol.

As all other cluster-sending protocols have a linear (PBS-cs and GeoBFT) or quadratic (Chainspace) message complexity, our probabilistic cluster-sending protocols outperform the other cluster-sending protocols. This is especially the case when dealing with bigger clusters, in which case the expected-case constant message complexity of our probabilistic cluster-sending protocols shows the biggest advantage. Only in the case of the smallest clusters can the other cluster-sending protocols outperform our probabilistic cluster-sending protocols, as PBS-cs, GeoBFT, and Chainspace use reliable communication to their advantage to eliminate any acknowledgment messages send from the receiving cluster to the sending cluster. We believe that the slightly higher cost of our probabilistic cluster-sending protocols in these cases is justified, as our protocols can effectively deal with unreliable communication.

7.2 Message Loss

In our second experiment, we measure the number of messages exchanged as a function of the number of faulty replicas and as a function of the message loss (in percent) *between the two clusters*. We only focus on message loss between clusters, and we assume that consensus steps *within a cluster* always succeed. In this case, we only included our probabilistic cluster-sending protocols, as PBS-cs and Chainspace both assume reliable communication and GeoBFT is only able to perform recovery via remote view-changes in periods of reliable communication. The results of this experiment can be found in Figure 8.

We note that with a message loss of $x\%$, the probability $p(x)$ of a successful cluster-sending step is only $(1-\frac{x}{100})^2$. E.g., $p(30\%) \approx 0.49$. As expected, the message complexity increases with an increase in message loss. Furthermore, the probabilistic cluster-sending protocols perform as expected (when taking into account the added cost to deal with message loss). Although the probabilistic arguments underpinning the expected-case cost of, on the one hand, CSP and CSPP and, on the other hand, CSPL are vastly different, the results of these experiments show that all three protocols behave similarly in practice.

These results further underline the practical benefits of each of the probabilistic cluster-sending protocols, especially for larger clusters: even in the case of high message loss rates, each of our probabilistic cluster-sending protocols are able to outperform the cluster-sending protocols PBS-cs, Chainspace, and GeoBFT, which can only operate with reliable-communication.

8 Related Work

Although there is abundant literature on distributed systems and on consensus-based resilient systems (e.g., [2, 5, 8, 14, 16, 28, 33]), there is only limited work on communication *between* resilient systems [1, 15, 17, 19]. In the previous section, we have already compared CSP, CSPP, and CSPL with the worst-case optimal cluster-sending protocols of Hellings et al. [17, 19], the optimistic cluster-sending protocol of GeoBFT [15], and the broadcast-based cluster-sending protocols of Chainspace [1]. Furthermore, we notice that *cluster-sending* can be solved using well-known Byzantine primitives such as consensus, interactive consistency, and Byzantine broadcasts [6, 9, 25]. These primitives are much more costly than cluster-sending protocols, however, and require huge amounts of communication between all involved replicas.

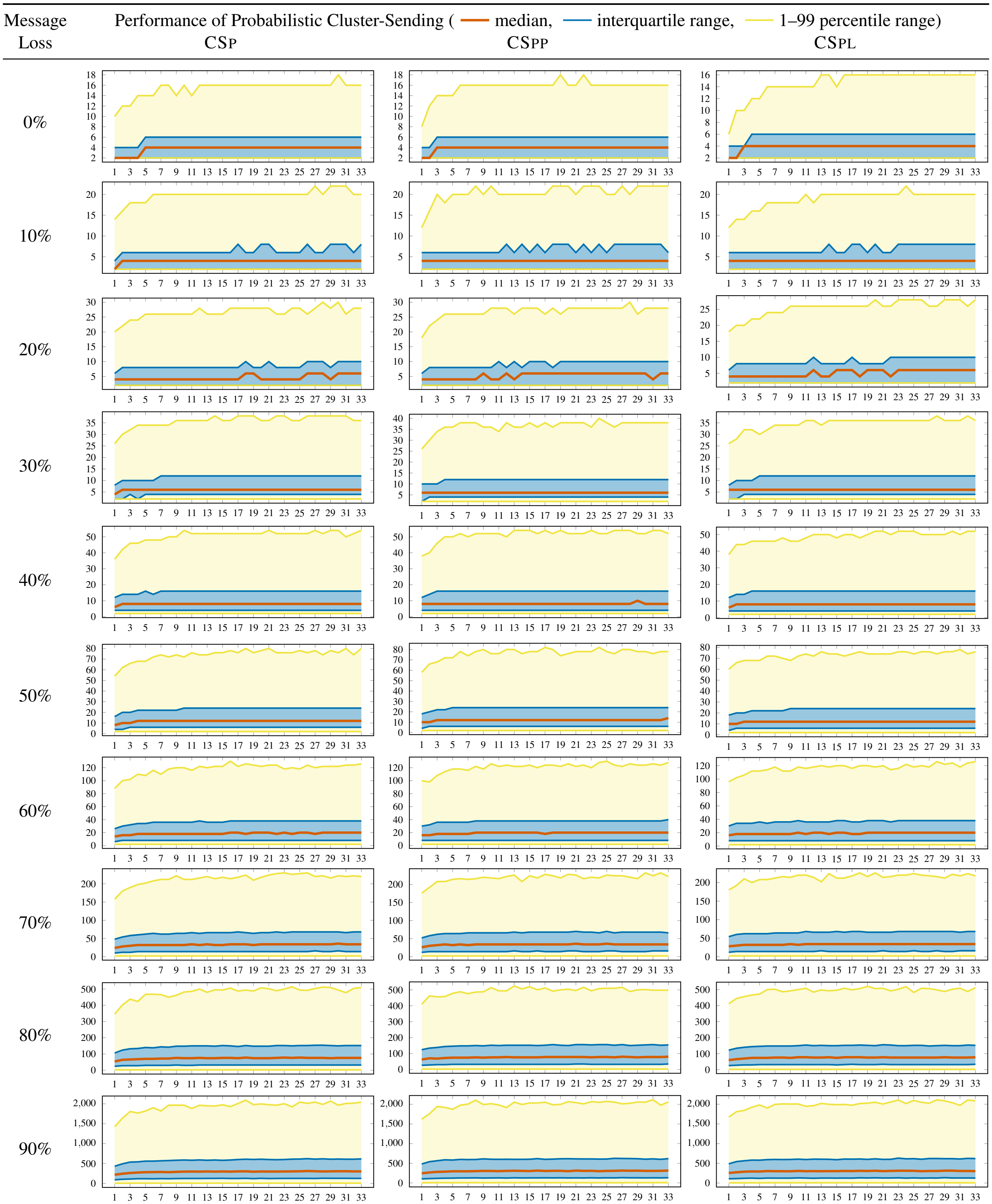

Figure 8: A comparison of the number of message exchange steps as a function of the number of faulty replicas in both clusters and of the message loss by our probabilistic cluster-sending protocols CSP, CSPP, and CSPL. For each protocol, we measured the number of message exchange steps to send 10 000 values between two equally-sized clusters, each cluster having $n = 3\mathbf{f}+1$ replicas, after which we aggregated the measurements to obtain a summary of the distribution of messages exchanged.

In parallel to the development of traditional resilient systems and permissioned blockchains, there has been promising work on sharding in permissionless blockchains such as BITCOIN [26] and ETHEREUM [34]. Examples include techniques for enabling reliable cross-chain coordination via sidechains, blockchain relays, atomic swaps, atomic commitment, and cross-chain deals [12, 13, 20, 22, 23, 35, 36]. Unfortunately, these techniques are deeply intertwined with the design goals of permissionless blockchains in mind (e.g., cryptocurrency-oriented), and are not readily applicable to traditional consensus-based Byzantine clusters.

9 Conclusion

In this paper, we presented probabilistic cluster-sending protocols that each provide highly-efficient solutions to the cluster-sending problem. Our probabilistic cluster-sending protocols can facilitate communication between Byzantine fault-tolerant clusters with expected constant communication between clusters. For practical environments, our protocols can support worst-case linear communication between clusters, which is optimal, and deal with asynchronous and unreliable communication. The low practical cost of our cluster-sending protocols further enables the development and deployment of high-performance systems that are constructed out of Byzantine fault-tolerant clusters, e.g., fault-resilient geo-aware sharded data processing systems.

References

[1] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform, 2017. URL: http://arxiv.org/abs/1708.03778.

[2] Christian Berger and Hans P. Reiser. Scaling byzantine consensus: A broad analysis. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, pages 13–18. ACM, 2018. https://doi.org/10.1145/3284764.3284767.

[3] Gabi Bracha and Ophir Rachman. Randomized consensus in expected $O((n^2 \log n))$ operations. In *Distributed Algorithms*, pages 143–150. Springer Berlin Heidelberg, 1992. https://doi.org/10.1007/BFb0022443.

[4] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology — CRYPTO 2001*, pages 524–541. Springer Berlin Heidelberg, 2001. https://doi.org/10.1007/3-540-44647-8_31.

[5] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild (keynote talk). In *31st International Symposium on Distributed Computing*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. https://doi.org/10.4230/LIPIcs.DISC.2017.1.

[6] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002. https://doi.org/10.1145/571637.571640.

[7] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, pages 123–140. ACM, 2019. https://doi.org/10.1145/3299869.3319889.

[8] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Trans. Knowl. Data Eng.*, 30(7):1366–1385, 2018. https://doi.org/10.1109/TKDE.2017.2781227.

[9] D. Dolev and H. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. https://doi.org/10.1137/0212045.

[10] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77—-97, 1987. https://doi.org/10.1145/7531.7533.

[11] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288—323, 1988. https://doi.org/10.1145/42282.42283.

[12] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. BlockchainDB: A shared database on blockchains. *Proc. VLDB Endow.*, 12(11):1597–1609, 2019. https://doi.org/10.14778/3342263.3342636.

[13] Ethereum Foundation. BTC Relay: A bridge between the bitcoin blockchain & ethereum smart contracts, 2017. URL: http://btcrelay.org.

[14] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. *Fault-Tolerant Distributed Transactions on Blockchain*. Synthesis Lectures on Data Management. Morgan & Claypool, 2021. https://doi.org/10.2200/S01068ED1V01Y202012DTM065.

[15] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. ResilientDB: Global scale resilient

blockchain fabric. *Proceedings of the VLDB Endowment*, 13(6):868—-883, 2020. https://doi.org/10.14778/3380750.3380757.

[16] Suyash Gupta and Mohammad Sadoghi. *Blockchain Transaction Processing*, pages 1–11. Springer International Publishing, 2018. https://doi.org/10.1007/978-3-319-63962-8_333-1.

[17] Jelle Hellings and Mohammad Sadoghi. Brief announcement: The fault-tolerant cluster-sending problem. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:3. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. https://doi.org/10.4230/LIPIcs.DISC.2019.45.

[18] Jelle Hellings and Mohammad Sadoghi. Byshard: Sharding in a byzantine environment. *Proceedings of the VLDB Endowment*, 14(11):2230–2243, 2021. https://doi.org/10.14778/3476249.3476275.

[19] Jelle Hellings and Mohammad Sadoghi. The fault-tolerant cluster-sending problem. In *Foundations of Information and Knowledge Systems*, pages 168–186. Springer, 2022. https://doi.org/10.1007/978-3-031-11321-5_10.

[20] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 245—254. ACM, 2018. https://doi.org/10.1145/3212734.3212736.

[21] Maurice Herlihy. Blockchains from a distributed computing perspective. *Communications of the ACM*, 62(2):78–85, 2019. https://doi.org/10.1145/3209623.

[22] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. Cross-chain deals and adversarial commerce. *The VLDB Journal*, 2021. https://doi.org/10.1007/s00778-021-00686-1.

[23] Jae Kwon and Ethan Buchman. Cosmos whitepaper: A network of distributed ledgers, 2019. URL: https://cosmos.network/cosmos-whitepaper.pdf.

[24] Leslie Lamport. Paxos made simple. *ACM SIGACT News, Distributed Computing Column 5*, 32(4):51–58, 2001. https://doi.org/10.1145/568425.568433.

[25] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. https://doi.org/10.1145/357172.357176.

[26] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. URL: https://bitcoin.org/en/bitcoin-paper.

[27] Arvind Narayanan and Jeremy Clark. Bitcoin's academic pedigree. *Communications of the ACM*, 60(12):36–45, 2017. https://doi.org/10.1145/3132259.

[28] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 2020. https://doi.org/10.1007/978-3-030-26253-2.

[29] Michael Pisa and Matt Juden. Blockchain and economic development: Hype vs. reality. Technical report, Center for Global Development, 2017. URL: https://www.cgdev.org/publication/blockchain-and-economic-development-hype-vs-reality.

[30] David Reinsel, John Gantz, and John Rydning. Data age 2025: The digitization of the world, from edge to core. Technical report, IDC, 2018. URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[31] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400. Springer, 2011. https://doi.org/10.1007/978-3-642-24550-3_29.

[32] Victor Shoup. Practical threshold signatures. In *Advances in Cryptology — EUROCRYPT 2000*, pages 207–220. Springer Berlin Heidelberg, 2000. https://doi.org/10.1007/3-540-45539-6_15.

[33] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2001.

[34] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. EIP-150 revision. URL: https://gavwood.com/paper.pdf.

[35] Gavin Wood. Polkadot: vision for a heterogeneous multi-chain framework, 2016. URL: https://polkadot.network/PolkaDotPaper.pdf.

[36] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. Atomic commitment across blockchains. *Proc. VLDB Endow.*, 13(9):1319–1331, 2020. https://doi.org/10.14778/3397230.3397231.

A The proof of Lemma 5.2

To get the intuition behind the closed form of Lemma 5.2, we take a quick look at the combinatorics of *list-merging*. Notice that we can merge lists S and T together by either first taking an element from S or first taking an element from T. This approach towards list-merging yields the following recursive solution to the list-merge problem:

$$\mathbb{M}(v,w) = \begin{cases} \mathbb{M}(v-1,w) + \mathbb{M}(v,w-1) & \text{if } v > 0 \text{ and } w > 0; \\ 1 & \text{if } v = 0 \text{ or } w = 0. \end{cases}$$

Consider lists S and T with $|S| = v$ and $|T| = w$ distinct values. We have $|\mathsf{perms}(S)| = v!$, $|\mathsf{perms}(T)| = w!$, and $|\mathsf{perms}(S \cup T)| = (v+w)!$. We observe that every list-merge of $(P_S, P_T) \in \mathsf{perms}(S) \times \mathsf{perms}(T)$ is a unique value in $\mathsf{perms}(S \cup T)$. Furthermore, every value in $\mathsf{perms}(S \cup T)$ can be constructed by such a list-merge. As we have $|\mathsf{perms}(S) \times \mathsf{perms}(T)| = v!w!$, we derive the closed form

$$\mathbb{M}(v,w) = \frac{(v+w)!}{(v!w!)}$$

of Lemma 5.2. Next, we formally prove this closed form.

Proof. We prove this by induction. First, the base cases $\mathbb{M}(0,w)$ and $\mathbb{M}(v,0)$. We have

$$\mathbb{M}(0,w) = \frac{(0+w)!}{0!w!} = \frac{w!}{w!} = 1;$$
$$\mathbb{M}(v,0) = \frac{(v+0)!}{v!0!} = \frac{v!}{v!} = 1.$$

Next, we assume that the statement of the lemma holds for all non-negative integers v', w' with $0 \leq v' + w' \leq j$. Now consider non-negative integers v, w with $v + w = j + 1$. We assume that $v > 0$ and $w > 0$, as otherwise one of the base cases applies. Hence, we have

$$\mathbb{M}(v,w) = \mathbb{M}(v-1,w) + \mathbb{M}(v,w-1).$$

We apply the induction hypothesis on the terms $\mathbb{M}(v-1,w)$ and $\mathbb{M}(v,w-1)$ and obtain

$$\mathbb{M}(v,w) = \left(\frac{((v-1)+w)!}{(v-1)!w!}\right) + \left(\frac{(v+(w-1))!}{v!(w-1)!}\right).$$

Next, we apply $x = x(x-1)!$ and simplify the result to obtain

$$\mathbb{M}(v,w) = \left(\frac{v(v+w-1)!}{v!w!}\right) + \left(\frac{w(v+w-1)!}{v!w!}\right)$$
$$= \left(\frac{(v+w)(v+w-1)!}{v!w!}\right) = \frac{(v+w)!}{v!w!},$$

which completes the proof. □

B The simplification of $\mathbb{F}(n, m_1, m_2, k)$

Let g be the expression

$$b_1!^2 \binom{m_1}{b_1}\binom{n-m_2}{b_1} b_2!^2 \binom{n-m_1}{b_2}\binom{m_2}{b_2} \cdot$$
$$\mathbb{M}(b_1,b_2) b_{1,2}!^2 \mathbb{M}(b_1+b_2, b_{1,2})(n-k)!^2 \mathbb{M}(k, n-k),$$

as stated right above Lemma 5.3. We will show that g is equivalent to the closed form of $\mathbb{F}(n, m_1, m_2, k)$, as stated in Lemma 5.3.

Proof. We use the shorthands $\mathbf{T}_1 = \binom{m_1}{b_1}\binom{n-m_2}{b_1}$ and $\mathbf{T}_2 = \binom{n-m_1}{b_2}\binom{m_2}{b_2}$, and we have

$$g = b_1!^2 \mathbf{T}_1 b_2!^2 \mathbf{T}_2 \cdot$$
$$\mathbb{M}(b_1,b_2) b_{1,2}!^2 \mathbb{M}(b_1+b_2, b_{1,2})(n-k)!^2 \mathbb{M}(k, n-k).$$

We apply Lemma 5.2 on terms $\mathbb{M}(b_1,b_2)$, $\mathbb{M}(b_1+b_2,b_{1,2})$, and $\mathbb{M}(k,n-k)$, apply $k = b_1 + b_2 + b_{1,2}$, and simplify to derive

$$g = b_1!^2 \mathbf{T}_1 b_2!^2 \mathbf{T}_2 \cdot$$
$$\frac{(b_1+b_2)!}{b_1!b_2!} b_{1,2}!^2 \frac{(b_1+b_2+b_{1,2})!}{(b_1+b_2)!b_{1,2}!} (n-k)!^2 \frac{(k+n-k)!}{k!(n-k)!}$$
$$= b_1! \mathbf{T}_1 b_2! \mathbf{T}_2 b_{1,2}!(n-k)!n!.$$

Finally, we expand the binomial terms $\mathbf{T}_1$ and $\mathbf{T}_2$, apply $b_{1,2} = m_1 - b_1 = m_2 - b_2$ and $k = m_1 + b_2 = m_2 + b_1$, and simplify to derive

$$g = b_1! \frac{m_1!}{b_1!(m_1-b_1)!} \frac{(n-m_2)!}{b_1!(n-m_2-b_1)!} \cdot$$
$$b_2! \frac{(n-m_1)!}{b_2!(n-m_1-b_2)!} \frac{m_2!}{b_2!(m_2-b_2)!} \cdot$$
$$b_{1,2}!(n-k)!n!$$
$$= \frac{m_1!}{b_{1,2}!} \frac{(n-m_2)!}{b_1!(n-k)!} \frac{(n-m_1)!}{b_2!(n-k)!} \frac{m_2!}{b_{1,2}!} b_{1,2}!(n-k)!n!$$
$$= \frac{m_1!m_2!(n-m_1)!(n-m_2)!n!}{b_1!b_2!b_{1,2}!(n-k)!},$$

which completes the proof. □

C The Closed Form of $\mathbb{E}(2f+1, f, f)$

Here, we shall prove that

$$\mathbb{E}(2f+1, f, f) = 4 - \frac{2}{(f+1)} - \frac{f!^2}{(2f)!}.$$

Proof. By Proposition 5.2 and some simplifications, we have

$$\mathbb{E}(2f+1,f,f) = \frac{1}{(2f+1)!^2} \cdot \left(\sum_{k=f}^{2f} \frac{2f+1}{2f+1-k} \frac{f!^2(f+1)!^2(2f+1)!}{(k-f)!^2(2f-k)!(2f+1-k)!}\right).$$

First, we apply $x! = x(x-1)!$, simplify, and obtain

$$\begin{aligned}\mathbb{E}(2f+1,f,f) &= \frac{f!^2(2f+1)}{(2f+1)!} \cdot \left(\sum_{k=f}^{2f} \frac{(f+1)!^2}{(k-f)!^2(2f+1-k)!^2}\right)\\ &= \frac{f!^2}{(2f)!}\left(\sum_{k=0}^{f} \frac{(f+1)!^2}{k!^2(f+1-k)!^2}\right)\\ &= \frac{f!^2}{(2f)!}\left(\sum_{k=0}^{f} \binom{f+1}{k}^2\right).\end{aligned}$$

Next, we apply $\binom{m}{n} = \binom{m}{m-n}$, extend the sum by one term, and obtain

$$\mathbb{E}(2f+1,f,f) = \frac{f!^2}{(2f)!} \cdot \left(\left(\sum_{k=0}^{f+1} \binom{f+1}{k}\binom{f+1}{f+1-k}\right) - \binom{f+1}{f+1}\binom{f+1}{0}\right).$$

Then, we apply Vandermonde's Identity to eliminate the sum and obtain

$$\mathbb{E}(2f+1,f,f) = \frac{f!^2}{(2f)!}\left(\binom{2f+2}{f+1} - 1\right).$$

Finally, we apply straightforward simplifications and obtain

$$\begin{aligned}\mathbb{E}(2f+1,f,f) &= \frac{f!^2}{(2f)!}\frac{(2f+2)!}{(f+1)!(f+1)!} - \frac{f!^2}{(2f)!}\\ &= \frac{f!^2}{(2f)!}\frac{(2f)!(2f+1)(2f+2)}{f!^2(f+1)^2} - \frac{f!^2}{(2f)!}\\ &= \frac{(2f+1)(2f+2)}{(f+1)^2} - \frac{f!^2}{(2f)!}\\ &= \frac{(2f+2)^2}{(f+1)^2} - \frac{2f+2}{(f+1)^2} - \frac{f!^2}{(2f)!}\\ &= \frac{4(f+1)^2}{(f+1)^2} - \frac{2(f+1)}{(f+1)^2} - \frac{f!^2}{(2f)!}\\ &= 4 - \frac{2}{f+1} - \frac{f!^2}{(2f)!},\end{aligned}$$

which completes the proof. □

Supplemental Materials

See attached the rebuttal comments based on the previous reviews. We have applied the changes proposed in these rebuttal comments in the revised paper.

Response to the Reviewers' comments
"Solution: Byzantine Cluster-Sending in Expected Constant Cost and Constant Time"

Anonymous authors

October 1, 2022

We like to thank each of the three reviewers for their careful review and their detailed feedback of our previous manuscript and the very positive evaluation of the accompanying artifact. To address the reviewers comments, we have detailed below our rebuttal to the reviewers comments and the corresponding revisions (in red).

Review #1 (reviewer HnvD) Besides the following major comments, the reviewer provided several minor comments (e.g., spelling) that we shall resolve in a revision.

> **Review:** When considering the sharded design to overcome limitations on consensus scalability it might be relevant to also consider that consensus might not be needed to quantity transfer tasks: cf: `https://hal.archives-ouvertes.fr/hal-02861511v3`.

To the best of our knowledge, strict ordering is necessary to support arbitrary general-purpose workloads. However, we agree that there are indeed classes of operations for which strict consensus-based ordering of (sharding) steps is unnecessary (e.g., balance changes and, more generally, operations on CRDTs). Hence, we can add a provision to recognize such cases to the introduction.

> **Revise:** Include design alternatives for scalable Byzantine fault-tolerant systems that do not require strict ordering in the Introduction.

> **Review:** In the CSpl protocol, it looks that a possible optimization would order the lists of pairs, instead of fully at random, by the likelihood of them having non faulty replicas. Apparently this could favour from transfer to transfer trying first pairs that are more likely to succeed by being successful in previous transfers. Would there be any obvious drawback in this?

Such optimizations are only possible if information on the likelihood of failure of individual replicas is available to all replicas in the sending cluster and these replicas all agree exactly on this information: otherwise, individual replicas might make distinct decisions on how to proceed with cluster-sending.

In the specific case of resending via previously-successful paths (as a first attempt): this seems like an obvious further optimization for repeated cluster-sending steps, as the information on previously-successful paths will be available to all replicas.

> **Revise:** We will add a *practical consideration* section (which will also addresses some of the comments of Reviewer #2) where the multiple send optimization is outlined.

> **Review:** On section 6, point 2, talking about eventual occurrence of reliable communication. The concept of "Fair-lossy link" can be useful here. cf: `https://dcl.epfl.ch/site/_media/education/da18-introduction.pdf`

In the text, we have opted for the informal model of asynchronous communication that is commonly used in PBFT-like algorithms. We did not choose a particular formalization, as we believe that our protocols can be adapted to any *practical* formalization of unreliable communication.

> **Review:** Figure 7 would be more clear by using in the y axis the average number of steps, instead of the total number of steps for 10000 repetitions.

> **Revise:** We agree with this comment and will update Figure 7 and Figure 8, *top*, to remove the factor 10^4 from the y-axis. We will update the captions accordingly.

> **Review:** "We assume that communication within each cluster is reliable. In this case, we only included our probabilistic cluster-sending protocols as PBS-CS and CHAINSPACE both assume reliable communication ..". By reliable did you meant to say unreliable here?

Both occurrences of *reliable* in the above excerpt are correct. In the experiment, we assume that communication within a cluster is reliable, as we only focus on failure to communicate between clusters. Hence, consensus steps within a cluster always succeed, while CS-STEP might fail due to communication failures.

Review #2 (reviewer 8v6c) The reviewer provided a non-exhaustive list of writing comments at `https://ipfs.infura.io/ipfs/QmUBLLUDfmgkkM9Yn99Vyk5K4NfLiqj68YLvPKSPXszVya`. We thank the reviewer for the detailed feedback and shall resolve these comments in a revision. Besides these comments, the reviewer provided the following major comments.

> **Review:** It is a bit questionable to me how practically relevant the chosen model of synchronous rounds is in practice, especially when the motivation seems to be blockchain systems and other large-scale geo-distributed systems, where communication in synchronous rounds is generally hard to achieve.

We agree that an always synchronous system is near-impossible to achieve in practical large-scale settings. At the same time, practical communication (e.g., via the internet) between any pair of replicas will work reliably with a reasonable delay with high likelihood.

As such, we have chosen to use a synchronous model to present the main ideas of probabilistic cluster-sending protocols, which allowed us to simplify presentation significantly. As argued in our paper, the probabilistic cluster-sending protocols can easily be extended to an asynchronous (practical) environment to deal with the cases in which communication is not reliable or does not have a reasonable delay.

> **Revise:** We will further clarify the structure of the paper with respect to asynchronous and syncrhonous communication, which is already mentioned in Section 4, in the Introduction.

> **Review:** Together with completely neglecting the intra-cluster overhead, this creates a very specific model whose relation to practice could have been better argumented.
>
> ...
>
> Also, at the beginning of page 5, I did not properly understand the implicit / free consensus. Indeed, nodes in a cluster first need to agree on some operation that triggers the cluster-sending, and thus agreeing on a message can be implicit. However, I do not see how, at the same time, agreeing on the reception of a value can also be considered "free". In such a case, it seems to me, the whole cluster-sending could be implicit and no algorithm would be necessary in the first place.

There are several cases in which intra-cluster overhead can be ignored. One such case is already mentioned in the paper: in the case in which clusters are geo-aware shards (local clusters, dispersed globally); the costs of intra-cluster steps are orders-of-magnitude lower than inter-cluster steps (see, e.g., [15]).

More generally, in most use cases we see for cluster-sending (e.g., multi-shard transaction execution), the intra-cluster *consensus* steps are implicit, as they are also required to execute the transaction itself (which we already briefly hinted at in Remark 4.3). To see this, consider a multi-shard transaction T that affects clusters C_1 and C_2. One way to execute this transaction in a *serializable* manner is (1) for C_1 to *lock* all data relevant for the transaction and determine which operations σ need to be performed by C_2; (2) for C_1 to transfer control to C_2; (3) for C_2 to perform σ; (4) for C_2 to transfer control back to C_1; and (5) for C_1 to make any required local changes and unlock all locked data. In this setting, a single cluster-sending step is required and all consensus steps of this cluster-sending step can be made implicit:

- For step (1), cluster C_1 needs to reach consensus on when to start execution of T (when to lock all data relevant, when to determine σ). As part of this consensus step, one can construct the `send`-message of Line 1 of CSP/CSPL.
- Likewise, for step (3), cluster C_2 needs to reach consensus on when to perform σ. The proof of this consensus step can be used as the proof of Line 4 of CS-STEP.
- Finally, for step (5), cluster C_1 needs to reach consensus on when to complete execution of T (when to make any local changes and unlock data). The proof of this consensus step can be used as the proof of Line 9 of CS-STEP.

We believe that such integration of the intra-cluster overhead of cluster-sending into the underlying task that needs to be performed (in this case, a multi-shard transaction execution) is typically possible, although the specifics on how to do so will vary greatly from use case to use case. Hence, as we believe the intra-cluster overhead is non-essential, we have not studied these costs in-depth.

> **Revise:** We will add a *practical consideration* section (which also addresses some of the comments of Reviewer #1) where the above practical approach toward eliminating additional intra-cluster consensus steps for the purpose of cluster-sending are discussed. This added section will expand on the current Remark 4.3.

> **Review:** More than number of cross-cluster messages exchanged, it would be interesting to also study the expected / worst-case latency in of cluster-sending in a realistic system.
>
> ...
>
> The evaluation is rather limited and only focuses on the number of messages sent as a function of cluster size. What would be much more interesting for me to see are actual time and bandwidth spent to complete a cluster-sending operation.

The messages exchanged (and the number of rounds of CS-STEP) is an objective measure that is independent of the environment and the application use case for which cluster-sending is used. Any measure of bandwidth and latency in a realistic system would be hugely dependent on deployment parameters. E.g., the type of values that need to be cluster-send (which depend on the use case), the consensus protocols used locally, any local execution cost (due to execution of transaction steps), and the cost of cryptographic libraries used. Hence, we believe that any bandwidth and latency measurements would provide only insight on the specifics of the artifact, and no further insight on these costs in any other use cases and deployments.

> **Revise:** We will add the above rational to Section 7 (in specific, right before Section 7.1).

> **Review:** Also, the authors clearly state that they consider a Byzantine model, but at the same time suggest Paxos, a CFT protocol, as an example of a protocol for intra-cluster agreement.

We mainly consider a Byzantine environment. This does not rule out deployments where:

1. resilience to Byzantine failures is not required at all and we only have to consider crashes;

2. resilience to Byzantine failures is not required at the level of individual clusters (e.g., tightly managed environments), while communication between clusters is done via an inherently hostile Byzantine channel (e.g., a public network such as the internet).

In both cases, Paxos (crash-fault tolerance) suffices at the level of clusters. Crash-fault tolerance can deal with a different ratio of faulty and non-faulty replicas than typical Byzantine fault-tolerant systems (without trusted hardware). Hence, the probabilities involved in probabilistic cluster-sending are rather different in case of crash-fault tolerant than in the case of Byzantine-fault tolerance. As we believe the crash-fault tolerant cases to be useful in their own right, we have also included results for them.

> **Revise:** Currently, the consideration of crash-fault tolerant deployments is only hinted at indirectly (e.g., by mentioning Paxos), while the expected case complexities for such deployments are included in the results of the paper. We will make this consideration explicit.

> **Review:** Although some parts of the paper make use of rather extensive formalism, the very definition of the cluster-sending problem is not very clear. Although Definition 2.1 conveys a reasonable intuition behind the problem, in itself it is rather vague. E.g., the terms RECEIVE, CONFIRM, and AGREE are not properly defined and even later on in the paper their exact definitions are not properly specified. I would suggest defining proper abstractions to model "local consensus" and communication between nodes / clusters and expressing the cluster-sending problem in terms of those.

We agree with the reviewer that current definition are on the intuitive side. We did so as we did not want to tie the definition of cluster-sending to our particular solution. E.g., a formalization in consensus and communication steps would clearly fit the steps of the protocols we propose, while excluding solutions to the cluster-sending problem that either do not require consensus steps at the receiving cluster or do not require acknowledgment messages sent by the receiving cluster. We note that such solutions should not be ruled out, as they do exist in environments with fully reliable communication.

> **Revise:** We can formalize the terms agree, receive, and confirm as *states* of the involved non-faulty replicas and require that all involved non-faulty replicas make the same state changes independent of Byzantine behavior. Such a formalization would apply to both our solution and other possible solutions.

> **Review:** I was a bit puzzled by the claims around optimality referring to [17], as it is a 3-page brief announcement that presents high-level intuitions about the cluster-sendig problem, but does not prove any lower bounds on its solution.

The work in [17] did claim the worst-case lower bounds (Theorem 2 and 3 in that work), but did not provide proofs for these claims. The details of the proof were included in the technical report available at `https://arxiv.org/abs/1908.01455`.

> **Revise:** We will update the references accordingly.

> **Review:** When talking about the complexity of the proposed algorithms, I did not fully understand how the number of "agreements" was counted, especially in the case where one of the selected nodes is faulty. In particular, is it possible for nodes to participate in an agreement protocol that fails due to a faulty proposer?

We believe that this comment refers to the number of consensus steps performed as part of the protocols (e.g., in Theorem 4.2, we state two local consensus steps in cluster C_1 and one local consensus step in cluster C_2). In this case, a *consensus step* is a completed round of consensus due to which all non-faulty replicas reach agreement on a consensus decision. In a blockchain, this would correspond to a block being added to their ledger.

We will further illustrate this using the consensus steps such as Line 8 of CS-step. With regards to this consensus step, we have three cases:

1. If the replica R_2 that received m and needs the construction of a proof at Line 8 is non-faulty, then it will always be able to assure a consensus step. In specific, we require that a consensus step includes any mitigation steps of the consensus protocol used to deal with faulty participants. E.g., if PBFT is used and the current primary P of the receiving cluster is faulty, then the replica R_2 that received m and needs the construction of a proof at Line 8 needs to force a view-change (which can be done by forwarding m to all non-faulty replicas in C_2, which then each can ask primary P to initiate a successful consensus step, failure of which will lead into a view-change).

2. If the replica R_2 that received m and needs the construction of a proof at Line 8 is faulty, then it might not initiate a consensus step. In this case, the CS-STEP will fail and a next iteration of CS-STEP will eventually be triggered.

3. If the replica R_2 that received m and needs the construction of a proof at Line 8 is faulty, then it can initiate a consensus step, leading to a proof of receipt m_p, but not send m_p back to cluster C_1. In this case, the CS-STEP will fail and a next iteration of CS-STEP will eventually be triggered. If the receiving replica of the next iteration is non-faulty, then it participated in the earlier round of consensus that resulted in m_p and, hence, this non-faulty replica can skip the consensus and send m_p back to cluster C_1.

Hence, in all cases, non-faulty receivers can force consensus steps in C_2 when necessary, while only a single consensus step in C_2 is required.

Revise: We will state explicitly what we mean by *consensus steps* in Theorem 4.2 and similar results. We will further clarify the second paragraph of the proof of Theorem 4.2 by incorporating a full case breakdown.

Review: I like the progressive approach to explaining the protocols and their analyses, starting simple and gradually moving to wards the more complex ones. Some of the proofs / analyses were rather hard to read though. A more structured approach, breaking them down into smaller parts that are easier to digest would have been welcome.

As the reviewer noted, we already work step-by-step in our write up to eventually arrive at the full details of CSPL. This structure is currently not yet explicitly outlined in the paper, however. We believe that making the structure of the following arguments explicit (especially in Section 5) will support and guide the reader throughout the paper.

Revise: Add supporting text to the organization outlined in the Introduction and to Sections 3–5 to support and guide the reader throughout the paper. Furthermore, integrate Appendix A into Section 5.

Review #3 (reviewer ZfrY)

Review: First, the solution itself is not very novel. The basic building block, CS-Step, is essentially a two-phase commit protocol between two clusters.

We agree with the reviewer that the basic building blocks in cluster-sending are *simple*, which is unsurprising as cluster-sending itself is a simple primitive (when compared to other Byzantine fault-tolerant primitives such as consensus). Indeed, the analogue of cluster-sending in a normal (non-fault tolerant) distributed system in which individual components are nodes (instead of fault-tolerant clusters) would be *reliable message passing*, which would be straightforward to implement on top of TCP. As such, we disagree with the comparison with two-phase commit: two-phase commit aims at solving a higher-level agreement problem (much more akin to consensus than cluster-sending).

Indeed in the fault-tolerant area several recent sharded fault-tolerant system designs such as AHL [7], ByShard [18], and Chainspace [1] have built multi-shard transaction execution capabilities on top of Byzantine fault-tolerant variants of *two-phase commit*. Each of these system designs use consensus and cluster-sending (or, informally, a form of inter-cluster communication) as building blocks to build these Byzantine fault-tolerant 2PCs.

Even though we agree that cluster-sending is a simple problem, we also believe it is a *fundamental primitive* for scalable resilient systems. This, we strongly believe cluster-sending warrants investigation: recent works, including the aforementioned AHL, ByShard, and Chainspace, have shown that some form of inter-cluster communication, which can be formalized as cluster-sending, is an essential building block (together with consensus for intra-cluster decision making).

> **Review:** Then, the extensions, CSp and CSpl, are just randomized versions of the basic algorithm to select nodes, which is a common process in Byzantine consensus and agreement algorithms. There may be some interesting insights in the paper, but as presented, it was not clear.

We agree with the reviewer that the protocols of Section 4, CSP and CSPP, are straightforward randomized protocols. In specific, in CSP, we randomly select replica pairs until we pick a pair of non-faulty replicas, whereas the protocol CSPP provides a fine-tuned version of CSP that eliminates provable unnecessary steps. Although both protocols can be used in practice and have very low expected-case complexities, we do not consider them to be a major contribution in this paper. Instead, the main purpose of Section 4 (and the protocols CSP and CSPP) is three-fold:

1. illustrate that cluster-sending with an expected constant cost is *possible* (even if the worst-case lower-bounds of [17] dictate linear costs);
2. illustrate the basic usage of CS-STEP in randomized protocols; and
3. illustrate the basic operations of randomized protocols, their complexity analysis, and ways to fine-tune them.

Unfortunately, both CSP and CSPP have very high worst-case complexity: CSP does not guarantee termination, whereas CSPP performs a worst-case quadratic amount of steps (as argued in Corollary 4.5).

Section 4 introduces all main concepts we rely on in later sections to *improve* on these straightforward protocols by constructing the randomized protocol CSPL. CSPL is able to combine an *expected constant cost* with a low worst-case optimal cost. We believe that the main *novelty* of the paper is CSPL, and the main complexity of the paper is the analysis of the probabilistic experiment at the basis of the expected-case analysis of this protocol (which covers Section 5).

> **Revise:** We will rework the organization part of the Introduction and make changes to Sections 3–5 to better highlight our contributions and support and guide the reader throughout the paper.

> **Review:** If not for the novelty of the algorithm itself, I think the analysis of the algorithm should be interesting and/or provide new insights. The authors indeed carefully analyze a closed form solution of probability of non-faulty position. This, for example, yields the bounds of 4 expected steps and $\frac{9}{4}$ expected steps in cases 1 and 2 of Theorem 5.7. This analysis, however, seems unnecessarily complex. **When considering $\Phi_{\min}$ in the first case of Theorem 5.7, the analysis is simply about the number of trials before selecting two good nodes without replacement. This is upper bounded by the same experiment but with replacement, since a failed node is removed when considering without replacement. This is upper bounded by the same experiment but with replacement, since a failed node is removed when considering without replacement. WLOG, assume $m_1 > m_2$. In this case, the probability of success is**

$$\begin{aligned}
&\Pr[\textbf{Selecting two good nodes without replacement}]\\
&\qquad > \Pr[\textbf{Selecting two good nodes with replacement}]\\
&\qquad > \left(1 - \frac{m_1}{n}\right)\left(1 - \frac{m_2}{n}\right)\\
&\qquad > \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{2}\right)\\
&\qquad = \frac{1}{4}
\end{aligned}$$

The "with replacement" case becomes a simple Bernoulli trial with $p > \frac{1}{4}$, which then has expected number of trial of $< \frac{1}{p}$. This provides the same bound of 4 steps, with a much simpler analysis. [emphasis added by the authors] Similar analysis for case 2 of Theorem 5.7 as well, where p is bounded below by $(1-\frac{1}{3})(1-\frac{1}{3}) = \frac{4}{9}$, and thus expected number of trials less than $\frac{9}{4}$, which again is the same bound as the analysis gives in the paper. Similar analysis can be applied to $\Phi_{\max}$ for the same result as case 3. Would it be possible to get a tighter expected value than this loose analysis?

The probabilistic argument described by the reviewer does not match the random experiment of CSpl. In specific, the core simplification on which the above argument rests is the statement

$$\Pr[\text{Selecting two good nodes with replacement}] > \left(1 - \frac{m_1}{n}\right)\left(1 - \frac{m_2}{n}\right).$$

We note that the above statement combines the probabilities $p_1 = \left(1 - \frac{m_1}{n}\right)$ (probability to pick a non-faulty sender from cluster C_1) and $p_2 = \left(1 - \frac{m_2}{n}\right)$ (probability to pick a non-faulty receiver from cluster C_2.

This simplified statement does not hold in the random experiment performed by CSpl: to combine probabilites p_1 and p_2 in this way, one requires that the two probabilites are *independent*, which is not the case in CSpl. Indeed, in CSpl a replica pair is picked from a pre-arranged set of pairs (that does not include all possible pairs, hence, making the pairing dependent).

The analysis of the reviewer does apply to CSp, however, as CSp performs a straightforward *independent* pick of replicas from the sending and receiving cluster. As such, the analysis presented in Theorem 4.2 is almost identical to the analysis proposed here by the reviewer.

Although CSp has the simple analysis proposed by the reviewer, this simple analysis can only exists because CSp inspects all $n_1 \cdot n_2$ possible pairings of n_1 replicas in the sending cluster and n_2 replicas in the receiving cluster (thereby guaranteeing that replicas in the sending and receiving cluster are picked *independently*). The downside of this approach followed by CSp is that it has to consider all possible pairing, which is a *quadratic amount* even *if* we do not replace pairs.

The *main contribution* of our paper is an alternative to CSp, the protocol CSpl, that guarantees to inspect only a worst-case *linear amount* of replica pairs (which matches the worst-case lower-bounds on complexity as proven in [17]), while at the same time guaranteeing a constant expected-case complexity. To be able to achieve these two guarantees in one protocol, CSpl performs a random experiment in which the choice of replica from the sending cluster *depends* on the choice of replica from the receiving cluster (and vice versa).

Even though this experiment considers strictly fewer replicas pairs than a standard random pick of replicas (as performed by CSp), we were able via an in-depth analysis (which is the majority of Section 5) to derive that the expected-case cost of both approaches are *similar* and provide the same *constant upper-bounds on the expected-case costs of* CSp *and* CSpl (they are not identical, however, as argued in Remark 5.8).

Revise: To address the above reviewer comment and the previous reviewer comment, we will further underline our contributions and their significance in the Introduction and Section 5 (the latter focussing on the differences in the random experiments of CSp and CSpl).

Review: The evaluation also needs improvement. For instance, while the expected number of steps is important, the tail latency is also important. For example, what is the 99% or 99.9% messaging latency for various sizes of clusters and number of faulty nodes?

Figure 8, *bottom*, already provides the complexity of the worst run (out of 10000) with respect to total number of CS-steps, which translate exactly to the number of communication phases and the latency. We can extend the experiments to also include further distribution data on the complexity of the runs.

Revise: Include further distribution data on the tail latency of the message complexity in the 10.000 runs performed in the experiments.

Review: This might also be a place where an exact closed form analysis of the probability might be beneficial, as you can analyze the variance and other properties of the distribution more theoretically, not just empirically.

The protocol CSP follows the standard geometric distribution. All other protocols follow non-standard distributions.

Revise: We can extend the claims of Theorem 4.2 with other theoretical details on the cost of CSP. For the other protocols, we do not have a standard distribution with known theoretical standard deviations and variance. We can, however, extend the experiments to include details on standard deviations in the cost of the protocols.

Review: In general, the evaluation also probably could benefit from showing more realistic results. E.g., running two small clusters as shown in Figure 1 (like, one in Europe and one in US), and measuring end to end latency to run the full protocol.

As detailed in the response to Reviewer #2, we used the messages exchanged (and the number of rounds of CS-STEP) as an objective measure that is independent of the environment and the application use case for which cluster-sending is used.

Any measure of bandwidth and latency in a realistic system would be hugely dependent on deployment parameters. E.g., the type of values that need to be cluster-send (which depend on the use case), the consensus protocols used locally, any local execution cost (due to execution of transaction steps), and the cost of cryptographic libraries used. Hence, we believe that any bandwidth and latency measurements would provide only insight on the specifics of the artifact, and no further insight on these costs in any other use cases and deployments.

Revise: We will add the above rational to Section 7 (in specific, right before Section 7.1).

Review: How does one perform "Choose replicas" in step 4 of CSp? This step also seem to require a Byzantine consensus, or a reliable (random) beacon. It's possible to bootstrap this with a random initial seed, and using a PRNG from that seed to locally determine the randomness for that round, but I believe it requires some care.

The protocols require a shared source of randomness for all replicas in the sending cluster. As stated in the paper (end of Section 2), such randomness can be provided via a distributed fault-tolerant random coin [3,4].

Review: The protocol doesn't seem to generalize that well to multiple clusters. As in, if a cluster needs to send values to clusters and all clusters need to be sure that all clusters saw the value, then the number of inter-cluster messages seems to scale quadratically in with naive extension of this scheme.

Indeed, the protocol is desingned for cluster-to-cluster communication (as a replacement for direct point-to-point communication when replacing nodes in a distributed system by fault-tolerant clusters).

Review: Table of notations somewhere would help the reader, given the lengthy probability analysis with many terms.

Thanks for this great suggestion to improve readability of Section 5.

Revise: We shall include such a table in the revision.

Review: The graphs are difficult to read with various lines looking very similar to each other. The zooming doesn't help readability here.

We have to redesign the figures to include details on variance and further details on tail costs (in response to earlier comments by the reviewer).

Revise: In the redesign of the figures, we will use additional space to further improve readability of the figures (as there is no limit on page size).

SOLUTION: BYZANTINE CLUSTER-SENDING IN EXPECTED CONSTANT COST AND CONSTANT TIME

Anonymous authors
Paper under double-blind review

Abstract

Traditional resilient systems operate on fully-replicated fault-tolerant clusters, which limits their scalability and performance. One way to make the step towards resilient high-performance systems that can deal with huge workloads, is by enabling independent fault-tolerant clusters to efficiently communicate and cooperate with each other, as this also enables the usage of high-performance techniques such as sharding. Recently, such inter-cluster communication was formalized as the *Byzantine cluster-sending problem*. Unfortunately, existing worst-case optimal protocols for cluster-sending all have *linear complexity* in the size of the clusters involved.

In this paper, we propose *probabilistic cluster-sending techniques* as a solution for the cluster-sending problem with only an *expected constant message complexity*, this independent of the size of the clusters involved and this even in the presence of highly unreliable communication. Depending on the robustness of the clusters involved, our techniques require only *two-to-four* message round-trips (without communication failures). Furthermore, our protocols can support worst-case linear communication between clusters. Finally, we have put our techniques to the test in an in-depth experimental evaluation that further underlines the exceptional low expected costs of our techniques in comparison with other protocols. As such, our work provides a strong foundation for the further development of resilient high-performance systems.

1 Introduction

The promises of *resilient data processing*, as provided by private and public blockchains [14, 20, 26], has renewed interest in traditional consensus-based Byzantine fault-tolerant resilient systems [5,6,23]. Unfortunately, blockchains and other consensus-based systems typically rely on fully-replicated designs, which limits their scalability and performance. Consequently, these systems cannot deal with the ever-growing requirements in data processing [28, 29].

One wat to improve on these limitations is by building complex system designs that consist of *independently-operating* resilient clusters that can cooperate to provide certain services. To illustrate this, one can consider a sharded resilient design. In a traditional resilient systems, resilience is provided by a fully-replicated consensus-based Byzantine fault-tolerant cluster in which all replicas hold all data and process all requests. This traditional design has only limited performance, even with the best consensus protocols, and lacks scalability. To improve on the design of traditional systems, one can employ the *sharded* design of Figure 1. In this sharded design, each cluster only holds part of the data. Consequently, each cluster only needs to process requests that affect data they hold. In this way, this sharded design improves performance by enabling *parallel processing* of requests by different clusters, while also improving storage scalability. To support requests that affect data in several clusters in such a sharded design, the clusters need to be able to *coordinate their operations*, however [1, 7, 15, 18].

Central to such complex system designs is the ability to reliably and efficiently communicate between independently-operating resilient clusters. Recently, this problem of communication *between* Byzantine fault-tolerant clusters has been formalized as the *cluster-sending problem* [17]. We believe that efficient solutions to this problem have a central role towards bridging *resilient* and *high-performance* data processing.

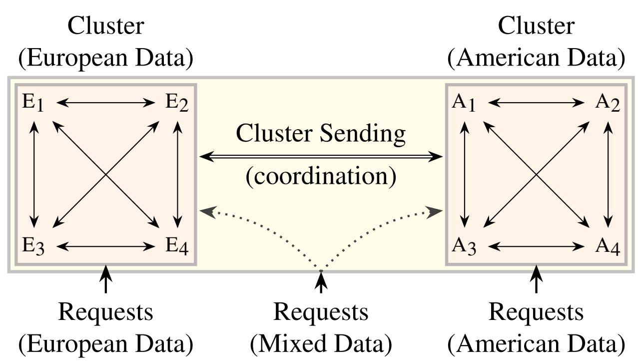

Figure 1: A *sharded* design in which each resilient cluster of four replicas holds only a part of the data. Local decisions within a cluster are made via *consensus* (⟷), whereas multi-shard coordination to process multi-shard transactions requires *cluster-sending* (⟺).

Although the cluster-sending problem has received some attention (e.g., as part of the design of AHL [7], BYSHARD [18], GEOBFT [15], and CHAINSPACE [1]), and cluster-sending protocols that solve the cluster-sending problem with worst-case optimal complexity are known [17], we believe there is still much room for improvement.

In this paper, we introduce a new solution to the cluster-

Figure 2: A comparison of *cluster-sending protocols* that send a value from cluster $\mathcal{C}_1$ with $\mathbf{n}_{\mathcal{C}_1}$ replicas, of which $\mathbf{f}_{\mathcal{C}_1}$ are faulty, to cluster $\mathcal{C}_2$ with $\mathbf{n}_{\mathcal{C}_2}$ replicas, of which $\mathbf{f}_{\mathcal{C}_2}$ are faulty. For each protocol P, *Protocol* specifies its name; *Robustness* specifies the conditions P puts on the clusters; *Message Steps* specifies the number of messages exchanges P performs; *Optimal* specifies whether P is worst-case optimal; and *Unreliable* specifies whether P can deal with unreliable communication.

	Protocol	Robustness[a]	Message Steps (expected-case)	Message Steps (worst-case)	Optimal	Unreliable
	PBS-cs [17]	$\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > \mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2}$	$\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2} + 1$		✔	✘
	PBS-cs [17]	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	$\max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$		✔	✘
	GeoBFT [15]	$\mathbf{n}_{\mathcal{C}_1} = \mathbf{n}_{\mathcal{C}_2} > 3\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$	$\mathbf{f}_{\mathcal{C}_2} + 1$[b]	$\Omega(\mathbf{f}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2})$	✘	✔
	ChainSpace [1]	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	$\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2}$		✘	✘
This Paper	CSpp	$\mathbf{n}_{\mathcal{C}_1} > 2\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 2\mathbf{f}_{\mathcal{C}_2}$	4	$(\mathbf{f}_{\mathcal{C}_1}+1)(\mathbf{f}_{\mathcal{C}_2}+1)$	✘	✔
	CSpp	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	$2\frac{1}{4}$	$(\mathbf{f}_{\mathcal{C}_1}+1)(\mathbf{f}_{\mathcal{C}_2}+1)$	✘	✔
	CSpl	$\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > \mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2}$	4	$\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2} + 1$	✔	✔
	CSpl	$\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 2(\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2})$	$2\frac{1}{4}$	$\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2} + 1$	✔	✔
	CSpl	$\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$	3	$\max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$	✔	✔

[a] Protocols that have different message step complexities depending on the robustness assumptions have been included for each of the robustness assumptions.
[b] Complexity when the coordinating primary in $\mathcal{C}_1$ is non-faulty and communication is reliable.

sending problem: we introduce cluster-sending protocols that use *probabilistic cluster-sending* techniques and are able to provide low *expected-case* message complexity (at the cost of higher communication latencies, a good trade-off in systems where inter-cluster network bandwidth is limited). In specific, our main contributions are as follows:

1. First, in Section 3, we introduce the cluster-sending step cs-step that attempts to send a value from a replica in the sending cluster to a replica in the receiving cluster in a verifiable manner and with a constant amount of inter-cluster communication.

2. Then, in Section 4, we introduce the *Synchronous Probabilistic Cluster-Sending protocol* CSp that uses cs-step with randomly selected sending and receiving replicas to provide cluster-sending in *expected constant* steps. We also propose *pruned*-CSp (CSpp), a fine-tuned version of CSp that guarantees termination.

3. In Section 5, we propose the *Synchronous Probabilistic Linear Cluster-Sending protocol* CSpl, that uses cs-step with a specialized randomized scheme to select replicas, this to provide cluster-sending in *expected constant* steps and *worst-case linear* steps, which is optimal.

4. Next, in Section 6, we discuss how CSp, CSpp, and CSpl can be generalized to operate in environments with *asynchronous and unreliable communication*.

5. Finally, in Section 7, we evaluate the behavior of the proposed probabilistic cluster-sending protocols via an in-depth evaluation. In this evaluation, we show that probabilistic cluster-sending protocols has exceptionally low communication costs in comparison with existing cluster-sending protocols, this even in the presence of communication failures.

A summary of our findings in comparison with existing techniques can be found in Figure 2. In Section 2, we introduce the necessary terminology and notation, in Section 8, we compare with related work, and in Section 9, we conclude on our findings.

2 The Cluster-Sending Problem

Before we present our probabilistic cluster-sending techniques, we first introduce all necessary terminology and notation. The formal model we use is based on the formalization of the cluster-sending problem provided by Hellings et al. [17]. If S is a set of replicas, then $\mathsf{f}(S) \subseteq S$ denotes the *faulty replicas* in S, whereas $\mathsf{nf}(S) = S \setminus \mathsf{f}(S)$ denotes the *non-faulty replicas* in S. We write $\mathbf{n}_S = |S|$, $\mathbf{f}_S = |\mathsf{f}(S)|$, and $\mathbf{nf}_S = |\mathsf{nf}(S)| = \mathbf{n}_S - \mathbf{f}_S$ to denote the number of replicas, faulty replicas, and non-faulty replicas in S, respectively. A *cluster* $\mathcal{C}$ is a finite set of replicas. We consider clusters with *Byzantine replicas* that behave in arbitrary manners. In specific, if $\mathcal{C}$ is a cluster, then any malicious adversary can control the replicas in $\mathsf{f}(\mathcal{C})$ at any time, but adversaries cannot bring non-faulty replicas under their control.

Definition 2.1. Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. The *cluster-sending problem* is the problem of sending a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ such that **(1)** all non-faulty replicas in $\mathsf{nf}(\mathcal{C}_2)$ receive the value v; **(2)** all non-faulty replicas in $\mathsf{nf}(\mathcal{C}_1)$ confirm that the value v was received by all non-faulty replicas in $\mathsf{nf}(\mathcal{C}_2)$; and **(3)** non-faulty replicas in $\mathsf{nf}(\mathcal{C}_2)$ only receive a value v if all non-faulty replicas in $\mathsf{nf}(\mathcal{C}_1)$ agree upon sending v.

We assume that there is no limitation on local communication within a cluster, while global communication between clusters is costly. This model is supported by practice, where communication between wide-area deployments of clusters is up-to-two orders of magnitudes more expensive than communication within a cluster [7, 15].

We assume that each cluster can make *local decisions* among all non-faulty replicas, e.g., via a *consensus protocol* such as PBFT or PAXOS [6, 23]. Furthermore, we assume that the replicas in each cluster can certify such local decisions via a *signature scheme*. E.g., a cluster $\mathcal{C}$ can certify a consensus decision on some message m by collecting a set of signatures for m of $\mathbf{f}_{\mathcal{C}}+1$ replicas in $\mathcal{C}$, guaranteeing one such signature is from a non-faulty replica (which would only signs values on which consensus is reached). We write $\langle m \rangle_{\mathcal{C}}$ to denote a message m certified by $\mathcal{C}$. To minimize the size of certified messages, one can utilize a threshold signature scheme [30]. To enable decision making and message certification, we assume, for every cluster $\mathcal{C}$, $\mathbf{n}_{\mathcal{C}} > 2\mathbf{f}_{\mathcal{C}}$, a minimal requirement [9, 24]. Lastly, we assume that there is a common source of randomness for all non-faulty replicas of each cluster, e.g., via a distributed fault-tolerant random coin [3, 4].

3 The Cluster-Sending Step

If communication is reliable and one knows non-faulty replicas $\mathrm{R}_1 \in \mathsf{nf}(\mathcal{C}_1)$ and $\mathrm{R}_2 \in \mathsf{nf}(\mathcal{C}_2)$, then cluster-sending a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ can be done via a straightforward *cluster-sending step*: one can simply instruct R_1 to send v to R_2. When R_2 receives v, it can disperse v locally in $\mathcal{C}_2$. Unfortunately, we do not know which replicas are faulty and which are non-faulty. Furthermore, it is practically impossible to reliably determine which replicas are non-faulty, as non-faulty replicas can appear faulty due to unreliable communication, while faulty replicas can appear well-behaved to most replicas, while interfering with the operations of only some non-faulty replicas.

To deal with faulty replicas when utilizing the above *cluster-sending step*, one needs a sufficient safeguards to detect *failure* of R_1, of R_2, or of the communication between them. To do so, we add receive and confirmation phases to the sketched cluster-sending step. During the *receive phase*, the receiving replica R_2 must construct a proof P that it received and dispersed v locally in $\mathcal{C}_2$ and then send this proof back to R_1. Finally, during the *confirmation phase*, R_1 can utilize P to prove to all other replicas in $\mathcal{C}_1$ that the cluster-sending step was successful. The pseudo-code of this *cluster-sending step protocol* CS-STEP can be found in Figure 3. We have the following:

Proposition 3.1. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters with $\mathrm{R}_1 \in \mathcal{C}_1$ and $\mathrm{R}_2 \in \mathcal{C}_2$. If $\mathcal{C}_1$ satisfies the pre-conditions of* CS-STEP(R_1, R_2, v)*, then execution of* CS-STEP(R_1, R_2, v) *satisfies the post-conditions and will exchange at most two messages between $\mathcal{C}_1$ and $\mathcal{C}_2$.*

Protocol CS-STEP(R_1, R_2, v), with $\mathrm{R}_1 \in \mathcal{C}_1$ and $\mathrm{R}_2 \in \mathcal{C}_2$**:**

Pre: Each replica in $\mathsf{nf}(\mathcal{C}_1)$ decided AGREE on sending v to $\mathcal{C}_2$ (and can construct $\langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$).

Post: **(i)** If communication is reliable, $\mathrm{R}_1 \in \mathsf{nf}(\mathcal{C}_1)$, and $\mathrm{R}_2 \in \mathsf{nf}(\mathcal{C}_2)$, then R_1 decides CONFIRM on v. **(ii)** If a replica in $\mathsf{nf}(\mathcal{C}_2)$ decides RECEIVE on v, then all replicas in $\mathsf{nf}(\mathcal{C}_1)$ decided AGREE on sending v to $\mathcal{C}_2$. **(iii)** If a replica in $\mathsf{nf}(\mathcal{C}_1)$ decides CONFIRM on v, then all replicas in $\mathsf{nf}(\mathcal{C}_2)$ decided RECEIVE on v and all replicas in $\mathsf{nf}(\mathcal{C}_1)$ eventually decide CONFIRM on v (whenever communication becomes reliable).

The *cluster-sending step* for R_1 and R_2:

1: Instruct R_1 to send $\langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ to R_2.

The *receive role* for $\mathcal{C}_2$:

2: **event** $\mathrm{R}_2 \in \mathsf{nf}(\mathcal{C}_2)$ receives message $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ from $\mathrm{R}_1 \in \mathcal{C}_1$ **do**
3: **if** R_2 does not have consensus on m **then**
4: Use *local consensus* on m and construct $\langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$.
5: {*Each replica in* $\mathsf{nf}(\mathcal{C}_2)$ *decides* RECEIVE *on* v.}
6: Send $\langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$ to R_1.

The *confirmation role* for $\mathcal{C}_1$:

7: **event** $\mathrm{R}_1 \in \mathsf{nf}(\mathcal{C}_1)$ receives message $m_p := \langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$ with $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ from $\mathrm{R}_2 \in \mathcal{C}_2$ **do**
8: **if** R_1 does not have consensus on m_p **then**
9: Use *local consensus* on m_p.
10: {*Each replica in* $\mathsf{nf}(\mathcal{C}_1)$ *decides* CONFIRM *on* v.}

Figure 3: The Cluster-sending step protocol CS-STEP(R_1, R_2, v). In this protocol, R_1 tries to send v to R_2, which will succeed if both R_1 and R_2 are non-faulty.

Proof. We prove the three post-conditions separately. **(i)** We assume that communication is reliable, $\mathrm{R}_1 \in \mathsf{nf}(\mathcal{C}_1)$, and $\mathrm{R}_2 \in \mathsf{nf}(\mathcal{C}_2)$. Hence, R_1 sends message $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ to R_2 (Line 1 of Figure 3). In the receive phase (Lines 2–6 of Figure 3), replica R_2 *receives* message m from R_1. Replica R_2 uses local consensus on m to replicate m among all replicas $\mathcal{C}_2$ and, along the way, to constructs a *proof of receipt* $m_p := \langle \texttt{proof} : m \rangle_{\mathcal{C}_2}$. As all replicas in $\mathsf{nf}(\mathcal{C}_2)$ participate in this local consensus, all replicas in $\mathsf{nf}(\mathcal{C}_2)$ will decide RECEIVE on v from $\mathcal{C}_1$. Finally, the proof m_p is returned to R_1. In the confirmation phase (Lines 7–10 of Figure 3), replica R_1 receives the proof of receipt m_p. Next, R_1 uses local consensus on m_p to replicate m_p among all replicas in $\mathsf{nf}(\mathcal{C}_1)$, after which all replicas in $\mathsf{nf}(\mathcal{C}_1)$ decide CONFIRM on sending v to $\mathcal{C}_2$

(ii) A replica in $\mathsf{nf}(\mathcal{C}_2)$ only decides RECEIVE on v after consensus is reached on a message $m := \langle \texttt{send} : v,\ \mathcal{C}_2 \rangle_{\mathcal{C}_1}$ (Line 5 of Figure 3). This message m not only contains the value v, but also the identity of the recipient cluster $\mathcal{C}_2$. Due to the usage of certificates and the pre-condition, the message m cannot be created without the replicas in $\mathsf{nf}(\mathcal{C}_1)$ deciding AGREE on sending v to $\mathcal{C}_2$.

(iii) A replica in $\mathsf{nf}(\mathcal{C}_1)$ only decides CONFIRM on v after consensus is reached on a *proof of receipt* message $m_p := \langle \texttt{proof} : m \rangle_{C_2}$ (Line 10 of Figure 3). This consensus step will complete for all replicas in C_1 whenever communication becomes reliable. Hence, all replicas in $\mathsf{nf}(C_1)$ will eventually decide CONFIRM on v. Due to the usage of certificates, the message m_p cannot be created without cooperation of the replicas in $\mathsf{nf}(C_2)$. The replicas in $\mathsf{nf}(C_2)$ only cooperate in constructing m_p as part of the consensus step of Line 4 of Figure 3. Upon completion of this consensus step, all replicas in $\mathsf{nf}(C_2)$ will decide RECEIVE on v. □

In the following sections, we show how to use the cluster-sending step in the construction of cluster-sending protocols. In Section 4, we introduce synchronous protocols that provide *expected constant message complexity*. Then, in Section 5, we introduce synchronous protocols that additionally provide *worst-case linear message complexity*, which is optimal. Finally, in Section 6, we show how to extend the presented techniques to asynchronous communication.

4 Probabilistic Cluster-Sending with Random Replica Selection

In the previous section, we introduced CS-STEP, the cluster-sending step protocol that succeeds whenever the participating replicas are non-faulty and communication is reliable. Using CS-STEP, we build a three-step protocol that cluster-sends a value v from C_1 to C_2:

1. First, the replicas in $\mathsf{nf}(C_1)$ reach agreement and decide AGREE on sending v to C_2.
2. Then, the replicas in $\mathsf{nf}(C_1)$ perform a *probabilistic cluster-sending step* by electing replicas $R_1 \in C_1$ and $R_2 \in C_2$ fully at random, after which CS-STEP(R_1, R_2, v) is executed.
3. Finally, each replicas in $\mathsf{nf}(C_1)$ waits for the completion of CS-STEP(R_1, R_2, v) If the waiting replicas decided CONFIRM on v during this wait, then cluster-sending is successful. Otherwise, we repeat the previous step.

To simplify presentation, we assume *synchronous* inter-cluster communication to enable replicas to wait for completion: messages sent by non-faulty replicas will be delivered within some known bounded delay. We refer to Section 6 on how to deal with asynchronous and unreliable communication. *Synchronous* systems can be modeled by *pulses* [10, 11]:

Definition 4.1. A system is *synchronous* if all inter-cluster communication happens in *pulses* such that every message sent in a pulse will be received in the same pulse.

The pseudo-code of the resultant *Synchronous Probabilistic Cluster-Sending protocol* CSP can be found in Figure 4. Next, we prove that CSP is correct and has expected-case constant message complexity:

Protocol CSP(C_1, C_2, v)**:**

1: Use *local consensus* on v and construct $\langle \texttt{send} : v,\ C_2 \rangle_{C_1}$.
2: {*Each replica in* $\mathsf{nf}(C_1)$ *decides* AGREE *on* v.}
3: **repeat**
4:     Choose replicas $(R_1, R_2) \in C_1 \times C_2$, fully at random.
5:     CS-STEP(R_1, R_2, v)
6:     Wait *three* global pulses.
7: **until** C_1 reaches consensus on $\langle \texttt{proof} : \langle \texttt{send} : v,\ C_2 \rangle_{C_1} \rangle_{C_2}$.

Figure 4: The Synchronous Probabilistic Cluster-Sending protocol CSP(C_1, C_2, v) that cluster-sends a value v from C_1 to C_2.

Theorem 4.2. *Let C_1, C_2 be disjoint clusters. If communication is synchronous, then* CSP(C_1, C_2, v) *results in cluster-sending v from C_1 to C_2. The execution performs two local consensus steps in C_1, one local consensus step in C_2, and is expected to make $(\mathbf{n}_{C_1}\mathbf{n}_{C_2})/(\mathbf{nf}_{C_1}\mathbf{nf}_{C_1})$ cluster-sending steps.*

Proof. Due to Lines 1–2 of Figure 4, CSP(C_1, C_2, v) establishes the pre-conditions for any execution of CS-STEP(R_1, R_2, v) with $R_1 \in C_1$ and $R_2 \in C_2$. Using the correctness of CS-STEP (Proposition 3.1), we conclude that CSP(C_1, C_2, v) results in cluster-sending v from C_1 to C_2 whenever the replicas $(R_1, R_2) \in C_1 \times C_2$ chosen at Line 4 of Figure 4 are non-faulty. As the replicas $(R_1, R_2) \in C_1 \times C_2$ are chosen fully at random, we have probability $p_i = \mathbf{nf}_{C_i}/\mathbf{n}_{C_i}$, $i \in \{1,2\}$, of choosing $R_i \in \mathsf{nf}(C_i)$. The probabilities p_1 and p_2 are independent of each other. Consequently, the probability of choosing $(R_1, R_2) \in \mathsf{nf}(C_1) \times \mathsf{nf}(C_2)$ is $p = p_1 p_2 = (\mathbf{nf}_{C_1}\mathbf{nf}_{C_2})/(\mathbf{n}_{C_1}\mathbf{n}_{C_2})$. As such, each iteration of the loop at Line 3 of Figure 4 can be modeled as an independent *Bernoulli trial* with probability of success p, and the expected number of iterations of the loop is $p^{-1} = (\mathbf{n}_{C_1}\mathbf{n}_{C_2})/(\mathbf{nf}_{C_1}\mathbf{nf}_{C_1})$.

Finally, we prove that each local consensus step needs to be performed only once. To do so, we consider the local consensus steps triggered by the loop at Line 3 of Figure 4. These are the local consensus steps at Lines 4 and 9 of Figure 3. The local consensus step at Line 4 can be initiated by a faulty replica R_2. After this single local consensus step reaches consensus on message $m := \langle \texttt{send} : v,\ C_2 \rangle_{C_1}$, each replica in $\mathsf{nf}(C_2)$ reaches consensus on m, decides RECEIVE on v, and can construct $m_p := \langle \texttt{proof} : m \rangle_{C_2}$, this independent of the behavior of R_2. Hence, a single local consensus step for m in C_2 suffices, and no replica in $\mathsf{nf}(C_2)$ will participate in future consensus steps for m. An analogous argument proves that a single local consensus step for m_p in C_1, performed at Line 9 of Figure 3, suffices. □

Remark 4.3. Although Theorem 4.2 indicates local consensus steps in clusters C_1 and C_2, these local consensus steps typically come for *free* as part of the protocol that uses cluster-sending as a building block. To see this, we consider a multi-shard transaction processed by clusters C_1 and C_2.

The decision of cluster $\mathcal{C}_1$ to send a value v to cluster $\mathcal{C}_2$ is a consequence of the execution of some transaction τ in $\mathcal{C}_1$. Before the replicas in $\mathcal{C}_1$ execute τ, they need to reach consensus on the order in which τ is executed in $\mathcal{C}_1$. As part of this consensus step, the replicas in $\mathcal{C}_1$ can also construct $\langle \texttt{send}: v,\ \mathcal{C}_2\rangle_{\mathcal{C}_1}$ without additional consensus steps. Hence, no consensus step is necessary in $\mathcal{C}_1$ to send value v. Likewise, if value v is received by replicas in $\mathcal{C}_2$ as part of some multi-shard transaction execution protocol, then the replicas in $\mathcal{C}_2$ need to perform the necessary transaction execution steps as a *consequence* of receiving v. To do so, the replicas in $\mathcal{C}_2$ need to reach consensus on the order in which these transaction execution steps are performed. As part of this consensus step, the replicas in $\mathcal{C}_2$ can also constructing a proof of receipt for v.

In typical fault-tolerant clusters, at least half of the replicas are non-faulty (e.g., in synchronous systems with Byzantine failures that use digital signatures, or in systems that only deal with crashes) or at least two-third of the replicas are non-faulty (e.g., asynchronous systems). In these systems, CSP is expected to only performs a few cluster-sending steps:

Corollary 4.4. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. If communication is synchronous, then the expected number of cluster-sending steps performed by* CSP$(\mathcal{C}_1,\ \mathcal{C}_2,\ v)$ *is upper bounded by* 4 *if* $\mathbf{n}_{\mathcal{C}_1} > 2\mathbf{f}_{\mathcal{C}_1}$ *and* $\mathbf{n}_{\mathcal{C}_2} > 2\mathbf{f}_{\mathcal{C}_2}$*; and by* $2\frac{1}{4}$ *if* $\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}$ *and* $\mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$.

In CSP, the replicas $(\mathrm{R}_1, \mathrm{R}_2) \in \mathcal{C}_1 \times \mathcal{C}_2$ are chosen fully at random and *with replacement*, as CSP does not retain any information on *failed* probabilistic steps. In the worst case, this prevents *termination*, as the same pair of replicas can be picked repeatedly. Furthermore, CSP does not prevent the choice of faulty replicas whose failure could be detected. We can easily improve on this, as the *failure* of a probabilistic step provides some information on the chosen replicas. In specific, we have the following technical properties:

Lemma 4.1. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. We assume synchronous communication and assume that each replica in* $\mathsf{nf}(\mathcal{C}_1)$ *decided* AGREE *on sending v to $\mathcal{C}_2$.*

1. *Let $(\mathrm{R}_1, \mathrm{R}_2) \in \mathcal{C}_1 \times \mathcal{C}_2$. If* CS-STEP$(\mathrm{R}_1,\ \mathrm{R}_2,\ v)$ *fails to cluster-send v, then either $\mathrm{R}_1 \in \mathsf{f}(\mathcal{C}_1)$, $\mathrm{R}_2 \in \mathcal{C}_2$, or both.*
2. *Let $\mathrm{R}_1 \in \mathcal{C}_1$. If* CS-STEP$(\mathrm{R}_1,\ \mathrm{R}_2,\ v)$ *fails to cluster-send v for $\mathbf{f}_{\mathcal{C}_2} + 1$ distinct replicas $\mathrm{R}_2 \in \mathcal{C}_2$, then $\mathrm{R}_1 \in \mathsf{f}(\mathcal{C}_1)$.*
3. *Let $\mathrm{R}_2 \in \mathcal{C}_2$. If* CS-STEP$(\mathrm{R}_1,\ \mathrm{R}_2,\ v)$ *fails to cluster-send v for $\mathbf{f}_{\mathcal{C}_1} + 1$ distinct replicas $\mathrm{R}_1 \in \mathcal{C}_1$, then $\mathrm{R}_2 \in \mathsf{f}(\mathcal{C}_2)$.*

Proof. The statement of this Lemma assumes that the pre-conditions for any execution of CS-STEP$(\mathrm{R}_1,\ \mathrm{R}_2,\ v)$ with $\mathrm{R}_1 \in \mathcal{C}_1$ and $\mathrm{R}_2 \in \mathcal{C}_2$ are established. Hence, by Proposition 3.1, CS-STEP$(\mathrm{R}_1,\ \mathrm{R}_2,\ v)$ will cluster-send v if $\mathrm{R}_1 \in \mathsf{nf}(\mathcal{C}_1)$ and $\mathrm{R}_2 \in \mathsf{nf}(\mathcal{C}_2)$. If the cluster-sending step fails to cluster-send v, then one of the replicas involved must be faulty, proving the first property. Next, let $\mathrm{R}_1 \in \mathcal{C}_1$ and consider a set $S \subseteq \mathcal{C}_2$ of $\mathbf{n}_S = \mathbf{f}_{\mathcal{C}_2} + 1$ replicas such that, for all $\mathrm{R}_2 \in S$, CS-STEP$(\mathrm{R}_1,\ \mathrm{R}_2,\ v)$ fails to cluster-send v. Let $S' = S \setminus \mathsf{f}(\mathcal{C}_2)$ be the non-faulty replicas in S. As $\mathbf{n}_S > \mathbf{f}_{\mathcal{C}_2}$, we have $\mathbf{n}_{S'} \geq 1$ and there exists a $\mathrm{R}_2' \in S'$. As $\mathrm{R}_2' \notin \mathsf{f}(\mathcal{C}_2)$ and CS-STEP$(\mathrm{R}_1,\ \mathrm{R}_2',\ v)$ fails to cluster-send v, we must have $\mathrm{R}_1 \in \mathsf{f}(\mathcal{C}_1)$ by the first property, proving the second property. An analogous argument proves the third property. □

We can apply the properties of Lemma 4.1 to actively *prune* which replica pairs CSP considers (Line 4 of Figure 4). Notice that pruning via Lemma 4.1(1) simply replaces choosing replica pairs *with replacement*, as done by CSP, by choosing replica pairs *without replacement*, this without further reducing the possible search space. Pruning via Lemma 4.1(2) does reduce the search space, however, as each replica in $\mathcal{C}_1$ will only be paired with a subset of $\mathbf{f}_{\mathcal{C}_2} + 1$ replicas in $\mathcal{C}_2$. Likewise, pruning via Lemma 4.1(3) also reduces the search space. We obtain the *Pruned Synchronous Probabilistic Cluster-Sending protocol* (CSPP) by applying all three prune steps to CSP. By construction, Theorem 4.2, and Lemma 4.1, we conclude:

Corollary 4.5. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters. If communication is synchronous, then* CSPP$(\mathcal{C}_1,\ \mathcal{C}_2,\ v)$ *results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$. The execution performs two local consensus steps in $\mathcal{C}_1$, one local consensus step in $\mathcal{C}_2$, is expected to make less than $(\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2})/(\mathbf{nf}_{\mathcal{C}_1}\mathbf{nf}_{\mathcal{C}_1})$ cluster-sending steps, and makes worst-case $(\mathbf{f}_{\mathcal{C}_1}+1)(\mathbf{f}_{\mathcal{C}_2}+1)$ cluster-sending steps.*

5 Worst-Case Linear-Time Probabilistic Cluster-Sending

In the previous section, we introduced CSP and CSPP, two probabilistic cluster-sending protocols that can cluster-send a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ with expected constant cost. Unfortunately, CSP does not guarantee termination, while CSPP has a worst-case *quadratic complexity*. To improve on this, we need to improve the scheme by which we select replica pairs $(\mathrm{R}_1, \mathrm{R}_2) \in \mathcal{C}_1 \times \mathcal{C}_2$ that we use in cluster-sending steps. The straightforward manner to guarantee a worst-case *linear complexity* is by using a scheme that can select only up-to-$n = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$ distinct pairs $(\mathrm{R}_1, \mathrm{R}_2) \in \mathcal{C}_1 \times \mathcal{C}_2$. To select n replica pairs from $\mathcal{C}_1 \times \mathcal{C}_2$, we will proceed in two steps.

1. We generate list S_1 of n replicas taken from $\mathcal{C}_1$ and list S_2 of n replicas taken from $\mathcal{C}_2$.
2. Then, we choose permutations $P_1 \in \mathsf{perms}(S_1)$ and $P_2 \in \mathsf{perms}(S_2)$ fully at random, and interpret each pair $(P_1[i], P_2[i])$. $0 \leq i < n$, as one of the chosen replica pairs.

We use the first step to deal with any differences in the sizes of $\mathcal{C}_1$ and $\mathcal{C}_2$, and we use the second step to introduce sufficient randomness in our protocol to yield an low expected-case message complexity.

Next, we introduce some notations to simplify reasoning about the above list-based scheme. If R is a set of replicas, then $\mathsf{list}(R)$ is the list consisting of the replicas in R placed in a predetermined order (e.g., on increasing replica identifier). If S is a list of replicas, then we write $\mathsf{f}(S)$ to denote the faulty replicas in S and $\mathsf{nf}(S)$ to denote the non-faulty replicas in S, and we write $\mathbf{n}_S = |S|$, $\mathbf{f}_S = |\{i \mid (0 \leq i < \mathbf{n}_S) \wedge S[i] \in \mathsf{f}(S)\}|$, and $\mathbf{nf}_S = \mathbf{n}_S - \mathbf{f}_S$ to denote the number of positions in S with replicas, faulty replicas, and non-faulty replicas, respectively. If (P_1, P_2) is a pair of equal-length lists of $n = |P_1| = |P_2|$ replicas, then we say that the i-th position is a *faulty position* if either $P_1[i] \in \mathsf{f}(P_1)$ or $P_2[i] \in \mathsf{f}(P_2)$. We write $\|P_1; P_2\|_{\mathbf{f}}$ to denote the number of *faulty positions* in (P_1, P_2). As faulty positions can only be constructed out of the $\mathbf{f}_{P_1}$ faulty replicas in P_1 and the $\mathbf{f}_{P_2}$ faulty replicas in P_2, we must have $\max(\mathbf{f}_{P_1}, \mathbf{f}_{P_2}) \leq \|P_1; P_2\|_{\mathbf{f}} \leq \min(n, \mathbf{f}_{P_1} + \mathbf{f}_{P_2})$.

Example 5.1. Consider clusters $\mathcal{C}_1, \mathcal{C}_2$ with

$$S_1 = \mathsf{list}(\mathcal{C}_1) = [\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,5}], \quad \mathsf{f}(\mathcal{C}_1) = \{\mathrm{R}_{1,1}, \mathrm{R}_{1,2}\};$$
$$S_2 = \mathsf{list}(\mathcal{C}_2) = [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,5}], \quad \mathsf{f}(\mathcal{C}_2) = \{\mathrm{R}_{2,1}, \mathrm{R}_{2,2}\}.$$

The set $\mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ contains $5!^2 = 14400$ list pairs. Now, consider the list pairs $(P_1, P_2), (Q_1, Q_2), (R_1, R_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with

$$P_1[\underline{\mathrm{R}_{1,1}}, \mathrm{R}_{1,5}, \underline{\mathrm{R}_{1,2}}, \mathrm{R}_{1,4}, \mathrm{R}_{1,3}],$$
$$P_2[\underline{\mathrm{R}_{2,1}}, \mathrm{R}_{2,3}, \underline{\mathrm{R}_{2,2}}, \mathrm{R}_{2,5}, \mathrm{R}_{2,4}];$$
$$Q_1[\underline{\mathrm{R}_{1,1}}, \mathrm{R}_{1,3}, \mathrm{R}_{1,5}, \mathrm{R}_{1,4}, \underline{\mathrm{R}_{1,2}}],$$
$$Q_2[\mathrm{R}_{2,5}, \mathrm{R}_{2,4}, \mathrm{R}_{2,3}, \underline{\mathrm{R}_{2,2}}, \underline{\mathrm{R}_{2,1}}];$$
$$R_1[\mathrm{R}_{1,5}, \mathrm{R}_{1,4}, \mathrm{R}_{1,3}, \underline{\mathrm{R}_{1,2}}, \underline{\mathrm{R}_{1,1}}],$$
$$R_2[\underline{\mathrm{R}_{2,1}}, \underline{\mathrm{R}_{2,2}}, \mathrm{R}_{2,3}, \mathrm{R}_{2,4}, \mathrm{R}_{2,5}].$$

We have underlined the faulty replicas in each list, and we have $\|P_1; P_2\|_{\mathbf{f}} = 2 = \mathbf{f}_{S_1} = \mathbf{f}_{S_2}$, $\|Q_1; Q_2\|_{\mathbf{f}} = 3$, and $\|R_1; R_2\|_{\mathbf{f}} = 4 = \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$.

In the following, we will use a *list-pair function* Φ to compute the initial list-pair (S_1, S_2) of n replicas taken from $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. We build a cluster-sending protocol that uses Φ to compute S_1 and S_2, uses randomization to choose n replica pairs from $S_1 \times S_2$, and, finally, performs cluster-sending steps using only these n replica pairs. The pseudo-code of the resultant *Synchronous Probabilistic Linear Cluster-Sending protocol* CSPL can be found in Figure 5. Next, we prove that CSPL is correct and has a worst-case linear message complexity:

Proposition 5.1. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters and let Φ be a list-pair function with $(S_1, S_2) := \Phi(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. If communication is synchronous and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$, then* CSPL$(\mathcal{C}_1, \mathcal{C}_2, v, \Phi)$ *results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$. The execution performs two local consensus steps in $\mathcal{C}_1$, one local consensus step in $\mathcal{C}_2$, and makes worst-case $\mathbf{f}_{S_1} + \mathbf{f}_{S_2} + 1$ cluster-sending steps.*

Protocol CSPL$(\mathcal{C}_1, \mathcal{C}_2, v, \Phi)$**:**

1: Use *local consensus* on v and construct $\langle \texttt{send} : v, \mathcal{C}_2 \rangle_{\mathcal{C}_1}$.
2: {*Each replica in* $\mathsf{nf}(\mathcal{C}_1)$ *decides* AGREE *on* v.}
3: Let $(S_1, S_2) := \Phi(\mathcal{C}_1, \mathcal{C}_2)$.
4: Choose $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ fully at random.
5: $i := 0$.
6: **repeat**
7: CS-STEP$(P_1[i], P_2[i], v)$
8: Wait *three* global pulses.
9: $i := i + 1$.
10: **until** $\mathcal{C}_1$ reaches consensus on $\langle \texttt{proof} : \langle \texttt{send} : v, \mathcal{C}_2 \rangle_{\mathcal{C}_1} \rangle_{\mathcal{C}_2}$.

Figure 5: The Synchronous Probabilistic Linear Cluster-Sending protocol CSPL$(\mathcal{C}_1, \mathcal{C}_2, v, \Phi)$ that cluster-sends a value v from $\mathcal{C}_1$ to $\mathcal{C}_2$ using list-pair function Φ.

Proof. Due to Lines 1–2 of Figure 5, CSPL$(\mathcal{C}_1, \mathcal{C}_2, v, \Phi)$ establishes the pre-conditions for any execution of CS-STEP$(\mathrm{R}_1, \mathrm{R}_2, v)$ with $\mathrm{R}_1 \in \mathcal{C}_1$ and $\mathrm{R}_2 \in \mathcal{C}_2$. Now let $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$, as chosen at Line 4 of Figure 5. As P_i, $i \in \{1, 2\}$, is a permutation of S_i, we have $\mathbf{f}_{P_i} = \mathbf{f}_{S_i}$. Hence, we have $\|P_1; P_2\|_{\mathbf{f}} \leq \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ and there must exist a position j, $0 \leq j < n$, such that $(P_1[j], P_2[j]) \in \mathsf{nf}(\mathcal{C}_1) \times \mathsf{nf}(\mathcal{C}_2)$. Using the correctness of CS-STEP (Proposition 3.1), we conclude that CSPL$(\mathcal{C}_1, \mathcal{C}_2, v, \Phi)$ results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$ in at most $\mathbf{f}_{S_1} + \mathbf{f}_{S_2} + 1$ cluster-sending steps. Finally, the bounds on the number of consensus steps follow from an argument analogous to the one in the proof of Theorem 4.2. $\square$

Next, we proceed in two steps to arrive at practical instances of CSPL with expected constant message complexity. First, in Section 5.1, we study the probabilistic nature of CSPL. Then, in Section 5.2, we propose practical list-pair functions and show that these functions yield instances of CSPL with expected constant message complexity.

5.1 The Expected-Case Complexity of CSPL

As the first step to determine the expected-case complexity of CSPL, we solve the following abstract problem that captures the probabilistic argument at the core of the expected-case complexity of CSPL:

Problem 5.2 (non-faulty position trials)**.** Let S_1 and S_2 be lists of $|S_1| = |S_2| = n$ replicas. Choose permutations $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ fully at random. Next, we inspect positions in P_1 and P_2 fully at random (with replacement). The *non-faulty position trials problem* asks how many positions one expects to inspect to find the first non-faulty position.

Let S_1 and S_2 be list of $|S_1| = |S_2| = n$ replicas. To answer the non-faulty position trials problem, we first look at the combinatorics of *faulty positions* in pairs $(P_1,P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. Let $m_1 = \mathbf{f}_{S_1}$ and $m_2 = \mathbf{f}_{S_2}$. By $\mathbb{F}(n,m_1,m_2,k)$, we denote the number of distinct pairs (P_1,P_2) one can construct that have exactly k faulty positions, hence, with $\|P_1;P_2\|_{\mathbf{f}} = k$. As observed, we have $\max(m_1,m_2) \leq \|P_1;P_2\|_{\mathbf{f}} \leq \min(n,m_1+m_2)$ for any pair (P_1,P_2). Hence, we have $\mathbb{F}(n,m_1,m_2,k) = 0$ for all $k < \max(m_1,m_2)$ and $k > \min(n,m_1+m_2)$.

Now consider the step-wise construction of any permutation $(P_1,P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with k faulty positions. First, we choose $(P_1[0],P_2[0])$, the pair at position 0, after which we choose pairs for the remaining $n-1$ positions. For $P_i[0]$, $i \in \{1,2\}$, we can choose n distinct replicas, of which m_i are faulty. If we pick a non-faulty replica, then the remainder of P_i is constructed out of $n-1$ replicas, of which m_i are faulty. Otherwise, the remainder of P_i is constructed out of $n-1$ replicas of which m_i-1 are faulty. If, due to our choice of $(P_1[0],P_2[0])$, the first position is faulty, then only $k-1$ out of the $n-1$ remaining positions must be faulty. Otherwise, k out of the $n-1$ remaining positions must be faulty. Combining this analysis yields four types for the first pair $(P_1[0],P_2[0])$:

1. *A non-faulty pair* $(P_1[0],P_2[0]) \in \mathsf{nf}(P_1) \times \mathsf{nf}(P_2)$. We have $(n-m_1)(n-m_2)$ such pairs, and we have $\mathbb{F}(n-1,m_1,m_2,k)$ different ways to construct the remainder of P_1 and P_2.
2. *A 1-faulty pair* $(P_1[0],P_2[0]) \in \mathsf{f}(P_1) \times \mathsf{nf}(P_2)$. We have $m_1(n-m_2)$ such pairs, and we have $\mathbb{F}(n-1,m_1-1,m_2,k-1)$ different ways to construct the remainder of P_1 and P_2.
3. *A 2-faulty pair* $(P_1[0],P_2[0]) \in \mathsf{nf}(P_1) \times \mathsf{f}(P_2)$. We have $(n-m_1)m_2$ such pairs, and we have $\mathbb{F}(n-1,m_1,m_2-2,k-1)$ different ways to construct the remainder of P_1 and P_2.
4. *A both-faulty pair* $(P_1[0],P_2[0]) \in \mathsf{f}(P_1) \times \mathsf{f}(P_2)$. We have m_1m_2 such pairs, and we have $\mathbb{F}(n-1,m_1-1,m_2-1,k-1)$ different ways to construct the remainder of P_1 and P_2.

Hence, for all k, $\max(m_1,m_2) \leq k \leq \min(n,m_1+m_2)$, $\mathbb{F}(n,m_1,m_2,k)$ is recursively defined by:

$$\begin{aligned}
\mathbb{F}(n,m_1,m_2,k) = (n-m_1)(n-m_2)&\mathbb{F}(n-1,m_1,m_2,k) \\
&\text{(non-faulty pair)} \\
+m_1(n-m_2)&\mathbb{F}(n-1,m_1-1,m_2,k-1) \\
&\text{(1-faulty pair)} \\
+(n-m_1)m_2&\mathbb{F}(n-1,m_1,m_2-1,k-1) \\
&\text{(2-faulty pair)} \\
+m_1m_2&\mathbb{F}(n-1,m_1-1,m_2-1,k-1), \\
&\text{(both-faulty pair)}
\end{aligned}$$

and the base case for this recursion is $\mathbb{F}(0,0,0,0) = 1$.

Example 5.3. Reconsider the list pairs (P_1,P_2), (Q_1,Q_2), and (R_1,R_2) from Example 5.1. In (P_1,P_2), we have both-faulty pairs at positions 0 and 2 and non-faulty pairs at positions 1, 3, and 4. In (Q_1,Q_2), we have a 1-faulty pair at position 0, non-faulty pairs at positions 1 and 2, a 2-faulty pair at position 3, and a both-faulty pair at position 4. Finally, in (R_1,R_2), we have 2-faulty pairs at positions 0 and 1, a non-faulty pair at position 2, and 1-faulty pairs at positions 3 and 4.

Using the combinatorics of faulty positions, we formalize an exact solution to the *non-faulty position trials problem*:

Lemma 5.1. *Let S_1 and S_2 be lists of $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$ replicas with $m_1 = \mathbf{f}_{S_1}$ and $m_2 = \mathbf{f}_{S_2}$. If $m_1+m_2 < n$, then the non-faulty position trials problem $\mathbb{E}(n,m_1,m_2)$ has solution*

$$\frac{1}{n!^2}\left(\sum_{k=\max(m_1,m_2)}^{m_1+m_2} \frac{n}{n-k}\mathbb{F}(n,m_1,m_2,k)\right).$$

Proof. We have $|\mathsf{perms}(S_1)| = |\mathsf{perms}(S_2)| = n!$. Consequently, we have $|\mathsf{perms}(S_1) \times \mathsf{perms}(S_2)| = n!^2$ and we have probability $1/(n!^2)$ to choose any pair $(P_1,P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. Now consider such a pair $(P_1,P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. As there are $\|P_1;P_2\|_{\mathbf{f}}$ faulty positions in (P_1,P_2), we have probability $p(P_1,P_2) = (n-\|P_1;P_2\|_{\mathbf{f}})/n$ to inspect a non-faulty position. Notice that $\max(m_1,m_2) \leq \|P_1;P_2\|_{\mathbf{f}} \leq m_1+m_2 < n$ and, hence, $0 < p(P_1,P_2) \leq 1$. Each of the inspected positions in (P_1,P_2) is chosen fully at random. Hence, each inspection is a *Bernoulli trial* with probability of success $p(P_1,P_2)$, and we expect to inspect a first non-faulty position in the $p(P_1,P_2)^{-1} = n/(n-\|P_1;P_2\|_{\mathbf{f}})$-th attempt. We conclude that the non-faulty position trials problem $\mathbb{E}(n,m_1,m_2)$ has solution

$$\frac{1}{n!^2}\left(\sum_{(P_1,P_2)\in\mathsf{perms}(S_1)\times\mathsf{perms}(S_2)} \frac{n}{n-\|P_1;P_2\|_{\mathbf{f}}}\right).$$

Notice that there are $\mathbb{F}(n,m_1,m_2,k)$ distinct pairs $(P_1,P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $\|P_1';P_2'\|_{\mathbf{f}} = k$ for each k, $\max(m_1,m_2) \leq k \leq m_1+m_2 < n$. Hence, in the above expression for $\mathbb{E}(n,m_1,m_2)$, we can group on these pairs (P_1',P_2') to obtain the searched-for solution. □

To further solve the non-faulty position trials problem, we work towards a *closed form* for $\mathbb{F}(n,m_1,m_2,k)$. Consider any pair $(P_1,P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $\|P_1;P_2\|_{\mathbf{f}} = k$ obtained via the outlined step-wise construction. Let b_1 be the number of 1-*faulty pairs*, let b_2 be the number of 2-*faulty pairs*, and let $b_{1,2}$ be the number of *both-faulty pairs* in (P_1,P_2). By construction, we must have $k = b_1+b_2+b_{1,2}$, $m_1 = b_1+b_{1,2}$, and $m_2 = b_2+b_{1,2}$ and by rearranging terms, we can derive

$$b_{1,2} = (m_1+m_2)-k, \quad b_1 = k-m_2, \quad b_2 = k-m_1.$$

Example 5.4. Consider

$$S_1 = [\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,5}], \qquad \mathsf{f}(S_1) = \{\mathrm{R}_{1,1}, \mathrm{R}_{1,2}, \mathrm{R}_{1,3}\};$$
$$S_2 = [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,5}], \qquad \mathsf{f}(S_2) = \{\mathrm{R}_{2,1}\}.$$

Hence, we have $n = 5$, $m_1 = \mathbf{f}_{S_1} = 3$, and $m_2 = \mathbf{f}_{S_2} = 1$. If we want to create a pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $k = \|P_1; P_2\|_{\mathbf{f}} = 3$ faulty positions, then (P_1, P_2) must have two non-faulty pairs, two 1-faulty pairs, no 2-faulty pairs, and one both-faulty pair. Hence, we have $n-k = 2$, $b_1 = 2$, $b_2 = 0$, and $b_{1,2} = 1$.

The above analysis only depends on the choice of m_1, m_2, and k, and not on our choice of (P_1, P_2). Next, we use this analysis to express $\mathbb{F}(n, m_1, m_2, k)$ in terms of the number of distinct ways in which one can *construct*

(A) lists of b_1 1-faulty pairs out of faulty replicas from S_1 and non-faulty replicas from S_2,

(B) lists of b_2 2-faulty pairs out of non-faulty replicas from S_1 and faulty replicas from S_2,

(C) lists of $b_{1,2}$ both-faulty pairs out of the remaining faulty replicas in S_1 and S_2 that are not used in the previous two cases, and

(D) lists of $n-k$ non-faulty pairs out of the remaining (non-faulty) replicas in S_1 and S_2 that are not used in the previous three cases;

and in terms of the number of distinct ways one can *merge* these lists. As the first step, we look at how many distinct ways we can merge two lists together:

Lemma 5.2. *For any two disjoint lists S and T with $|S| = v$ and $|T| = w$, there exist $\mathbb{M}(v, w) = (v+w)!/(v!w!)$ distinct lists L with $L|_S = S$ and $L|_T = T$, in which $L|_M$, $M \in \{S, T\}$, is the list obtained from L by only keeping the values that also appear in list M.*

Next, we look at the number of distinct ways in which one can construct lists of type A, B, C, and D. Consider the construction of a list of type A. We can choose $\binom{m_1}{b_1}$ distinct sets of b_1 faulty replicas from S_1 and we can choose $\binom{n-m_2}{b_1}$ distinct sets of b_1 non-faulty replicas from S_2. As we can order the chosen values from S_1 and S_2 in $b_1!$ distinct ways, we can construct $b_1!^2\binom{m_1}{b_1}\binom{n-m_2}{b_1}$ distinct lists of type A. Likewise, we can construct $b_2!^2\binom{n-m_1}{b_2}\binom{m_2}{b_2}$ distinct lists of type B.

Example 5.5. We continue from the setting of Example 5.4: we want to create a pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$ with $k = \|P_1; P_2\|_{\mathbf{f}} = 3$ faulty positions. To create (P_1, P_2), we need to create $b_1 = 2$ pairs that are 1-faulty. We have $\binom{m_1}{b_1} = \binom{3}{2} = 3$ sets of two faulty replicas in S_1 that we can choose, namely the sets $\{\mathrm{R}_{1,1}, \mathrm{R}_{1,2}\}$, $\{\mathrm{R}_{1,1}, \mathrm{R}_{1,3}\}$, and $\{\mathrm{R}_{1,2}, \mathrm{R}_{1,3}\}$. Likewise, we have $\binom{n-m_2}{b_1} = \binom{4}{2} = 6$ sets of two non-faulty replicas in S_2 that we can choose. Assume we choose $T_1 = \{\mathrm{R}_{1,1}, \mathrm{R}_{1,3}\}$ from S_1 and $T_2 = \{\mathrm{R}_{2,4}, \mathrm{R}_{2,5}\}$ from S_2. The two replicas in T_1 can be ordered in $\mathbf{n}_{T_1}! = 2! = 2$ ways, namely $[\mathrm{R}_{1,1}, \mathrm{R}_{1,3}]$ and $[\mathrm{R}_{1,3}, \mathrm{R}_{1,1}]$. Likewise, the two replicas in T_2 can be ordered in $\mathbf{n}_{T_2}! = 2! = 2$ ways. Hence, we can construct $2 \cdot 2 = 4$ distinct lists of type A out of this single choice for T_1 and T_2, and the sequences S_1 and S_2 provide us with $\binom{m_1}{b_1}\binom{n-m_2}{b_1} = 18$ distinct choices for T_1 and T_2. We conclude that we can construct 72 distinct lists of type A from S_1 and S_2.

By construction, lists of type A and type B cannot utilize the same replicas from S_1 or S_2. After choosing $b_1 + b_2$ replicas in S_1 and S_2 for the construction of lists of type A and B, the remaining $b_{1,2}$ faulty replicas in S_1 and S_2 are all used for constructing lists of type C. As we can order these remaining values from S_1 and S_2 in $b_{1,2}!$ distinct ways, we can construct $b_{1,2}!^2$ distinct lists of type C (per choice of lists of type A and B). Likewise, the remaining $n-k$ non-faulty replicas in S_1 and S_2 are all used for constructing lists of type D, and we can construct $(n-k)!^2$ distinct lists of type D (per choice of lists of type A and B).

As the final steps, we merge lists of type A and B into lists of type AB. We can do so in $\mathbb{M}(b_1, b_2)$ ways and the resultant lists have size $b_1 + b_2$. Next, we merge lists of type AB and C into lists of type ABC. We can do so in $\mathbb{M}(b_1 + b_2, b_{1,2})$ ways and the resultant lists have size k. Finally, we merge list of type ABC and D together, which we can do in $\mathbb{M}(k, n-k)$ ways. From this construction, we derive that $\mathbb{F}(n, m_1, m_2, k)$ is equivalent to

$$b_1!^2\binom{m_1}{b_1}\binom{n-m_2}{b_1}b_2!^2\binom{n-m_1}{b_2}\binom{m_2}{b_2}\cdot \mathbb{M}(b_1, b_2)b_{1,2}!^2\mathbb{M}(b_1+b_2, b_{1,2})(n-k)!^2\mathbb{M}(k, n-k),$$

which can be simplified to the following:

Lemma 5.3. *Let $\max(m_1, m_2) \leq k \leq \min(n, m_1 + m_2)$ and let $b_1 = k - m_2$, $b_2 = k - m_1$, and $b_{1,2} = (m_1 + m_2) - k$. We have*

$$\mathbb{F}(n, m_1, m_2, k) = \frac{m_1!m_2!(n-m_1)!(n-m_2)n!}{b_1!b_2!b_{1,2}!(n-k)!}.$$

We combine Lemma 5.1 and Lemma 5.3 to conclude

Proposition 5.2. *Let S_1 and S_2 be lists of $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$ replicas with $m_1 = \mathbf{f}_{S_1}$, $m_2 = \mathbf{f}_{S_2}$, $b_1 = k - m_2$, $b_2 = k - m_1$, and $b_{1,2} = (m_1 + m_2) - k$. If $m_1 + m_2 < n$, then the non-faulty position trials problem $\mathbb{E}(n, m_1, m_2)$ has solution*

$$\frac{1}{n!^2}\left(\sum_{k=\max(m_1,m_2)}^{m_1+m_2} \frac{n}{n-k}\frac{m_1!m_2!(n-m_1)!(n-m_2)!n!}{b_1!b_2!b_{1,2}!(n-k)!}\right).$$

Finally, we use Proposition 5.2 to derive

Proposition 5.3. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters and let Φ be a list-pair function with $(S_1, S_2) := \Phi(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. If communication is synchronous and $\mathbf{f}_{S_1} + \mathbf{f}_{S_2} < n$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, Φ) *is less than $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$.*

Proof. Let $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. We notice that CSPL inspects positions in P_1 and P_2 in a different way than the non-faulty trials problem: at Line 7 of Figure 5, positions are inspected one-by-one in a predetermined order and not fully at random (with replacement). Next, we will argue that $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$ provides an upper bound on the expected number of cluster-sending steps regardless of these differences. Without loss of generality, we assume that S_1 and S_2 each have n distinct replicas. Consequently, the pair (P_1, P_2) represents a set R of n distinct replica pairs taken from $\mathcal{C}_1 \times \mathcal{C}_2$. We notice that each of the $n!$ permutations of R is represented by a single pair $(P_1', P_2') \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$.

Now consider the selection of positions in (P_1, P_2) fully at random, but without replacement. This process will yield a list $[j_0, \dots, j_{n-1}] \in \mathsf{perms}([0, \dots, n-1])$ of positions fully at random. Let $Q_i = [P_i[j_0], \dots, P_i[j_{n-1}]]$, $i \in \{1, 2\}$. We notice that the pair (Q_1, Q_2) also represents R and we have $(Q_1, Q_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$. Hence, by choosing a pair $(P_1, P_2) \in \mathsf{perms}(S_1) \times \mathsf{perms}(S_2)$, we choose set R fully at random and, at the same time, we choose the order in which replica pairs in R are inspected fully at random.

Finally, we note that CSPL inspects positions without replacement. As the number of expected positions inspected in the non-faulty position trials problem decreases if we choose positions without replacement, we have proven that $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$ is an upper bound on the expected number of cluster-sending steps. □

5.2 Practical Instances of CSPL

As the last step in providing practical instances of CSPL, we need to provide practical list-pair functions to be used in conjunction with CSPL. We provide two such functions that address most practical environments. Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters, let $n_{\min} = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$, and let $n_{\max} = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$. We provide list-pair functions

$$\Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2) \mapsto (\mathsf{list}(\mathcal{C}_1)^{:n_{\min}}, \mathsf{list}(\mathcal{C}_2)^{:n_{\min}}),$$
$$\Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2) \mapsto (\mathsf{list}(\mathcal{C}_2)^{:n_{\max}}, \mathsf{list}(\mathcal{C}_2)^{:n_{\max}}),$$

in which $L^{:n}$ denotes the first n values in the list obtained by repeating list L. Next, we illustrate usage of these functions:

Example 5.6. Consider clusters $\mathcal{C}_1, \mathcal{C}_2$ with

$$S_1 = \mathsf{list}(\mathcal{C}_1) = [\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,9}];$$
$$S_2 = \mathsf{list}(\mathcal{C}_2) = [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,4}].$$

We have

$$\Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2) = ([\mathrm{R}_{1,1}, \dots, \mathrm{R}_{1,4}], \mathsf{list}(\mathcal{C}_2));$$
$$\Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2) = (\mathsf{list}(\mathcal{C}_1), [\mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,4}, \mathrm{R}_{2,1}, \dots, \mathrm{R}_{2,4}, \mathrm{R}_{2,1}]).$$

Next, we combine $\Phi_{\min}$ and $\Phi_{\max}$ with CSPL, show that in practical environments $\Phi_{\min}$ and $\Phi_{\max}$ satisfy the requirements put on list-pair functions in Proposition 5.1 to guarantee termination and cluster-sending, and use these results to determine the expected constant complexity of the resulting instances of CSPL.

Theorem 5.7. *Let $\mathcal{C}_1, \mathcal{C}_2$ be disjoint clusters with synchronous communication.*

1. *If $n = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 2\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\min}$) *is upper bounded by* 4. *For every $(S_1, S_2) := \Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2)$, we have $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$, $n > 2\mathbf{f}_{S_1}$, $n > 2\mathbf{f}_{S_2}$, and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$*

2. *If $n = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 3\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\min}$) *is upper bounded by* $2\frac{1}{4}$. *For every $(S_1, S_2) := \Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2)$, we have $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$, $n > 3\mathbf{f}_{S_1}$, $n > 3\mathbf{f}_{S_2}$, and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$.*

3. *If $\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}$ and $\mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$, then the expected number of cluster-sending steps performed by* CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\max}$) *is upper bounded by* 3. *For every $(S_1, S_2) := \Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2)$, we have $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2} = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ and either we have $\mathbf{n}_{\mathcal{C}_1} \geq \mathbf{n}_{\mathcal{C}_2}$, $n > 3\mathbf{f}_{S_1}$, and $n > 2\mathbf{f}_{S_2}$; or we have $\mathbf{n}_{\mathcal{C}_2} \geq \mathbf{n}_{\mathcal{C}_1}$, $n > 2\mathbf{f}_{S_1}$, and $n > 3\mathbf{f}_{S_2}$.*

Each of these instance of CSPL *results in cluster-sending v from $\mathcal{C}_1$ to $\mathcal{C}_2$.*

Proof. First, we prove the properties of $\Phi_{\min}$ and $\Phi_{\max}$ claimed in the three statements of the theorem. In the first and second statement of the theorem, we have $\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > c\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$, $c \in \{2, 3\}$. Let $(S_1, S_2) := \Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. By definition of $\Phi_{\min}$, we have $n = \min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2})$, in which case S_i, $i \in \{1, 2\}$, holds n distinct replicas from $\mathcal{C}_i$. Hence, we have $\mathbf{f}_{\mathcal{C}_i} \geq \mathbf{f}_{S_i}$ and, as $n > c\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2}) \geq c\mathbf{f}_{\mathcal{C}_i}$, also $n > c\mathbf{f}_{S_i}$. Finally, as $n > 2\mathbf{f}_{S_1}$ and $n > 2\mathbf{f}_{S_2}$, also $2n > 2\mathbf{f}_{S_1} + 2\mathbf{f}_{S_2}$ and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ holds.

In the last statement of the theorem, we have $\mathbf{n}_{\mathcal{C}_1} > 3\mathbf{f}_{\mathcal{C}_1}$ and $\mathbf{n}_{\mathcal{C}_2} > 3\mathbf{f}_{\mathcal{C}_2}$. Without loss of generality, we assume $\mathbf{n}_{\mathcal{C}_1} \geq \mathbf{n}_{\mathcal{C}_2}$. Let $(S_1, S_2) := \Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2)$ and $n = \mathbf{n}_{S_1} = \mathbf{n}_{S_2}$. By definition of $\Phi_{\max}$, we have $n = \max(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) = \mathbf{n}_{\mathcal{C}_1}$. As $n = \mathbf{n}_{\mathcal{C}_1}$, we have $S_1 = \mathsf{list}(\mathcal{C}_1)$. Consequently, we also have $\mathbf{f}_{S_1} = \mathbf{f}_{\mathcal{C}_1}$ and, hence, $\mathbf{n}_{S_1} > 3\mathbf{f}_{\mathcal{C}_1}$. Next, we will show that $\mathbf{n}_{S_2} > 2\mathbf{f}_{S_2}$. Let $q = \mathbf{n}_{\mathcal{C}_1} \operatorname{div} \mathbf{n}_{\mathcal{C}_2}$ and $r = \mathbf{n}_{\mathcal{C}_1} \bmod \mathbf{n}_{\mathcal{C}_2}$. We note that $\mathsf{list}(\mathcal{C}_2)^{:n}$ contains q full copies of $\mathsf{list}(\mathcal{C}_2)$ and one partial copy of $\mathsf{list}(\mathcal{C}_2)$. Let $T \subset \mathcal{C}_2$ be the set of replicas in this partial copy. By construction, we have $\mathbf{n}_{S_2} = q\mathbf{n}_{\mathcal{C}_2} + r >$

$q3\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T + \mathbf{n}\mathbf{f}_T$ and $\mathbf{f}_{S_2} = q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T$ with $\mathbf{f}_T \leq \min(\mathbf{f}_{\mathcal{C}_2}, r)$. As $q > 1$ and $\mathbf{f}_{\mathcal{C}_2} \geq \mathbf{f}_T$, we have $q\mathbf{f}_{\mathcal{C}_2} \geq \mathbf{f}_{\mathcal{C}_2} \geq \mathbf{f}_T$. Hence, $\mathbf{n}_{S_2} > 3q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T + \mathbf{n}\mathbf{f}_T > 2q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T + \mathbf{n}\mathbf{f}_T \geq 2(q\mathbf{f}_{\mathcal{C}_2} + \mathbf{f}_T) + \mathbf{n}\mathbf{f}_T \geq 2\mathbf{f}_{S_2}$. Finally, as $n > 3\mathbf{f}_{S_1}$ and $n > 2\mathbf{f}_{S_2}$, also $2n > 3\mathbf{f}_{S_1} + 2\mathbf{f}_{S_2}$ and $n > \mathbf{f}_{S_1} + \mathbf{f}_{S_2}$ holds.

Now, we prove the upper bounds on the expected number of cluster-sending steps for CSPL($\mathcal{C}_1$, $\mathcal{C}_2$, v, $\Phi_{\min}$) with $\min(\mathbf{n}_{\mathcal{C}_1}, \mathbf{n}_{\mathcal{C}_2}) > 2\max(\mathbf{f}_{\mathcal{C}_1}, \mathbf{f}_{\mathcal{C}_2})$. By Proposition 5.3, the expected number of cluster-sending steps is upper bounded by $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2})$. In the worst case, we have $n = 2f + 1$ with $f = \mathbf{f}_{S_1} = \mathbf{f}_{S_2}$. Hence, the expected number of cluster-sending steps is upper bounded by $\mathbb{E}(2f+1, f, f)$, $f \geq 0$. We claim that $\mathbb{E}(2f+1, f, f)$ simplifies to $\mathbb{E}(2f+1, f, f) = 4 - 2/(f+1) - f!^2/(2f)!$. Hence, for all S_1 and S_2, we have $\mathbb{E}(n, \mathbf{f}_{S_1}, \mathbf{f}_{S_2}) < 4$. An analogous argument can be used to prove the other upper bounds. □

Note that the third case of Theorem 5.7 corresponds with cluster-sending between arbitrary-sized resilient clusters that each operate using Byzantine fault-tolerant consensus protocols.

Remark 5.8. The upper bounds on the expected-case complexity of instances of CSPL presented in Theorem 5.7 match the upper bounds for CSP presented in Corollary 4.4. This does not imply that the expected-case complexity for these protocols is the same, however, as the probability distributions that yield these expected-case complexities are very different. To see this, consider a system in which all clusters have n replicas of which f, $n = 2f + 1$, are faulty. Next, we denote the expected number of cluster-sending steps of protocol P by $\mathbf{E}_P$, and we have

$$\mathbf{E}_{\text{CSP}} = \frac{(2f+1)^2}{(f+1)^2} = 4 - \frac{4f+3}{(f+1)^2};$$

$$\mathbf{E}_{\text{CSPL}} = \mathbb{E}(2f+1, f, f) = 4 - \frac{2}{(f+1)} - \frac{f!^2}{(2f)!}.$$

In Figure 6, we have illustrated this difference by plotting the expected-case complexity of CSP and CSPL for systems with equal-sized clusters. In practice, we see that the expected-case complexity for CSP is slightly lower than the expected-case complexity for CSPL.

6 Asynchronous Communication

In the previous sections, we introduced CSP, CSPP, and CSPL, three probabilistic cluster-sending protocols with expected constant message complexity. To simplify presentation, we have presented their design with respect to a synchronous environment. Next, we consider their usage in environments with asynchronous inter-cluster communication due to which messages can get arbitrary delayed, duplicated, or dropped.

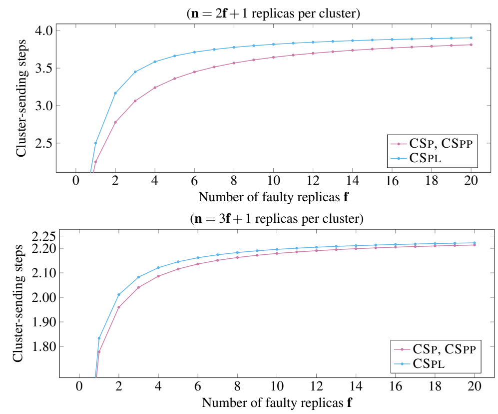

Figure 6: Comparison of the expected-case complexity of CSPL and CSP as a function of the number of faulty replicas.

We notice that the presented protocols *only* depend on synchronous communication to minimize communication: at the core of the correctness of CSP, CSPP, and CSPL is the cluster-sending step performed by CS-STEP, which does not make any assumptions on communication (Proposition 3.1). Consequently, CSP, CSPP, and CSPL can easily be generalized to operate in environments with asynchronous communication:

1. First, we observe that message duplication and out-of-order delivery have no impact on the cluster-sending step performed by CS-STEP. Hence, we do not need to take precautions against such asynchronous behavior.

2. If communication is asynchronous, but reliable (messages do not get lost, but can get duplicated, be delivered out-of-order, or get arbitrarily delayed), both CSPP and CSPL will be able to always perform cluster-sending in a finite number of steps. If communication becomes unreliable, however, messages sent between non-faulty replicas can get lost and all cluster-sending steps can fail. To deal with this, replicas in $\mathcal{C}_1$ simply continue cluster-sending steps until a step succeeds (CSP) or rerun the protocol until a step succeeds (CSPP, and CSPL), which will eventually happen in an expected constant number steps whenever communication becomes reliable again.

3. If communication is asynchronous, then messages can get arbitrarily delayed. Fortunately, practical environments operate with large periods of reliable communication in which the majority of the messages arrive within some bounded delay unknown to $\mathcal{C}_1$ and $\mathcal{C}_2$. Hence, replicas in $\mathcal{C}_1$ can simply assume some delay δ. If this delay is too short, then a cluster-sending step can *appear to fail* simply because the proof of receipt is still under way. In this case, cluster-sending will still be achieved when the proof of receipt arrives, but spurious cluster-sending steps can be initiated in the meantime. To reduce the number of such spurious cluster-sending steps, all

non-faulty replicas in $\mathcal{C}_1$ can use *exponential backup* to increase the message delay δ up-to-some reasonable upper bound (e.g., 100 s).

4. Finally, asynchronous environments often necessitate rather high assumptions on the message delay δ. Consequently, the duration of a single failed cluster-sending step performed by CS-STEP will be high. Here, a trade-off can be made between *message complexity* and *duration* by starting several rounds of the cluster-sending step at once. E.g., when communication is sufficiently reliable, then all three protocols are expected to finish in four rounds or less, due to which starting four rounds initially will sharply reduce the duration of the protocol with only a constant increase in expected message complexity.

7 Performance evaluation

In the previous sections, we introduced probabilistic cluster-sending protocols with expected-case constant message complexity. To gain further insight in the performance attainable by these protocols, especially in environments with unreliable communication, we implemented these protocols in a simulated sharded resilient environment that allows us to control the faulty replicas and the message loss rates.[1] As a baseline of comparison, we also evaluated three cluster-sending protocols from the literature:

1. The *worst-case optimal cluster-sending protocol* PBS-CS of Hellings et al. [17] that can perform cluster-sending using only $\mathbf{f}_{\mathcal{C}_1} + \mathbf{f}_{\mathcal{C}_2} + 1$ messages, which is worst-case optimal. This protocol requires reliable communication.

2. The *broadcast-based cluster-sending protocol* of CHAINSPACE [1] that can perform cluster-sending using $\mathbf{n}_{\mathcal{C}_1}\mathbf{n}_{\mathcal{C}_2}$ messages. This protocol requires reliable communication.

3. The *global sharing protocol* of GEOBFT [15], an optimistic cluster-sending protocol that assumes that each cluster uses a primary-backup consensus protocol (e.g., PBFT [6]) and optimizes for the case in which the coordinating primary of $\mathcal{C}_1$ is non-faulty. In this optimistic case, GEOBFT can perform cluster-sending using only $\mathbf{f}_{\mathcal{C}_2} + 1$ messages. To deal with faulty primaries and unreliable communication, GEOBFT employs a costly remote view-change protocol, however.

We refer to Figure 2 for an analytical comparison between these three cluster-sending protocols and our three probabilistic cluster-sending protocols.

[1] The full implementation of this experiment is available at `anonymized`.

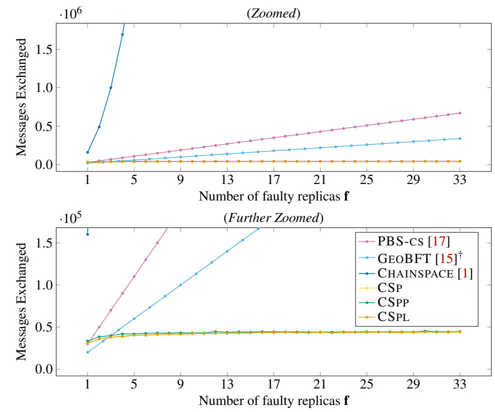

Figure 7: A comparison of the number of message exchange steps as a function of the number of faulty replicas in both clusters by our probabilistic cluster-sending protocols CSP, CSPP, and CSPL, and by three protocols from the literature. For each protocol, we measured the number of message exchange steps to send 10 000 values between two equally-sized clusters, each cluster having $n = 3\mathbf{f} + 1$ replicas. †The results for GEOBFT are a plot of the best-case optimistic phase of that protocol.

In each experiment, we measured the number of messages exchanged in 10 000 runs of the cluster-sending protocol under consideration. In specific, in each run we measure the number of messages exchanged when sending a value v from a cluster $\mathcal{C}_1$ to a cluster $\mathcal{C}_2$ with $\mathbf{n}_{\mathcal{C}_1} = \mathbf{n}_{\mathcal{C}_2} = 3\mathbf{f}_{\mathcal{C}_1} + 1 = 3\mathbf{f}_{\mathcal{C}_2} + 1$, and we aggregate this data over 10 000 runs. As we use equal-sized clusters, we have $\Phi_{\min}(\mathcal{C}_1, \mathcal{C}_2) = \Phi_{\max}(\mathcal{C}_1, \mathcal{C}_2)$ and, hence, we use a singe instance of CSPL.

Next, we detail the two experiments we performed and look at their results.

7.1 Performance of Cluster-Sending Protocols

In our first experiment, we measure the number of messages exchanged as a function of the number of faulty replicas. In this case, we assumed reliable communication, due to which we could include all six protocols. The results of this experiment can be found in Figure 7.

As is clear from the results, our probabilistic cluster-sending protocols are able to perform cluster-sending with only a constant number of messages exchanged. Furthermore, we see that the performance of our cluster-sending protocols matches the theoretical expected-case analysis in this paper and closely follows the expected performance illustrated in Figure 6 (note that Figure 6 plots cluster-sending steps and each cluster-sending step involves the exchange of *two* messages between clusters).

As all other cluster-sending protocols have a linear (PBS-CS and GEOBFT) or quadratic (CHAINSPACE) message complexity, our probabilistic cluster-sending protocols outperform the other cluster-sending protocols. This is especially the case when dealing with bigger clusters, in which case the expected-case constant message complexity of our probabilistic cluster-sending protocols shows the biggest advantage. Only in the case of the smallest clusters can the other cluster-sending protocols outperform our probabilistic cluster-sending protocols, as PBS-CS, GEOBFT, and CHAINSPACE use reliable communication to their advantage to eliminate any acknowledgment messages send from the receiving cluster to the sending cluster. We believe that the slightly higher cost of our probabilistic cluster-sending protocols in these cases is justified, as our protocols can effectively deal with unreliable communication.

7.2 Message Loss

In our second experiment, we measure the number of messages exchanged as a function of the number of faulty replicas and as a function of the message loss (in percent) *between the two clusters*. We assume that communication within each cluster is reliable. In this case, we only included our probabilistic cluster-sending protocols as PBS-CS and CHAINSPACE both assume reliable communication and GEOBFT is only able to perform recovery via remote view-changes in periods of reliable communication. The results of this experiment can be found in Figure 8.

We note that with a message loss of $x\%$, the probability $p(x)$ of a successful cluster-sending step is only $(1-\frac{x}{100})^2$. E.g., $p(30\%) \approx 0.49$. As expected, the message complexity increases with an increase in message loss. Furthermore, the probabilistic cluster-sending protocols perform as expected (when taking into account the added cost to deal with message loss). These results further underline the practical benefits of each of the probabilistic cluster-sending protocols, especially for larger clusters: even in the case of high message loss rates, each of our probabilistic cluster-sending protocols are able to outperform the cluster-sending protocols PBS-CS, CHAINSPACE, and GEOBFT, which can only operate with reliable-communication.

8 Related Work

Although there is abundant literature on distributed systems and on consensus-based resilient systems (e.g., [2, 5, 8, 14, 16, 27, 31]), there is only limited work on communication *between* resilient systems [1, 15, 17]. In the previous section, we have already compared CSP, CSPP, and CSPL with the worst-case optimal cluster-sending protocols of Hellings et al. [17], the optimistic cluster-sending protocol of GEOBFT [15], and the broadcast-based cluster-sending protocols of CHAINSPACE [1]. Furthermore, we notice that *cluster-sending* can be solved using well-known Byzantine primitives such as consensus, interactive consistency, and Byzantine broadcasts [6, 9, 24]. These primitives are much more costly than cluster-sending protocols, however, and require huge amounts of communication between all involved replicas.

In parallel to the development of traditional resilient systems and permissioned blockchains, there has been promising work on sharding in permissionless blockchains such as BITCOIN [25] and ETHEREUM [32]. Examples include techniques for enabling reliable cross-chain coordination via sidechains, blockchain relays, atomic swaps, atomic commitment, and cross-chain deals [12, 13, 19, 21, 22, 33, 34]. Unfortunately, these techniques are deeply intertwined with the design goals of permissionless blockchains in mind (e.g., cryptocurrency-oriented), and are not readily applicable to traditional consensus-based Byzantine clusters.

9 Conclusion

In this paper, we presented probabilistic cluster-sending protocols that each provide highly-efficient solutions to the cluster-sending problem. In specific, our probabilistic cluster-sending protocols can facilitate communication between Byzantine fault-tolerant clusters with expected constant communication between clusters. For practical environments, our protocols can support worst-case linear communication between clusters, which is optimal, and deal with asynchronous and unreliable communication. The low practical cost of our cluster-sending protocols further enables the development and deployment of high-performance systems that are constructed out of Byzantine fault-tolerant clusters, e.g., fault-resilient geo-aware sharded data processing systems.

References

[1] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform, 2017. URL: http://arxiv.org/abs/1708.03778.

[2] Christian Berger and Hans P. Reiser. Scaling byzantine consensus: A broad analysis. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, pages 13–18. ACM, 2018. https://doi.org/10.1145/3284764.3284767.

[3] Gabi Bracha and Ophir Rachman. Randomized consensus in expected $O((n^2 \log n))$ operations. In *Distributed Algorithms*, pages 143–150. Springer Berlin Heidelberg, 1992. https://doi.org/10.1007/BFb0022443.

[4] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology — CRYPTO 2001*, pages 524–541. Springer Berlin Heidelberg, 2001. https://doi.org/10.1007/3-540-44647-8_31.

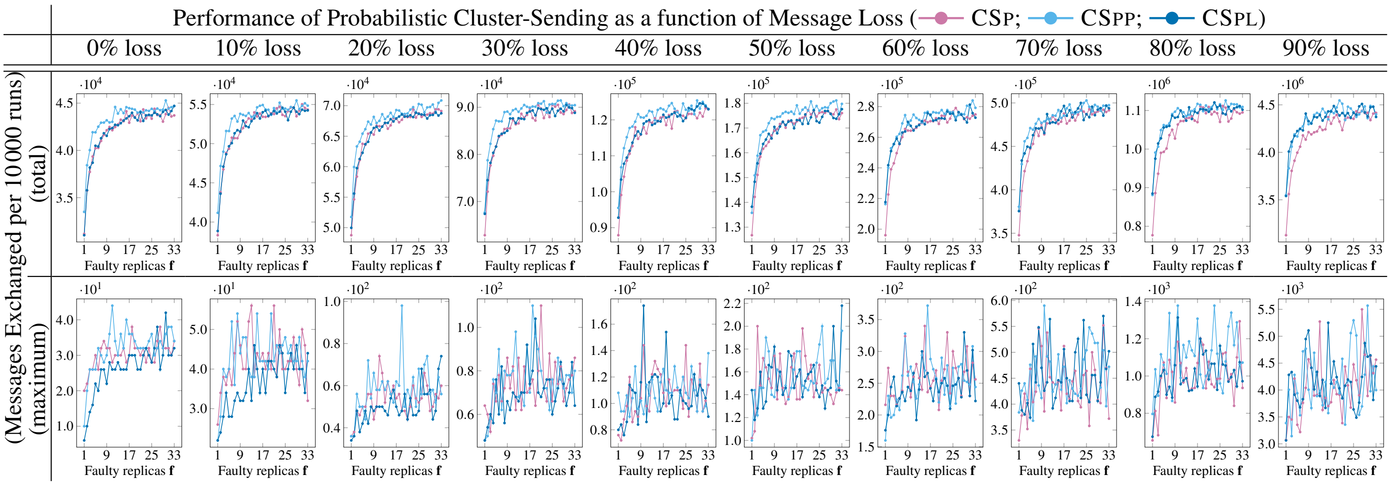

Figure 8: A comparison of the number of message exchange steps as a function of the number of faulty replicas in both clusters and of the message loss by our probabilistic cluster-sending protocols CSP, CSPP, and CSPL. For each protocol, we measured the number of message exchange steps to send 10 000 values between two equally-sized clusters, each cluster having $n = 3\mathbf{f} + 1$ replicas. At the *top* row, we have plotted the total number of messages exchanged in the 10 000 runs, and at the *bottom* row,we have plotted the maximum number of messages exchanged in any of the 10 000 runs.

[5] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild (keynote talk). In *31st International Symposium on Distributed Computing*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. https://doi.org/10.4230/LIPIcs.DISC.2017.1.

[6] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002. https://doi.org/10.1145/571637.571640.

[7] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, pages 123–140. ACM, 2019. https://doi.org/10.1145/3299869.3319889.

[8] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Trans. Knowl. Data Eng.*, 30(7):1366–1385, 2018. https://doi.org/10.1109/TKDE.2017.2781227.

[9] D. Dolev and H. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. https://doi.org/10.1137/0212045.

[10] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77—-97, 1987. https://doi.org/10.1145/7531.7533.

[11] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288—323, 1988. https://doi.org/10.1145/42282.42283.

[12] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. BlockchainDB: A shared database on blockchains. *Proc. VLDB Endow.*, 12(11):1597–1609, 2019. https://doi.org/10.14778/3342263.3342636.

[13] Ethereum Foundation. BTC Relay: A bridge between the bitcoin blockchain & ethereum smart contracts, 2017. URL: http://btcrelay.org.

[14] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. *Fault-Tolerant Distributed Transactions on Blockchain*. Synthesis Lectures on Data Management. Morgan & Claypool, 2021. https://doi.org/10.2200/S01068ED1V01Y202012DTM065.

[15] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. ResilientDB: Global scale resilient blockchain fabric. *Proceedings of the VLDB Endowment*, 13(6):868—-883, 2020. https://doi.org/10.14778/3380750.3380757.

[16] Suyash Gupta and Mohammad Sadoghi. *Blockchain Transaction Processing*, pages 1–11. Springer International Publishing, 2018. https://doi.org/10.1007/978-3-319-63962-8_333-1.

[17] Jelle Hellings and Mohammad Sadoghi. Brief announcement: The fault-tolerant cluster-sending problem. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:3. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. https://doi.org/10.4230/LIPIcs.DISC.2019.45.

[18] Jelle Hellings and Mohammad Sadoghi. Byshard: Sharding in a byzantine environment. *Proceedings of the VLDB Endowment*, 14(11):2230–2243, 2021. https://doi.org/10.14778/3476249.3476275.

[19] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 245—254. ACM, 2018. https://doi.org/10.1145/3212734.3212736.

[20] Maurice Herlihy. Blockchains from a distributed computing perspective. *Communications of the ACM*, 62(2):78–85, 2019. https://doi.org/10.1145/3209623.

[21] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. Cross-chain deals and adversarial commerce. *The VLDB Journal*, 2021. https://doi.org/10.1007/s00778-021-00686-1.

[22] Jae Kwon and Ethan Buchman. Cosmos whitepaper: A network of distributed ledgers, 2019. URL: https://cosmos.network/cosmos-whitepaper.pdf.

[23] Leslie Lamport. Paxos made simple. *ACM SIGACT News, Distributed Computing Column 5*, 32(4):51–58, 2001. https://doi.org/10.1145/568425.568433.

[24] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. https://doi.org/10.1145/357172.357176.

[25] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. URL: https://bitcoin.org/en/bitcoin-paper.

[26] Arvind Narayanan and Jeremy Clark. Bitcoin's academic pedigree. *Communications of the ACM*, 60(12):36–45, 2017. https://doi.org/10.1145/3132259.

[27] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 2020. https://doi.org/10.1007/978-3-030-26253-2.

[28] Michael Pisa and Matt Juden. Blockchain and economic development: Hype vs. reality. Technical report, Center for Global Development, 2017. URL: https://www.cgdev.org/publication/blockchain-and-economic-development-hype-vs-reality.

[29] David Reinsel, John Gantz, and John Rydning. Data age 2025: The digitization of the world, from edge to core. Technical report, IDC, 2018. URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[30] Victor Shoup. Practical threshold signatures. In *Advances in Cryptology — EUROCRYPT 2000*, pages 207–220. Springer Berlin Heidelberg, 2000. https://doi.org/10.1007/3-540-45539-6_15.

[31] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2001.

[32] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. EIP-150 revision. URL: https://gavwood.com/paper.pdf.

[33] Gavin Wood. Polkadot: vision for a heterogeneous multi-chain framework, 2016. URL: https://polkadot.network/PolkaDotPaper.pdf.

[34] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. Atomic commitment across blockchains. *Proc. VLDB Endow.*, 13(9):1319–1331, 2020. https://doi.org/10.14778/3397230.3397231.

A The proof of Lemma 5.2

To get the intuition behind the closed form of Lemma 5.2, we take a quick look at the combinatorics of *list-merging*. Notice that we can merge lists S and T together by either first taking an element from S or first taking an element from T. This approach towards list-merging yields the following recursive solution to the list-merge problem:

$$\mathbb{M}(v,w) = \begin{cases} \mathbb{M}(v-1,w)+\mathbb{M}(v,w-1) & \text{if } v>0 \text{ and } w>0; \\ 1 & \text{if } v=0 \text{ or } w=0. \end{cases}$$

Consider lists S and T with $|S| = v$ and $|T| = w$ distinct values. We have $|\mathsf{perms}(S)| = v!$, $|\mathsf{perms}(T)| = w!$, and $|\mathsf{perms}(S \cup T)| = (v+w)!$. We observe that every list-merge of $(P_S, P_T) \in \mathsf{perms}(S) \times \mathsf{perms}(T)$ is a unique value in $\mathsf{perms}(S \cup T)$. Furthermore, every value in $\mathsf{perms}(S \cup T)$ can be constructed by such a list-merge. As we have $|\mathsf{perms}(S) \times \mathsf{perms}(T)| = v!w!$, we derive the closed form

$$\mathbb{M}(v,w) = \frac{(v+w)!}{(v!w!)}$$

of Lemma 5.2. Next, we formally prove this closed form.

Proof. We prove this by induction. First, the base cases $\mathbb{M}(0,w)$ and $\mathbb{M}(v,0)$. We have

$$\mathbb{M}(0,w) = \frac{(0+w)!}{0!w!} = \frac{w!}{w!} = 1;$$
$$\mathbb{M}(v,0) = \frac{(v+0)!}{v!0!} = \frac{v!}{v!} = 1.$$

Next, we assume that the statement of the lemma holds for all non-negative integers v', w' with $0 \leq v' + w' \leq j$. Now consider non-negative integers v, w with $v + w = j + 1$. We assume that $v > 0$ and $w > 0$, as otherwise one of the base cases applies. Hence, we have

$$\mathbb{M}(v,w) = \mathbb{M}(v-1,w) + \mathbb{M}(v,w-1).$$

We apply the induction hypothesis on the terms $\mathbb{M}(v-1,w)$ and $\mathbb{M}(v,w-1)$ and obtain

$$\mathbb{M}(v,w) = \left(\frac{((v-1)+w)!}{(v-1)!w!}\right) + \left(\frac{(v+(w-1))!}{v!(w-1)!}\right).$$

Next, we apply $x = x(x-1)!$ and simplify the result to obtain

$$\mathbb{M}(v,w) = \left(\frac{v(v+w-1)!}{v!w!}\right) + \left(\frac{w(v+w-1)!}{v!w!}\right)$$
$$= \left(\frac{(v+w)(v+w-1)!}{v!w!}\right) = \frac{(v+w)!}{v!w!},$$

which completes the proof. □

B The proof of Lemma 5.3

Let g be the expression

$$b_1!^2\binom{m_1}{b_1}\binom{n-m_2}{b_1}b_2!^2\binom{n-m_1}{b_2}\binom{m_2}{b_2}\cdot$$
$$\mathbb{M}(b_1,b_2)b_{1,2}!^2\mathbb{M}(b_1+b_2,b_{1,2})(n-k)!^2\mathbb{M}(k,n-k),$$

as stated right above Lemma 5.3. We will show that g is equivalent to the closed form of $\mathbb{F}(n,m_1,m_2,k)$, as stated in Lemma 5.3.

Proof. We use the shorthands $\mathbf{T}_1 = \binom{m_1}{b_1}\binom{n-m_2}{b_1}$ and $\mathbf{T}_2 = \binom{n-m_1}{b_2}\binom{m_2}{b_2}$, and we have

$$g = b_1!^2\mathbf{T}_1 b_2!^2\mathbf{T}_2\cdot$$
$$\mathbb{M}(b_1,b_2)b_{1,2}!^2\mathbb{M}(b_1+b_2,b_{1,2})(n-k)!^2\mathbb{M}(k,n-k).$$

We apply Lemma 5.2 on terms $\mathbb{M}(b_1,b_2)$, $\mathbb{M}(b_1+b_2,b_{1,2})$, and $\mathbb{M}(k,n-k)$, apply $k = b_1+b_2+b_{1,2}$, and simplify to derive

$$g = b_1!^2\mathbf{T}_1 b_2!^2\mathbf{T}_2\cdot$$
$$\frac{(b_1+b_2)!}{b_1!b_2!}b_{1,2}!^2\frac{(b_1+b_2+b_{1,2})!}{(b_1+b_2)!b_{1,2}!}(n-k)!^2\frac{(k+n-k)!}{k!(n-k)!}$$
$$= b_1!\mathbf{T}_1 b_2!\mathbf{T}_2 b_{1,2}!(n-k)!n!.$$

Finally, we expand the binomial terms $\mathbf{T}_1$ and $\mathbf{T}_2$, apply $b_{1,2} = m_1 - b_1 = m_2 - b_2$ and $k = m_1 + b_2 = m_2 + b_1$, and simplify to derive

$$g = b_1!\frac{m_1!}{b_1!(m_1-b_1)!}\frac{(n-m_2)!}{b_1!(n-m_2-b_1)!}\cdot$$
$$b_2!\frac{(n-m_1)!}{b_2!(n-m_1-b_2)!}\frac{m_2!}{b_2!(m_2-b_2)!}\cdot$$
$$b_{1,2}!(n-k)!n!$$
$$= \frac{m_1!}{b_{1,2}!}\frac{(n-m_2)!}{b_1!(n-k)!}\frac{(n-m_1)!}{b_2!(n-k)!}\frac{m_2!}{b_{1,2}!}b_{1,2}!(n-k)!n!$$
$$= \frac{m_1!m_2!(n-m_1)!(n-m_2)!n!}{b_1!b_2!b_{1,2}!(n-k)!},$$

which completes the proof. □

C The proof of Lemma 5.3

Proof. We write $f(n,m_1,m_2,k)$ for the closed form in the statement of this lemma and we prove the statement of this lemma by induction. First, the base case $\mathbb{F}(0,0,0,0)$. In this case, we have $n = m_1 = m_2 = k = 0$ and, hence, $b_1 = b_2 = b_{1,2} = 0$, and we conclude $f(0,0,0,0) = 1 = \mathbb{F}(0,0,0,0)$.

Now assume $\mathbb{F}(n',m_1',m_2',k') = f(n',m_1',m_2',k')$ for all $n' < n$ and all k' with $\max(m_1',m_2') \leq k' \leq \min(n',m_1'+m_2')$. Next, we prove $\mathbb{F}(n,m_1,m_2,k) = f(n,m_1,m_2,k)$ with $\max(m_1,m_2) \leq k \leq \min(n,m_1+m_2)$. We use the shorthand $\mathbb{G} = \mathbb{F}(n,m_1,m_2,k)$ and we have

$$\begin{aligned}\mathbb{G} = {} & (n-m_1)(n-m_2)\mathbb{F}(n-1,m_1,m_2,k) && \text{(non-faulty pair)}\\ & + m_1(n-m_2)\mathbb{F}(n-1,m_1-1,m_2,k-1) && \text{(1-faulty pair)}\\ & + (n-m_1)m_2\mathbb{F}(n-1,m_1,m_2-1,k-1) && \text{(2-faulty pair)}\\ & + m_1m_2\mathbb{F}(n-1,m_1-1,m_2-1,k-1). && \text{(both-faulty pair)}\end{aligned}$$

Notice that if $n = k$, then the non-faulty pair case does not apply, as $\mathbb{F}(n-1,m_1,m_2,k) = 0$, and evaluates to zero. Likewise, if $b_1 = 0$, then the 1-faulty pair case does not apply, as $\mathbb{F}(n-1,m_1-1,m_2,k-1) = 0$, and evaluates to zero; if $b_2 = 0$, then the 2-faulty pair case does not apply, as $\mathbb{F}(n-1,m_1,m_2-1,k-1) = 0$, and evaluates to zero; and, finally, if $b_{1,2} = 0$, then the both-faulty pair case does not apply, as $\mathbb{F}(n-1,m_1-1,m_2-1,k-1) = 0$, and evaluates to zero.

First, we consider the case in which $n > k$, $b_1 > 0$, $b_2 > 0$, and $b_{1,2} > 0$. Hence, each of the four cases apply and evaluate to non-zero values. We directly apply the induction hypothesis on $\mathbb{F}(n-1,m_1,m_2,k)$, $\mathbb{F}(n-1,m_1-1,m_2,k-1)$,

$\mathbb{F}(n-1,m_1,m_2-1,k-1)$, and $\mathbb{F}(n-1,m_1-1,m_2-1,k-1)$, and obtain

$$\begin{aligned}\mathbb{G} = (n-m_1)(n-m_2)\cdot \\ &\frac{m_1!m_2!(n-1-m_1)!(n-1-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-1-k)!} \\ &+m_1(n-m_2)\cdot \\ &\frac{(m_1-1)!m_2!(n-m_1)!(n-1-m_2)!(n-1)!}{(b_1-1)!b_2!b_{1,2}!(n-1-(k-1))!} \\ &+(n-m_1)m_2\cdot \\ &\frac{m_1!(m_2-1)!(n-1-m_1)!(n-m_2)!(n-1)!}{b_1!(b_2-1)!b_{1,2}!(n-1-(k-1))!} \\ &+m_1m_2\cdot \\ &\frac{(m_1-1)!(m_2-1)!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!(b_{1,2}-1)!(n-1-(k-1))!}.\end{aligned}$$

We apply $x! = x(x-1)!$ and further simplify and obtain

$$\begin{aligned}\mathbb{G} &= \frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-1-k)!} \\ &+\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{(b_1-1)!b_2!b_{1,2}!(n-k)!} \\ &+\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!(b_2-1)!b_{1,2}!(n-k)!} \\ &+\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!(b_{1,2}-1)!(n-k)!} \\ &=(n-k)\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\ &+b_1\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\ &+b_2\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\ &+b_{1,2}\frac{m-1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!}.\end{aligned}$$

We have $k = b_1+b_2+b_{1,2}$ and, hence, $n = (n-k)+b_1+b_2+b_{1,2}$ and we conclude

$$\begin{aligned}\mathbb{G} &= ((n-k)+b_1+b_2+b_{1,2})\cdot \\ &\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\ &=n\frac{m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!} \\ &=\frac{m_1!m_2!(n-m_1)!(n-m_2)!n!}{b_1!b_2!b_{1,2}!(n-k)!}.\end{aligned}$$

Next, in all other cases, we can repeat the above derivation while removing the terms corresponding to the cases that evaluate to 0. By doing so, we end up with the expression

$$\mathbb{G} = \frac{((\sum_{t\in T}\ t)m_1!m_2!(n-m_1)!(n-m_2)!(n-1)!}{b_1!b_2!b_{1,2}!(n-k)!}.$$

in which T contains the term $(n-k)$ if $n > k$ (the non-faulty pair case applies), the term b_1 if $b_1 > 0$ (the 1-faulty case applies), the term b_2 if $b_2 > 0$ (the 2-faulty case applies), and the term $b_{1,2}$ if $b_{1,2} > 0$ (the both-faulty case applies). As each term $(n-k)$, b_1, b_2, and $b_{1,2}$ is in T whenever the term is non-zero, we have $\sum_{t\in T}\ t = (n-k)+b_1+b_2+b_{1,2} = n$. Hence, we can repeat the steps of the above derivation in all cases, and complete the proof. □

D The Closed Form of $\mathbb{E}(2f+1,f,f)$

Here, we shall prove that

$$\mathbb{E}(2f+1,f,f) = 4-\frac{2}{(f+1)}-\frac{f!^2}{(2f)!}.$$

Proof. By Proposition 5.2 and some simplifications, we have

$$\begin{aligned}\mathbb{E}(2f+1,f,f) &= \frac{1}{(2f+1)!^2}\cdot \\ &\left(\sum_{k=f}^{2f}\frac{2f+1}{2f+1-k}\frac{f!^2(f+1)!^2(2f+1)!}{(k-f)!^2(2f-k)!(2f+1-k)!}\right).\end{aligned}$$

First, we apply $x! = x(x-1)!$, simplify, and obtain

$$\begin{aligned}\mathbb{E}(2f+1,f,f) &= \frac{f!^2(2f+1)}{(2f+1)!}\cdot \\ &\left(\sum_{k=f}^{2f}\frac{(f+1)!^2}{(k-f)!^2(2f+1-k)!^2}\right) \\ &=\frac{f!^2}{(2f)!}\left(\sum_{k=0}^{f}\frac{(f+1)!^2}{k!^2(f+1-k)!^2}\right) \\ &=\frac{f!^2}{(2f)!}\left(\sum_{k=0}^{f}\binom{f+1}{k}^2\right).\end{aligned}$$

Next, we apply $\binom{m}{n} = \binom{m}{m-n}$, extend the sum by one term, and obtain

$$\begin{aligned}\mathbb{E}(2f+1,f,f) &= \frac{f!^2}{(2f)!}\cdot \\ &\left(\left(\sum_{k=0}^{f+1}\binom{f+1}{k}\binom{f+1}{f+1-k}\right)-\binom{f+1}{f+1}\binom{f+1}{0}\right).\end{aligned}$$

Then, we apply Vandermonde's Identity to eliminate the sum and obtain

$$\mathbb{E}(2f+1,f,f) = \frac{f!^2}{(2f)!}\left(\binom{2f+2}{f+1}-1\right).$$

Finally, we apply straightforward simplifications and obtain

$$\begin{aligned}
\mathbb{E}(2f+1,f,f) &= \frac{f!^2}{(2f)!}\frac{(2f+2)!}{(f+1)!(f+1)!} - \frac{f!^2}{(2f)!} \\
&= \frac{f!^2}{(2f)!}\frac{(2f)!(2f+1)(2f+2)}{f!^2(f+1)^2} - \frac{f!^2}{(2f)!} \\
&= \frac{(2f+1)(2f+2)}{(f+1)^2} - \frac{f!^2}{(2f)!} \\
&= \frac{(2f+2)^2}{(f+1)^2} - \frac{2f+2}{(f+1)^2} - \frac{f!^2}{(2f)!} \\
&= \frac{4(f+1)^2}{(f+1)^2} - \frac{2(f+1)}{(f+1)^2} - \frac{f!^2}{(2f)!} \\
&= 4 - \frac{2}{f+1} - \frac{f!^2}{(2f)!},
\end{aligned}$$

which completes the proof. □