

# Hard Shell, Reliable Core: Improving Resilience in Replicated Systems with Selective Hybridization

Laura Lawniczak and Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Email: {lawniczak,distler}@cs.fau.de

**Abstract**—Hybrid fault models are known to be an effective means for enhancing the robustness of consensus-based replicated systems. However, existing hybridization approaches suffer from limited flexibility with regard to the composition of crash-tolerant and Byzantine fault-tolerant system parts and/or are associated with a significant diversification overhead. In this paper we address these issues with SHELLFT, a framework that leverages the concept of micro replication to allow system designers to freely choose the parts of the replication logic that need to be resilient against Byzantine faults. As a key benefit, such a selective hybridization makes it possible to develop hybrid solutions that are tailored to the specific characteristics and requirements of individual use cases. To illustrate this flexibility, we present three custom SHELLFT protocols and analyze the complexity of their implementations. Our evaluation shows that compared with traditional hybridization approaches, SHELLFT is able to decrease diversification costs by more than 70%.

## I. INTRODUCTION

Although a variety of causes such as software bugs, hardware errors, or malicious attacks potentially result in arbitrary behavior of components, crash-tolerant state-machine replication protocols [1], [2] today still are the norm in many practical use cases. Full-fledged Byzantine fault-tolerant (BFT) protocols [3], [4] could offer improvements with regard to resilience, however they are often considered too expensive, both in terms of complexity as well as resource consumption [5]–[7]. As a consequence, in recent years the use of hybrid fault models has drawn a significant amount of attention due to offering a tradeoff between both worlds [8]–[11].

In a nutshell, the fundamental idea of hybridization is to combine different fault assumptions within a single system. For replication protocols, this can for example mean to only tolerate crashes in certain (trusted) components while being resilient against Byzantine faults in other (untrusted) parts [12], [13], or to have distinct resilience thresholds for different classes of faults [14], [15]. Unfortunately, despite their effectiveness, existing hybridization approaches have at least one of two drawbacks: (1) With the partitioning of trusted/untrusted areas typically being dictated by the protocol design, and resilience thresholds generally applying to the replicated system as a whole, it is inherently difficult to adjust them to new use cases with different demands. (2) Requiring each replica to perform a complex set of tasks, they commonly involve a large overhead when it comes to diversifying the replica logic (e.g., using N-version programming [16]).

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 541017677, 446811880.

In an effort to address these issues we present SHELLFT, a novel approach for the hybridization of replicated systems that offers developers an unprecedented degree of flexibility and significantly reduces the costs for diversification. SHELLFT builds on the observations that (1) with regard to overall system robustness, some mechanisms of a replication protocol are more critical or more vulnerable than others, and that (2) this subset of crucial mechanisms usually depends on the specific use-case scenario of a replicated system. For example, for user-facing services that are directly accessible from the Internet, the communication with clients (if not protected properly) can be used as a gateway for attacks. On the other hand, if a replicated system only serves trusted clients (e.g., due to hosting a lower-tier service inside a data center), the leader-replica functionality is often the pivotal part impacting system stability [17]. Taking into account these insights, SHELLFT offers system developers the possibility of deliberately choosing the replication-protocol components and mechanisms that need to be protected against Byzantine faults. As a key benefit, this *selective hybridization* significantly increases overall system resilience at only small additional costs.

To achieve this, SHELLFT leverages the fine-grained modularization of a replication protocol into clusters of tiny components that each are responsible for a dedicated protocol step. More specifically, the fact that these clusters are largely independent of each other allows SHELLFT to treat every cluster as a separate domain that has its own fault model. Among other things, this for example makes it possible to selectively tolerate a (predefined) maximum number of Byzantine faults in a particular protocol mechanism without the need to tighten overall synchrony assumptions. Finally, as an additional advantage, the clustering enables a targeted diversification of critical parts which, in contrast to the diversification of entire replicas in traditional hybrid architectures, in a SHELLFT protocol is inexpensive due to being limited to individual protocol steps.

In particular, this paper makes the following contributions: (1) It introduces SHELLFT, an approach to improve the resilience of state-machine replication protocols by applying selective hybridization. (2) It provides details on the SHELLFT framework, a tool that assists system developers by configuring our SHELLFT codebase depending on their choices of fault domains. (3) It illustrates the flexibility of SHELLFT by discussing three custom protocols that are tailored to specific use cases. (4) It evaluates the three SHELLFT protocols with regard to complexity, diversification costs, and performance.

## II. BACKGROUND AND PROBLEM STATEMENT

In this section, we provide necessary background and highlight the differences between the state of the art and SHELLFT.

### A. State-Machine Replication Architectures

State-machine replication [18] achieves fault tolerance by hosting multiple instances of the same service on different machines (see Figure 1, left side). To ensure consistency, the replicas run an agreement protocol [1], [3] that establishes a total order on incoming client requests, and in addition performs tasks such as view change and checkpointing. Traditionally, this multifaceted set of responsibilities is handled by a group of monolithic replicas, however several works have shown that there are alternatives to this basic design. Specifically, a number of architectures have been proposed that are separating agreement from execution [19], [20], possibly adding a third stage for the reception of requests [14]. Although reducing the functionality required by individual nodes compared with the traditional monolithic approach, the scopes of replicas in these architectures still consist of entire protocol stages, including for example the whole consensus process. Among other things, this leads to significant overhead for the implementation of heterogeneous variants in cases in which N-version programming [16] is applied for diversification. To some degree, this overhead can be reduced by employing architectures that rely on compartmentalization [11], [21], however for SHELLFT we strive for an even more fine-grained diversification.

*Our Approach:* As basis for the SHELLFT protocol architecture we rely on micro replication [22], a concept that splits a replication protocol into atomic tasks (see Figure 1, right side) and hence results in replica implementations with very low complexity. This way, with each micro replica only handling a single protocol step, a tailored diversification of critical protocol parts becomes both feasible and affordable.

Compared with traditional approaches, the distribution of a protocol across a larger number of (small) replicas comes with increased configuration and deployment costs. For SHELLFT, we consider this an acceptable trade-off for the fact that micro replication offers us the flexibility to selectively increase the robustness of individual protocol steps. In general, developing a micro-replicated protocol from scratch is not always straightforward, which is why for SHELLFT we circumvent this problem by utilizing the already existing Mirador [22] protocol as starting point for our work (see Section III-B). However, since the general concept behind SHELLFT’s selective hybridization is not intrinsically linked to Mirador, it can be adapted to other micro-replicated protocols once they become available.

### B. Hybrid Fault Models

Although the failures occurring in replicated systems in practice are not only limited to crashes, this does not necessarily mean that resorting to full-fledged Byzantine fault tolerance is automatically the best option, especially when taking into account the additional costs in terms of resources and complexity [8], [23]. Leveraging this observation, several previous works have examined the use of hybrid fault models which (in

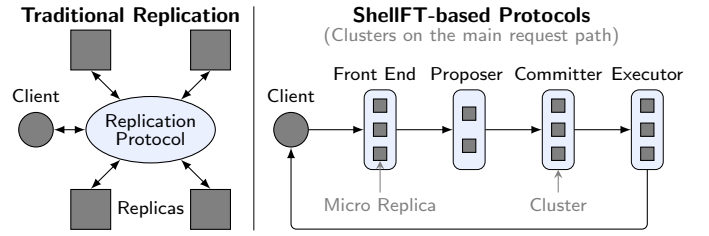


Fig. 1: State-machine replication architectures.

addition to crashes) assume Byzantine-faulty behavior only in specific system parts and/or under certain circumstances. In the following, we elaborate on various incarnations of this concept and highlight the differences to our SHELLFT approach. Notice that in the context of replicated systems the term “hybrid” has been used to describe a plethora of ideas that are orthogonal to the work presented in this paper (e.g., the design of an overall replication protocol as a composition of individual protocols with particular characteristics [24], [25], a management model with dedicated responsibilities [26], the combination of failure detection and randomization to solve consensus [27]). Thus, below we focus our discussion on approaches that are closely related to our notion of hybridization.

*Special-Purpose Trusted Components.* Applying the concept of architectural hybridization [12], the key idea is to split fault assumptions on a spatial level. Specifically, this means that some parts of a replicated system are considered to be trusted and to only fail by crashing, whereas the remaining parts may be subject to Byzantine faults. The sizes of these two areas significantly vary between systems. While some works rely on a comparably large trusted computing base (e.g., a distributed communication subsystem [12] or a virtualization layer [28], [29]), others have shown that it is possible to reduce the trusted part to relatively small components such as a log [13], a counter [5], [6], [9], [30], or other special-purpose modules implementing parts of the replication protocol [10], [11].

As a main benefit, the use of dedicated trusted components enables resource-efficient systems that tolerate up to  $f$  Byzantine faults in the untrusted areas by using  $2f+1$  replicas, which is the same as for crash-tolerant replication. On the downside, due to the dependence on special-purpose trusted components there is typically no flexibility with regard to the selection of trusted/untrusted parts. In all works mentioned above, the partitioning between trusted and untrusted areas is hardwired into the system design, thereby making it inherently difficult to harness these systems for application scenarios in which the fault and threat assumptions differ from the ones they have been developed for. As an additional drawback, designing replication protocols based on special-purpose trusted components is not straightforward, as recent examples have shown [31].

*Our Approach:* In contrast to previous works, SHELLFT offers the opportunity to freely choose the trusted and untrusted parts of a replicated system at configuration time. As illustrated in Section IV, this allows systems to be tailored to the individual fault and threat models of specific use cases.

	Hybridization		Diversification
	Type	Configurability	Granularity
MinBFT [5]	Subsystem	Hardwired	Monolithic replica
XFT [8]	Either-or	Global	Monolithic replica
UpRight [14]	Fault classes	Global	Protocol stage
VFT [15]	Fault classes	Global	Monolithic replica
SHELLFT	Fault domains	Selective	Protocol step/task

TABLE I: Comparison of hybridization approaches.

**Relaxed Synchrony Assumptions.** A major reason for the increased complexity and resource consumption of many traditional BFT protocols is rooted in their goal to tolerate Byzantine faults in the presence of an asynchronous network [3]. Several authors [8], [15] argue that for many practical use cases such a combination of assumptions is unnecessarily strong; for example, when systems comprise redundant (wide-area) communication links between replicas, and hence make it difficult for an adversary to not only control a subset of replicas but also the network. XFT [8] exploits this insight by designing state-machine replication protocols in such a way that they are able to deal with both a certain number of Byzantine faults as well as asynchrony, but not at the same time. As a main advantage, this approach makes it possible to minimize protocol complexity and keep the minimum number of required replicas to  $2f + 1$ . On the downside, if indeed the maximum number of Byzantine faults concurs with network partitions, then these kinds of replicated systems do not just lose liveness, but potentially become unsafe.

**Our Approach:** SHELLFT minimizes complexity by adding resilience to selected protocol tasks, not by trading off network asynchrony for Byzantine fault tolerance. Thus, SHELLFT protocols remain safe even in the presence of partitions.

**Distinction of Fault Classes.** Systems such as UpRight [14] distinguish between two thresholds to determine the overall number of replicas  $n = 2u + r + 1$ : a threshold  $u$ , which denotes the maximum number of tolerated faults in total (i.e., crashes plus Byzantine faults), and a threshold  $r$  representing the maximum number of tolerated Byzantine faults. VFT [15] extends this idea of handling some fault classes separately by introducing further thresholds for slow replicas and correlated faulty behavior. Similarly, other works use thresholds for disconnected sites [26] or concurrently recovering replicas [32].

Distinguishing between fault classes offers the advantage of reducing replication costs when not all expected faults are assumed to be of arbitrary nature [33]. On the other hand, as applied by UpRight and other systems, the method is rather coarse-grained due to defining fault assumptions at the level of the entire replicated system. That is, even though UpRight comprises three stages, the threshold values are cross-cutting parameters and thus rule out stage-specific configurations.

**Our Approach:** SHELLFT separates a replication protocol into loosely coupled clusters (each representing an individual protocol step) and allows each cluster to be flexibly assigned its own fault model. Within such a fault domain, applying different thresholds for different fault classes would be possible, however the specifics are outside the scope of this paper.

### C. Problem Statement

As summarized in Table I, our analysis in the previous sections has shown state-of-the-art approaches to have two main drawbacks: (1) A limited flexibility with respect to hybridization, either because the partitioning between trusted and untrusted parts is hardwired into the system design, or due to fault classes being configured globally. (2) A high diversification overhead caused by the concentration of complex functionality at replicas that are required to handle a whole protocol stage or even the entire replication protocol. In the following section, we elaborate on how SHELLFT addresses these issues by offering selective hybridization and diversification.

With regard to diversification, we especially focus on measures that introduce heterogeneity by implementing replicas in different programming languages [16], and thereby reduce the risk of vulnerabilities in the language runtime or libraries affecting multiple replicas and therefore, in the worst case, the whole replicated system. Due to the programming effort for these kinds of approaches (and hence the associated economical cost) typically depending on the complexity of the replica logic, our goal is to improve this situation by minimizing the amount of code that actually needs to be diversified in order to increase the robustness of selected protocol steps.

## III. SHELLFT

SHELLFT is both a novel concept for flexibly applying hybridization in replicated systems as well as an accompanying framework that automates the tailoring to specific use cases. In this section, we first give an overview of the main idea behind SHELLFT and then provide details on the customization.

### A. Overview

SHELLFT relies on a system architecture in which each replication-protocol task is handled by a dedicated cluster of micro replicas. As a key benefit, this partitioning allows us to treat each of these clusters as a separate fault domain with individual fault and threat model. Specifically, we distinguish between three domain types: *Shell* clusters are considered to require resilience against arbitrary faults, *core* clusters represent the crash-tolerant parts of the system, and *filter* clusters act as a barrier between these two, thereby shielding the core from the shell. More precisely, the three domain types have the following characteristics:

- **Shell:** Replicas of the clusters belonging to this type of domain may be subject to Byzantine faults. The assignment of clusters to this category is made by the user of the SHELLFT framework based on the individual properties and requirements of a protocol's application scenario.
- **Filter:** Replicas of this domain type are assumed to only fail by crashing, however they receive inputs from at least one shell cluster and therefore require means to tolerate Byzantine-faulty input values provided by these sources. The classification as filter cluster is automatically made by the SHELLFT framework based on knowledge of the identity of the user-defined shell clusters as well as the cluster-interaction dependencies of a replication protocol.

- **Core:** Replicas included in the core domain are only subject to crash faults and exclusively process inputs obtained from other potentially crash-faulty clusters (i.e., filters and other cores). The SHELLFT framework automatically labels all clusters as cores that are neither shells nor filters.

As illustrated in Figure 2, as a starting point for the tailoring process SHELLFT relies on the implementation of a purely crash-tolerant *base protocol* which our framework then successively transforms into a hybrid protocol implementation taking into account a user’s shell selection. Among other things, this process may include changes to the number of replicas comprised in certain clusters, a reconfiguration of the thresholds based on which individual replicas accept an input value, or the replacement of crash-tolerant protocol mechanisms with more resilient Byzantine fault-tolerant logic. Once the tailoring is complete, the result is a custom and preconfigured hybrid protocol implementation that is ready for deployment.

To account for the specific characteristics of our three fault domains, when deploying a SHELLFT system we physically isolate the shell from the rest of the protocol by running the shell clusters on a separate group of machines. This way, if a Byzantine fault (e.g., caused by an attack) is not limited to a shell replica but compromises the entire machine hosting the replica, the filter and core clusters still remain unaffected. In sum, the use of two server groups enables us to minimize SHELLFT’s resource consumption and deployment costs, while at the same time offering a high degree of resilience.

### B. Base Protocol

The base protocol is a crash-tolerant replication protocol that we specially developed as basis for our tailoring process. To facilitate the tailoring, we designed the base protocol in such a way that its architecture and workflow closely resemble the architecture and workflow of the existing micro-replicated BFT protocol Mirador [22]. Most notably, this allows us to substitute selected base-protocol clusters for their Mirador counterparts as part of the adaptation (see Section III-C).

For the agreement on client requests, the base protocol relies on a consensus algorithm that is comparable to Paxos [1]. In particular, we target systems in which the number of concurrent failures to tolerate is small (e.g.,  $f \leq 2$ ). Such system environments match our goal of further strengthening robustness through diversification, which in practice (despite the benefits offered by SHELLFT) is unlikely to become affordable for deployments comprising tens or even hundreds of servers. For the base protocol, this means that scalability with the number of faults ( $f \gg 1$ ) is not a requirement. Exploring our approach in large systems is a potential direction for future work and presumably involves the development of another base protocol.

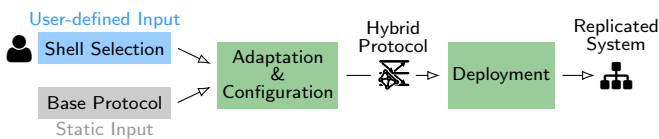


Fig. 2: Overview of the SHELLFT tailoring process.

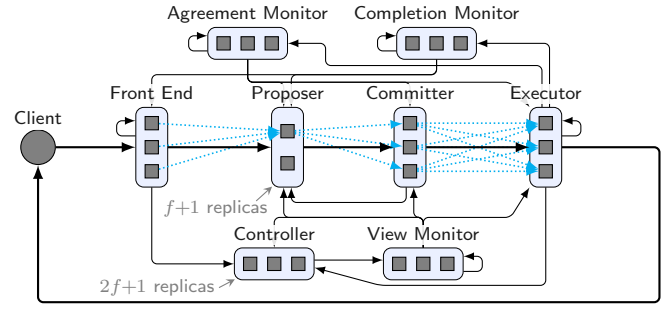


Fig. 3: Base protocol.

Although the base protocol is executable on its own, we did not concentrate on optimizing for such a scenario. Instead, our primary focus was to create a parameterized template protocol that the SHELLFT framework can use to support selective hybridization. With the resulting tailored protocol later potentially being subject to Byzantine faults, this for example means that replicas in the base protocol already communicate via authenticated messages to prevent adversaries from successfully impersonating correct replicas.

The full specification and system model of the base protocol can be found in [34]. In the following, we give a brief overview of the protocol and its parameterization options.

**Protocol Architecture.** As shown in Figure 3, the base protocol consists of eight micro-replica clusters, four of which represent the main request path through the system. Specifically, incoming client requests first arrive at a cluster of front-end replicas. Next, they are forwarded through a chain of proposer, committer, and executor clusters which together implement a Paxos-style consensus process [35] that assigns a unique sequence number to each request. Finally, the executors process committed requests in the order of their sequence numbers and send the corresponding replies back to the client.

In addition to these four main clusters, the base protocol contains three different monitor clusters that are responsible for obtaining and distributing progress information on the consensus process (agreement monitor), executed requests (completion monitors), and view number (view monitors). Acting as control loops, monitors enable replicas of the main path to perform garbage collection of no longer needed state (e.g., agreement slots that are superseded by a stable checkpoint). For clarity, we omit a fine-grained discussion of these mechanisms because (1) control loops in the base protocol follow the same principles as control loops in Mirador and (2) their specifics (like other Mirador details deliberately left out here) are not of relevance for the contributions of this paper, namely selective hybridization and diversification. The same applies to the controller cluster, a group of replicas that monitors agreement and execution progress and if necessary triggers a view change for proposers, committers, and executors.

Relying on this architecture, information flows through the system by replicas of a cluster providing the outcomes of their own protocol step as inputs to the replicas of their successor cluster(s), typically in an all-to-all manner, as illustrated for the main-path clusters in Figure 3 (dotted blue arrows). Deciding

on a protocol step in general requires a replica to analyze the input values provided by different predecessors and accept a value once a predefined threshold is reached; either in terms of quorum size (e.g., an executor commits a request once  $f + 1$  committers have confirmed the request’s reception) or in terms of numerical value (e.g., the current view is determined as the  $f + 1$  highest number announced by view monitors).

**Parameterization.** Leveraging its micro-replicated protocol architecture, the base protocol serves as starting point for the creation of tailored replication protocols. In particular, the base protocol offers adaptations in three main dimensions:

- **Adjustment of Replication Factors:** With clusters being largely independent of each other, the number of involved replicas can be defined on a per-cluster basis. For some protocol steps, this offers the opportunity to achieve robustness against Byzantine faults by adding replicas to the corresponding cluster (see Section IV).
- **Configuration of Acceptance Thresholds:** Raising the bar for the acceptance of input values enables a replica to limit the impact faulty inputs can have on its decisions. Usually, such a measure is complemented by a matching increase in the replication factor of the respective predecessor cluster.
- **Substitution of Protocol Mechanisms:** Exploiting the loose coupling of micro-replica clusters, entire mechanisms of the base protocol can be modularly replaced by their counterparts from other micro-replicated protocols. As shown in Section IV-B, this for example makes it possible to substitute the base protocol’s logic for distributing proposals with an enhanced mechanism that is resilient against equivocation by a faulty proposer.

Unlike the selection of shell clusters, decisions regarding base-protocol adaptations are not made by users, but by the SHELLFT framework as part of the automated tailoring process.

### C. Tailoring Process

The main purpose of the tailoring process is to transform the base protocol into a customized implementation. To automate this process and set up the resulting system, we developed the SHELLFT framework. Next, we present the tailoring in detail.

**Cluster Adaptation.** Starting with the user-specified selection of shell clusters, the SHELLFT framework first decides on which functionality and clusters are required for the chosen shell configuration. For this purpose, the framework relies on a predefined set of rules that for each base-protocol cluster defines the specific actions that need to be performed in order for the cluster to become a shell. The extent of these actions depends on the particular protocol step affected and hence varies between clusters (cf. Section III-D). As summarized in Table II, for most clusters the base-protocol implementation can either be directly used without modification, or substituted in place with its counterparts from the Byzantine fault-tolerant protocol Mirador. In other cases, the changes are more wide-ranging and, for example, affect multiple clusters. Most notably, putting the proposer inside a shell domain leads to the addition of new clusters, as further detailed in Section IV-B.

Shell Selection	Replacements	Size Update
Front end	–	–
Proposer	Mirador agreement stage	–
Committer	Adapted proposer, Mirador executor	$3f + 1$
Executor	Mirador executor and client	$3f + 1$
Agreement monitor	Mirador agreement monitor	$3f + 1$
Completion monitor	Mirador completion monitor	$3f + 1$
View monitor	Mirador view monitor	$3f + 1$
Controller	–	–

TABLE II: Cluster replacements and size updates performed by the SHELLFT framework if a cluster is selected as shell.

**Cluster Configuration.** At this point, the SHELLFT framework knows all relevant clusters and which of them are part of the shell domain. Using this information, in the next step the framework classifies the remaining clusters as either filters or cores based on a dependency graph that models the interaction between clusters (cf. Figure 3). Most importantly, as soon as a cluster processes direct input from shell functionality, it is automatically included in the filter domain. This rule also applies if multiple inputs are combined and only some of them come from the shell. Only clusters that receive inputs *solely* from filters or cores (i.e., replicas that the framework user assumes to fail by crashing) are themselves assigned to the core domain.

Once all clusters are assigned a domain, the SHELLFT framework then proceeds by configuring the individual size of each cluster. With filter and core clusters representing crash-tolerant domains, their replication factors remain the same as in the base protocol. In contrast, to account for potential Byzantine faults, shell clusters are typically expanded by  $f$  additional replicas, resulting in BFT-typical cluster sizes of  $3f + 1$ . To account for changes in replication factors, for those shell clusters whose sizes have increased (in case there are any), the SHELLFT framework in a final configuration step increases the corresponding acceptance thresholds of all affected (shell and filter) successor clusters by  $f$ . This allows replicas of the successor clusters to make effective use of the  $f$  additional inputs that are now available as a result of the cluster expansion, and to thereby actually tolerate Byzantine faults in the shell. For more details on how specific transformation decisions are made, please refer to Section III-D.

**Cluster Deployment.** Although clusters are isolated entities from a protocol perspective, there are several opportunities with regard to their deployment; they range from hosting each replica on a separate server to co-locating replicas of different clusters within the same thread. For SHELLFT we exploit this flexibility to achieve a balance between resilience and efficiency by (1) physically separating shell clusters from the rest of the system and (2) integrating multiple replicas with each other whenever possible. This strategy results in a setting comprising two groups of machines: one group for shell replicas and one group for filter and core replicas. Figure 4 shows an example of such a deployment for a configuration in which the shell consists of front ends and executors (cf. Section IV-A). Keeping the shell clusters isolated from the crash-tolerant domains increases robustness because, even if an adversary manages to compromise an entire machine of



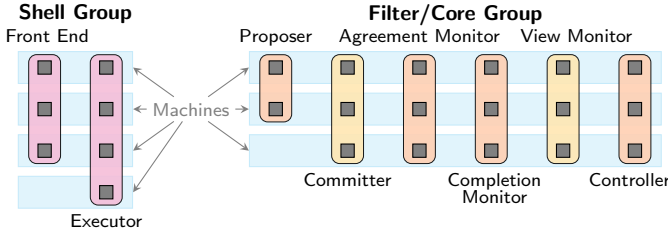


Fig. 4: Group-based deployment on separate sets of machines.

the shell group, the adversary still does not have control over filter and core replicas. At the same time, without impairing availability, the co-location of replicas from different clusters within each of the two groups minimizes the communication overhead between the corresponding protocol steps and therefore improves both resource consumption and efficiency.

#### D. Pattern-based Protocol Transformation

During its adaptation and configuration steps, the SHELLFT tailoring process leverages the fact that micro-replicated protocols are designed as a composition of established architectural patterns with specific safety and liveness guarantees [22]. This way, the transformation essentially becomes the task of translating a pattern that provides a certain property in the presence of crash faults into its counterpart pattern providing the same property in the presence of Byzantine faults.

In this context, it is important to note that SHELLFT's tailoring process does not change how individual patterns are interweaved to form the overall protocol composition, and thus ensures that the associated inter-pattern correctness arguments remain unaffected. Instead, the transformation performed by SHELLFT occurs at the pattern level, thereby making it significantly easier to maintain correctness due to only a comparably small number of precisely defined properties having to be preserved across the transformation. Next, we illustrate this approach for the base protocol's two main patterns (see Figure 5).

**Reliable Distribution Pattern.** This pattern propagates a value from a potentially faulty *source* replica (possibly via intermediate replicas called *witnesses*) to a group of *sink* replicas while ensuring that correct sinks do not accept diverging values. Similar to traditional protocols [3], [35], such a task, for example, marks the first step of the base protocol's agreement logic. For correctness, two properties are required from this pattern and, as proven in [34], both the base version and the transformed version of the pattern provide them:

**Property RDP.1.** *If a correct sink  $s_1$  accepts a value  $v$  and another correct sink  $s_2$  accepts a value  $v'$ , then  $v = v'$ .*

**Property RDP.2.** *If the source is correct and proposes  $v$ , all correct sinks eventually accept  $v$ , even if  $f$  witnesses are faulty.*

As shown at the top of Figure 5a, the (crash-fault) base version of this pattern involves the source directly broadcasting the proposed value to all sinks. In contrast, if the source cluster is selected as shell, SHELLFT's tailoring process switches to the Byzantine-resilient counterpart pattern (see bottom of Figure 5a). Here, an additional cluster of  $3f + 1$  witness

replicas observe the value proposed by the source, and sinks only accept a value after having received matching opinions from  $2f + 1$  witnesses. That is, in this case the transformation includes (1) the insertion of a new cluster as well as (2) the increase of the sinks' acceptance threshold from 1 to  $2f + 1$ .

**Relay Pattern.** This pattern is used to distribute a value (e.g., the sequence number of a reached checkpoint) from multiple sources to multiple sinks; up to  $f$  sources may be faulty. A correct source either puts out a specific value  $v$  or no value at all. Under these conditions, the relay pattern ensures that if a correct sink accepts  $v$ , then eventually all correct sinks accept the same value. In the base protocol, such a guarantee represents the foundation of control loops (cf. Section III-B). As proven in [34], SHELLFT's tailoring process maintains two key properties across the protocol transformation:

**Property RP.1.** *If a correct sink accepts a value  $v$ , then  $v$  was proposed by a correct source.*

**Property RP.2.** *If a correct sink accepts a value  $v$ , then all correct sinks eventually accept  $v$ , even if  $f$  relays are faulty.*

In the base pattern (Figure 5b),  $2f + 1$  sources send their value to a cluster of  $2f + 1$  relays. In addition, relays propagate an accepted value among each other. A correct relay accepts a value based on either  $t_s = f + 1$  matching inputs from different sources or one accepted value forwarded by another relay. A sink accepts a value after obtaining  $f + 1$  matching inputs from different relays. If the sources are in the shell, our transformation (1) changes the the source-cluster size from  $2f + 1$  to  $3f + 1$  replicas and (2) increases  $t_s$  to  $2f + 1$  inputs.

#### E. Selective Diversification

For additional resilience, SHELLFT enables selective hybridization to be complemented with *selective diversification*. In general, the diversification of replicas (i.e., the use of heterogeneous implementations or deployments) significantly lowers the probability of common-mode failures and makes it more difficult for an adversary to take over multiple replicas at once [36]–[38]. SHELLFT improves this process in two complementary ways: It reduces the diversification costs for critical parts and, at the same time, increases the diversification space.

**Reducing the Diversification Costs.** While monolithic architectures suffer from high diversification costs, SHELLFT allows diversification techniques to be applied to individual clusters, and hence with low overhead. Having been identified

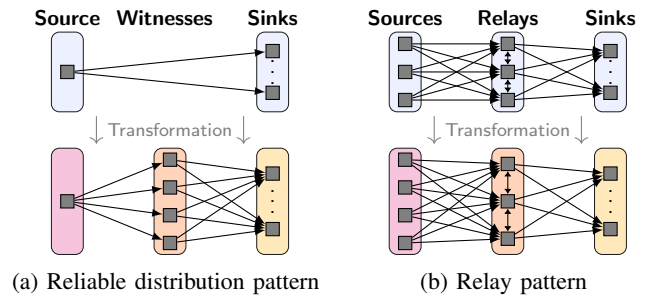


Fig. 5: Transformation based on design patterns.

	# Successful Exploits					
	1	2	3	4	5	9
Monolithic protocol	0%			100%		
Group-based deployment	0%	42.9%		100%		
Fully diversified base protocol	0%	12.5%	25%	37.5%	50%	100%

TABLE III: Probability of system-wide failure ( $f = 1$ ).

as the most critical/vulnerable parts of the overall system, the shell clusters are primary candidates for methods such as N-version programming. However, if considered beneficial, additional filter and core clusters may be included to further improve resilience. Besides, our approach complements currently developed LLM-based automation techniques for N-version programming [39], which require isolated (“pure”) functions.

**Increasing the Diversification Space.** As diversification space we define the number of possible heterogeneous configurations deployed within a system. A larger diversification space allows to use a more diverse system layout and hence increases the system’s resilience as a whole by minimizing the number of individual parts that are affected by a single common-mode failure (e.g., an exploit or bug in the runtime environment). In traditional monolithic implementations, each replica runs on exactly one physical node using one configuration. This results in the diversification space being directly related to the replication factor dependent on  $f$ . In contrast, in SHELLFT a replica is a logical construct that is more independent of the physical layout, which offers the opportunity to increase the diversification space without having to adjust the replication factor.

To illustrate this aspect, Table III shows how many successful exploits an attacker requires to disrupt a diversified system. Using a heterogeneous monolithic protocol, each exploit takes down one replica, and with the second exploit (for  $f = 1$ ), the system is no longer operable. In contrast, SHELLFT’s group-based deployment presented in Figure 4 already increases overall system resilience. With a SHELLFT-based system, an attacker first has to find exploits for  $f + 1$  micro replicas of the same type. If the attacker has no intricate knowledge of the system deployment that would allow the attacker to target parts of the system specifically, this decreases the percentage of a system-wide failure to less than 50% for two exploits.

Note that if resilience is the main goal, this can be extended to a point where all micro replicas are run in their own deployment configuration and diversified individually. Such *full diversification* significantly decreases the likelihood of a system-wide failure, with only 50% for even five exploits.

#### IV. SHELLFT PROTOCOL EXAMPLES

SHELLFT enables replicated systems that are tailored to the individual characteristics and requirements of use cases. To illustrate how different shell selections influence the resulting protocol, in this section we present three SHELLFT protocols produced by our framework. MINAS is based on the concept of perimeter security and splits the protocol into outside and inside clusters. In SENTRY, the shell consists of the protocol steps that are most critical for the overall system. Our third protocol is a composition of MINAS and SENTRY showing that different shell selections can be combined with each other.

##### A. MINAS: Perimeter Security

MINAS applies the principle of perimeter security [40], [41]. Of our three protocols, it best represents the visual image of a separation between an outer shell and an inner core.

**Use-Case Scenario.** MINAS targets data-center environments in which all replicas are connected via a private network that is isolated from public traffic. Such a scenario is not only common for local replicated services but can also be found in hierarchical geo-replicated systems [20], [42], [43]. Combined with perimeter security inside a data center (e.g., multiple firewalls, network segmentation [44], SDNs [45], intrusion detection [46], [47]), this means that only client-facing clusters must reside in the outer perimeter; all others can be placed in an inner perimeter and are thus less exposed to malicious attacks.

**Selection of Shell Functionality.** Given these properties, we instruct the SHELLFT framework to put the two client-facing clusters (i.e., front ends and executors) inside the shell.

**Protocol Architecture.** Figure 6 shows the resulting MINAS system architecture after the tailoring process. As indicated in Table II, assigning the executor cluster to the shell results in replacements of the executor and client logic, whereas the selection of the front-end cluster does not require any changes. Overall, with only two clusters in the shell, most parts of the system remain in the filter and core domains, thereby keeping replication costs close to those of the original crash-tolerant base protocol while significantly increasing resilience.

**Additional Details.** As shown by previous works [14], [19], it is possible to design BFT execution stages requiring only  $2f + 1$  replicas. Applying this optimization to the executor cluster is outside the scope of this paper, but would further minimize MINAS’s resource consumption. In particular, with regard to the deployment of the system (see Section III-C, “Cluster Deployment”), it would allow MINAS to reduce the size of its shell group from  $3f + 1$  to  $2f + 1$  machines.

##### B. SENTRY: Safety First

For the second protocol SENTRY, we take a different approach. In SENTRY, security considerations do not include the locations of individual system parts, instead shell replicas are selected based on the negative impact that a cluster can have on the overall system when exhibiting Byzantine behavior.

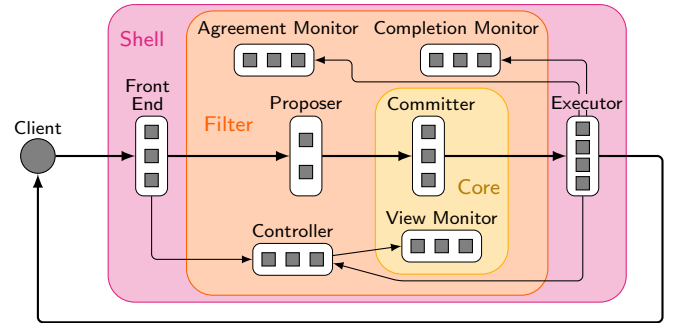


Fig. 6: Overview of the MINAS system architecture; some cluster dependencies have been omitted for better readability.

**Use-Case Scenario.** Several recent works argue that, given the choice, for many replicated systems in practice it is much more important to preserve safety than liveness [11], [15]. The rationale behind this consideration is the insight that from a client perspective, for example, it is usually better to receive no response at all than to receive an erroneous response that reflects inconsistent system state. In addition, it is generally easier to automatically detect liveness issues (e.g., by using external monitoring tools) than safety violations. Following this idea, for SENTRY we specifically apply selective hybridization to those parts of a replicated system that are critical for safety.

**Selection of Shell Functionality.** In a crash-tolerant protocol, there are typically four tasks that pose a particular threat to the safety if they are subject to Byzantine faults: (1) Faced with an agreement protocol that consists of only two phases, by performing equivocation (i.e., proposing different client requests for the same sequence number to different follower replicas) a Byzantine leader replica can trick correct followers into executing requests in diverging orders. This in turn may cause replica states to become inconsistent. (2) In a similar way, a Byzantine leader is able to manipulate the outcome of a view change by distributing different opinions on the set of requests that need to be re-proposed in the new view. (3) Having executed a request, a Byzantine replica may provide the client with an incorrect result. (4) If a state transfer becomes necessary (e.g., due to a replica rejoining the rest of the system after the end of a temporary network partition), a Byzantine replica can supply a correct replica with an incorrect checkpoint and thereby tamper with the correct replica’s state.

In our base protocol, these four tasks are the responsibilities of two clusters, proposer and executor, which is why for SENTRY we select both of them as shell. By design, faulty replicas of all other clusters can only impede the liveness of the overall protocol, even if they show Byzantine behavior.

**Protocol Architecture.** Based on this configuration, the SHELLFT framework produces the system architecture shown in Figure 7. As the most significant change, in consequence of the proposer being part of the shell, SHELLFT’s tailoring process applies the replacements associated with the reliable-distribution pattern (see Section III-D) by substituting the proposer and committer clusters with the agreement stage of Mirador, thereby introducing an additional preparer cluster on the main request path. This enables SENTRY to deal with equivocation attempts potentially made by a Byzantine proposer.

Furthermore, off the main path, the agreement stage now includes Mirador’s clusters for performing a view change, which in a nutshell works as follows. Once a view change is triggered, a specific cluster (“conservators”) collects the preparers’ and committers’ opinions on the requests to be re-proposed in the new view. Led by a dedicated replica (“curator”), this information is then agreed on using a three-step consensus process, which is similar to the three-phase agreement on the main path. To complete the view change, at the end the agreed outcome is fed back to the proposer and preparer clusters, enabling them to continue with normal-case operation.

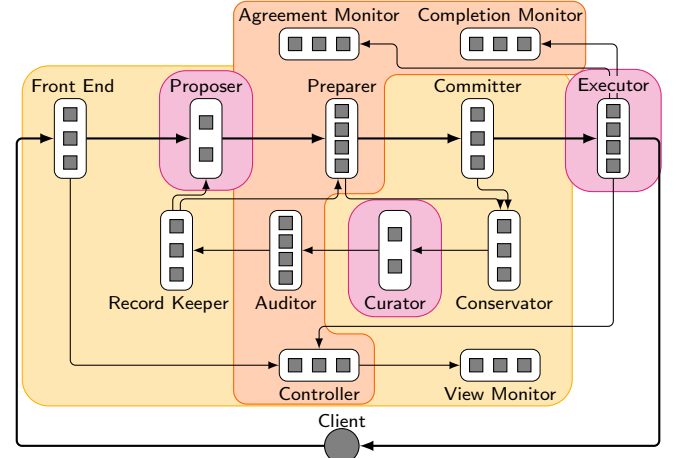


Fig. 7: Overview of the SENTRY system architecture; some cluster dependencies have been omitted for better readability.

To summarize, in SENTRY’s architecture the base protocol’s proposer functionality is divided among two clusters, with the SENTRY proposer handling the main request agreement and the curator leading the view change. As shown in Figure 7, this results in the curator also constituting a part of the shell. For the third shell cluster, the executor, the SHELLFT’s tailoring entails the same modifications and benefits as in MINAS.

### C. MINAS+SENTRY

For our third protocol, we combine MINAS and SENTRY to MINAS+SENTRY in order to illustrate that there is no need to focus on a single criterion when selecting the shell functionality. In recent years, the concepts of zero trust [48] or the rogue administrator [49] have received much attention in the area of cloud computing. So even if a deployment environment fits the characteristics of MINAS, for certain applications additional safety mechanisms can still be beneficial. The resulting system architecture of MINAS+SENTRY is very similar to SENTRY’s architecture in Figure 7, except that the front-end cluster in MINAS+SENTRY is also part of the shell. With this design, MINAS+SENTRY is able to target two threat vectors: The shell now contains both the most *exposed* functionality from MINAS as well as the most *critical* functionality as defined in SENTRY.

### V. DIVERSIFICATION-COST ANALYSIS

Of the two main goals of our work, selective hybridization and selective diversification, the former is inherently provided by SHELLFT’s design (as discussed in Section III). For this reason, in this section we focus on assessing the diversification costs of SHELLFT-based replicated systems. SHELLFT’s modular hybrid architecture makes it possible to develop heterogeneous system implementations in which only the most exposed and/or critical components are actually diversified. In the following, we seek to examine the ramifications of this approach with regard to complexity and programming effort. Since both aspects are known to be difficult to quantify when it comes to actual implementations, we rely on two different methodologies for this purpose. Our first analysis, on a more



Functionality Component	Cluster Counterpart	Protocol					
		Baseline	Hybrid	Mirador	MINAS	SENTRY	MINAS+SENTRY
Accept requests from clients	Front end	$2f + 1$	$2f + 1$	$2f + 1$	$2f + 1$	$2f + 1$	$2f + 1$
Assign sequence numbers to requests	Proposer	$2f + 1$	$2f + 1$	$f + 1$	$f + 1$	$f + 1$	$f + 1$
Verify and relay proposals	Preparer	0	0	$3f + 1$	0	$3f + 1$	$3f + 1$
Confirm sequence-number assignment	Committer	$2f + 1$	$2f + 1$	$3f + 1$	$2f + 1$	$2f + 1$	$2f + 1$
Execute requests (and send/store reply)	Executor	$2f + 1$	$2f + 1$	$3f + 1$	$3f + 1$	$3f + 1$	$3f + 1$
Monitor progress (to trigger view change)	Controller	$2f + 1$	$2f + 1$	$2f + 1$	$2f + 1$	$2f + 1$	$2f + 1$
Monitor and broadcast current view	View monitor	$2f + 1$	$2f + 1$	$3f + 1$	$2f + 1$	$2f + 1$	$2f + 1$
Gather prepared requests for next view	Conservator	0	0	$3f + 1$	0	$2f + 1$	$2f + 1$
Decide on unique set of requests (across views)	Curator	0	0	$f + 1$	0	$f + 1$	$f + 1$
Verify set of requests to re-propose	Auditor	0	0	$3f + 1$	0	$3f + 1$	$3f + 1$
Store and provide set to re-propose	Record keeper	0	0	$3f + 1$	0	$2f + 1$	$2f + 1$
Determine active consensus instances	Agreement monitor	$2f + 1$	$2f + 1$	$3f + 1$	$2f + 1$	$2f + 1$	$2f + 1$
Determine executed requests	Completion monitor	$2f + 1$	$2f + 1$	$3f + 1$	$2f + 1$	$2f + 1$	$2f + 1$
Total		$16f + 8$	$16f + 8$	$33f + 13$	$16f + 8$	$27f + 13$	$27f + 13$
Byzantine fault model		0	$16f + 8$	$33f + 13$	$5f + 2$	$5f + 3$	$7f + 4$
Percentage of functionality to diversify (compared with baseline): $f = 1$		0%	100%	192%	29%	33%	46%
$f \rightarrow \infty$		0%	100%	206%	31%	31%	44%

TABLE IV: Complexity comparison. Components using a Byzantine fault model are highlighted in dark. The diversification percentage compares the number of diversified components to the number of baseline components (e.g.,  $\frac{5f+2}{16f+8}$  for MINAS).

theoretical level, concentrates on the individual protocol-task functionality that needs to be diversified to improve resilience. As a complement, our second analysis studies the code size of heterogeneous SHELLFT-cluster implementations we developed by applying N-version programming.

#### A. Diversification of Protocol Functionality

Our first analysis is based on the notion of a replication protocol being a composition of multiple tasks. In SHELLFT protocols, each of these tasks is represented by a dedicated cluster, which allows us to use their number and sizes as a metric for complexity. Specifically, we estimate and compare the diversification costs of different approaches by determining how often the logic of each cluster needs to be diversified. Notice that this methodology involves simplifying assumptions: (1) Although some tasks require more sophistication than others, it treats all tasks as similarly complex. We argue that this does not pose a major problem because, as a consequence of these tasks essentially representing the atoms of a replication protocol, the overall differences are not extensive. (2) It does not consider general functionality such as communication, which in practice also needs to be diversified. These parts are included in our second analysis in Section V-B which examines the actual code bases of our prototype implementations and comes to very similar conclusions, thereby confirming that the assumptions made for our first study are justified.

**Baseline.** We derive the baseline for our first study from the base protocol (see Section III-B) due to its consensus mechanism being closely related to Paxos in Kirsch and Amir’s system-builders variant [35], a protocol that represents the design of a typical crash-tolerant protocol in practice and applies a style of consensus that resembles the agreement process of the other analyzed protocols. Using our metric, a common crash-tolerant system consisting of  $2f + 1$  replicas needs to perform all tasks handled by the 8 server-side base-protocol clusters, with every one of these tasks being implemented in each replica. In sum, as shown in Table IV, this results in a baseline complexity equivalent to  $16f + 8$  micro replicas.

**Traditional Hybrids.** With traditional hybrid systems such as MinBFT, XFT, and VFT (see Section II) consisting of monolithic replicas, diversifying them requires heterogeneous implementations of the entire replica logic (i.e., all base-protocol tasks), and hence leads to diversification costs of at least 100% (Table IV, “Hybrid” column). Note that this number is a conservative estimate as it does not include the added logic (and thus complexity) these approaches introduce.

**Full-Fledged BFT Replication.** Increasing the resilience of Byzantine fault-tolerant components by introducing heterogeneity in a Mirador-based system involves the diversification of all of its  $33f + 13$  micro replicas. For  $f = 1$ , for example, the added complexity of full-fledged BFT replication leads to overall diversification costs of  $\frac{33f+13}{16f+8} = \frac{46}{24} = 192\%$ .

**SHELLFT Protocols.** SHELLFT’s selective diversification offers the benefit of limiting diversification costs to the most exposed/critical parts: the shell clusters. In MINAS, the shell consists of only two clusters (i.e., front ends and executors) with a total of  $5f + 2$  micro replicas. For a system tolerating a single fault in each cluster ( $f = 1$ ), this reduces the diversification efforts to only  $\frac{5f+2}{16f+8} = \frac{7}{24} = 29\%$  of the costs associated with traditional hybrid approaches. Relying on larger shells, the costs for SENTRY ( $\frac{5f+3}{16f+8} = \frac{8}{24} = 33\%$ ) and MINAS+SENTRY ( $\frac{7f+4}{16f+8} = \frac{11}{24} = 46\%$ ) are slightly higher, nevertheless both numbers still represent a significant improvement over traditional hybrid systems (100%).

**Impact of System Size.** To examine the relationship between diversification costs and system size we not only compute values for  $f = 1$ , but also for  $f \rightarrow \infty$ . As shown in the last line of Table IV, for the three SHELLFT protocols the resulting numbers do not differ significantly from the ones for  $f = 1$ , which means that the savings enabled by SHELLFT are largely independent of system size. In particular, this confirms that our target systems, in which (as discussed in Section III-B)  $f$  is small, are already able to benefit from the approach.

	Scope	Full Code	Replica Logic
Java	All clusters	10,101	7,037
Java	Base protocol	8,900	5,836
C++	Executor + front end	2,756	1,643
Elixir	Executor + front end	2,126	1,469
Go	Executor, proposer, curator	3,081	2,179

TABLE V: Diverse SHELLFT implementations (lines of code).

### B. N-Version Programming

In our second study, we focus on N-version programming as a means to achieve diversification. To evaluate the impact of SHELLFT on implementation costs in this context, we applied the concept to all shell clusters of MINAS and SENTRY. In an effort to cover a wide range of heterogeneity, we chose a variety of programming languages, using different runtime environments and even paradigms: Java (the language in which the SHELLFT framework itself is written), C++ (standard C++20), Go (v1.22), as well as Elixir (v1.12 with Erlang/OTP 24), a functional programming language designed for distributed and fault-tolerant systems [50].

Table V reports the sizes of these code bases measured in lines of code (LOC) as calculated by `clloc` (v1.90) [51]. We are aware that LOC can be a somewhat imprecise unit of measurement, especially when comparing different programming languages, but it nevertheless allows us to get a good impression on the potential costs of development and maintenance. The full-scope implementation in Java comprises the functionality for the entire replicated system and consists of around 10k LOC, of which about 7k LOC are dedicated to the protocol logic of replicas. The difference of about 3k LOC includes general system functionality (e.g., communication, startup procedures) that is required for each replica regardless of its cluster. When considering only the eight clusters that are part of the base protocol, this leaves around 8.9k LOC for the whole implementation and 5.8k LOC for the replicas. In comparison, our N-version programming implementations in C++, Go and Elixir either contain two or three clusters and have a greatly reduced code size. Specifically, they range between 2.1k LOC and 3.1k LOC, which is 21–31% and 23–35% of the full-scope Java implementation and base protocol, respectively. This difference is maintained for the replica logic and corresponds to the fact that either only two (25%) or three (37.5%) out of eight clusters are diversified in another programming language.

### C. Discussion

Although applying different methodologies, both of our analyses arrive at consistent conclusions with respect to the reduction of diversification costs enabled by SHELLFT. For MINAS, for example, the first analysis determined the need to diversify 29% of the functionality to improve shell resilience. As shown by the second analysis, this closely matches the actual code sizes of our heterogeneous MINAS shell-cluster implementations in C++ and Elixir. On average, they only comprise 27% of the size of the base-protocol code base (both in terms of the full code as well as the replica logic), and hence confirm the significant savings (more than 70%) made possible by SHELLFT compared with traditional hybridization approaches.

## VI. PERFORMANCE AND FAULT-HANDLING EVALUATION

In this section, we present the results of experiments that compare MINAS, SENTRY, and MINAS+SENTRY against state-of-the-art protocols, evaluating both performance and the impact of various replica failures. Notice that the goal of the performance experiments is to study our approach’s impact on throughput and latency. We do not promote SHELLFT as a technique for improving performance.

### A. Performance

We evaluate the performance of our SHELLFT protocols with multiple baselines: the base protocol, the micro-replicated BFT protocol Mirador, and the widely used BFT-SMaRt [52] library, both in a Byzantine and a crash fault-tolerant configuration (BFT-SMaRt *BFT* and BFT-SMaRt *CFT*). Since SHELLFT primarily focuses on small systems (see Section III-B), we dimension all systems for  $f=1$ , meaning that MINAS, SENTRY and MINAS+SENTRY replicas are hosted on 7 machines (cf. Figure 4). As application, we employ a key-value store and run YCSB [53] with an update-heavy workload and 1 KB records containing fields of 100 B values. Each reported data point represents the average of three runs.

**Throughput and Latency.** As shown in Figure 8, with a maximum throughput of  $\sim 42$ k requests/s and a latency of  $\sim 1$  ms, the three SHELLFT protocols are in between the base protocol/BFT-SMaRt *CFT* and Mirador/BFT-SMaRt *BFT*, indicating that they do not only represent an intermediate between crash tolerance and Byzantine fault tolerance in terms of complexity, but also with regard to performance.

**Performance of Diversified Systems.** Comparing the pure Java implementations of MINAS and SENTRY with their heterogeneous counterparts comprising the diversified shell components listed in Table V, we observe similar throughput and latency results. This shows that a diversification of crucial parts of a replicated system as enabled by SHELLFT can increase resilience without noticeably impacting performance.

### B. Impact of Replica Failures

The final part of our evaluation investigates the impact of replica failures, which is why we subject the systems to both replica crashes as well as Byzantine behavior (see Figure 9). The view-change timeout in these experiments is set to 1 s.

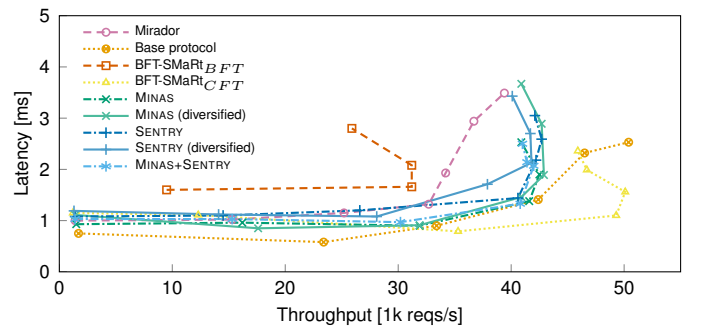


Fig. 8: Performance comparison.

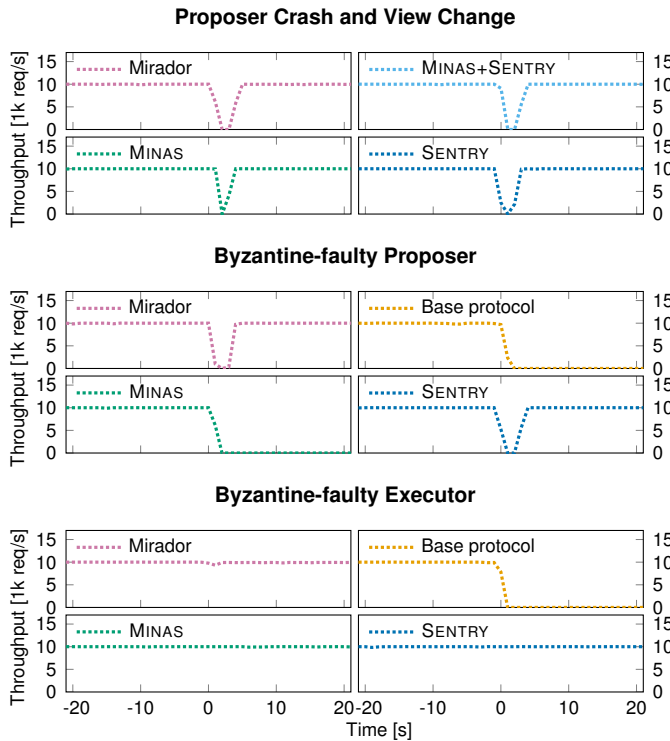


Fig. 9: Impact of different fault scenarios.

**Leader Crash.** As the consequence of a crash of the current proposer, all three SHELLFT protocols experience the expected downtime of 1–2 s and resume a stable performance after the successful view change. The view change in MINAS is completed slightly faster due to comprising fewer phases.

**Byzantine Failures.** To study the impact of arbitrary faults on two key clusters (i.e., proposer and executor), we employ the full extent of their potential Byzantine behavior, ranging from equivocation to forging responses and checkpoints. Again, all evaluated protocols demonstrate the behavior expected from their respective fault model. While (Byzantine) equivocation of the proposer in MINAS and the base protocol leads to diverging replicas, Mirador and SENTRY (as well as MINAS+SENTRY) tolerate the fault by issuing a view change. In contrast, with executors being part of their shell, all SHELLFT protocols tolerate a Byzantine-faulty executor. Unlike the base protocol, none of the SHELLFT protocols experiences any notable impact.

## VII. RELATED WORK

Having already discussed a large body of related work in the area of hybridization in Section II, in the following we focus on additional aspects with relevance to SHELLFT.

**Modularized Replication Architectures.** Micro replication so far has only been investigated in the context of improving debuggability [22]. With SHELLFT, we are the first to harness its properties for the design of flexible hybrid architectures.

Whittaker et al. [21] presented a systematic approach to eliminate performance bottlenecks by compartmentalizing the affected replication-protocol parts into multiple components.

Concentrating on efficiency and scalability, this method is orthogonal to our goal of combining different fault models within the same system. However, being a general technique, compartmentalization could be used to improve performance in SHELLFT protocols. The same is true for mechanisms improving the communication flow between replicas [54], [55].

**N-Versioning of Microservices.** Espinoza et al. [56] showed that heterogeneous implementations are an effective means to increase robustness in microservice architectures, especially when applied to components handling user data (e.g., input sanitizers, which in SHELLFT can be implemented in the front ends). Unlike SHELLFT, their approach does not consider replication protocols and introduces non-replicated (and hence non-fault-tolerant) proxies for each diversified microservice.

**Physical Separation of Replicas.** Placing replicas at significant geographic distances from each other, geo-replicated systems such as GeoPaxos [57], GeoPaxos+ [58], or ATLAS [59] make it highly unlikely that a single root cause leads to the failure of multiple replicas. Like these systems, a SHELLFT deployment relies on a physical separation between replicas providing the same functionality (i.e., SHELLFT replicas belonging to the same cluster), but in addition it also isolates shell replicas from filters/cores. Studying the benefits and implications of implementing our two-level separation in geo-replicated settings is an interesting direction for future work.

**Adaptation to Threats.** Targeting scenarios in which the strength of an adversary evolves over time, Silva et al. [60] developed a BFT system that is able to dynamically adapt both the number of replicas and its resilience threshold at runtime. Integrating this concept with SHELLFT protocols (at the cluster level) would further decrease their resource footprint.

**Hardening.** Correia et al. [61] proposed to extend crash-tolerant systems with integrity checks to detect arbitrary state corruption and avoid error propagation through erroneous messages. Behrens et al. [62] showed that for Paxos these kinds of checks can be limited to a small part of the protocol logic. Compared with the diversification of replicas, such hardening techniques incur less development costs but, on the other hand, are only able to cope with a subset of Byzantine faults.

## VIII. CONCLUSION

SHELLFT’s selective hybridization offers an unprecedented degree of flexibility when it comes to tailoring the resilience of replicated systems to specific use cases. At the same time, SHELLFT is able to decrease diversification costs by more than 70% compared with traditional hybridization approaches.

## REFERENCES

- [1] L. Lamport, “The part-time parliament,” *ACM Trans. on Computer Systems*, vol. 16, no. 2, 1998.
- [2] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *USENIX ATC*, 2014.
- [3] M. Castro and B. Liskov, “Practical Byzantine fault tolerance,” in *OSDI*, 1999.
- [4] T. Distler, “Byzantine fault-tolerant state-machine replication from a systems perspective,” *ACM Computing Surveys*, vol. 54, no. 1, 2021.

- [5] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Veríssimo, "Efficient Byzantine fault tolerance," *IEEE Trans. on Computers*, vol. 62, no. 1, 2011.
- [6] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "CheapBFT: Resource-efficient Byzantine fault tolerance," in *EuroSys*, 2012.
- [7] T. Distler, C. Cachin, and R. Kapitza, "Resource-efficient Byzantine fault tolerance," *IEEE Trans. on Computers*, vol. 65, no. 9, 2016.
- [8] S. Liu, P. Viotti, C. Cachin, V. Quema, and M. Vukolić, "XFT: Practical fault tolerance beyond crashes," in *OSDI*, 2016.
- [9] J. Behl, T. Distler, and R. Kapitza, "Hybrids on steroids: SGX-based high performance BFT," in *EuroSys*, 2017.
- [10] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, "DAMYSUS: Streamlined BFT consensus leveraging trusted components," in *EuroSys*, 2022.
- [11] I. Messadi, M. H. Becker, K. Blecke, L. Jehl, S. B. Mokhtar, and R. Kapitza, "SplitBFT: Improving Byzantine fault tolerance safety using trusted compartments," in *Middleware*, 2022.
- [12] M. Correia, N. F. Neves, L. C. Lung, and P. Veríssimo, "Low complexity Byzantine-resilient consensus," *Distributed Computing*, vol. 17, no. 3, 2005.
- [13] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: Making adversaries stick to their word," in *SOSP*, 2007.
- [14] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, "UpRight cluster services," in *SOSP*, 2009.
- [15] D. Porto, J. Leitão, C. Li, A. Clement, A. Kate, F. Junqueira, and R. Rodrigues, "Visigoth fault tolerance," in *EuroSys*, 2015.
- [16] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *FTCS-8*, 1978.
- [17] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine fault tolerant systems tolerate Byzantine faults," in *NSDI*, 2009.
- [18] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computer Survey*, vol. 22, no. 4, 1990.
- [19] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for Byzantine fault tolerant services," in *SOSP*, 2003.
- [20] M. Eischer and T. Distler, "Resilient cloud-based replication with low latency," in *Middleware*, 2020.
- [21] M. Whittaker, A. Ailijiang, A. Charapko, M. Demirbas, N. Giridharan, J. M. Hellerstein, H. Howard, I. Stoica, and A. Szekeres, "Scaling replicated state machines with compartmentalization," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, 2021.
- [22] T. Distler, M. Eischer, and L. Lawnczak, "Micro replication," in *DSN*, 2023.
- [23] P. Kuznetsov and R. Rodrigues, "BFTW<sup>3</sup>: Why? When? Where? Workshop on the theory and practice of Byzantine fault tolerance," *SIGACT News*, vol. 40, no. 4, 2009.
- [24] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shriram, "HQ replication: A hybrid quorum protocol for Byzantine fault tolerance," in *OSDI*, 2006.
- [25] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *DISC*, 2017.
- [26] M. Khan and A. Babay, "Making intrusion tolerance accessible: A cloud-based hybrid management approach to deploying resilient systems," in *SRDS*, 2023.
- [27] M. K. Aguilera and S. Toueg, "Failure detection and randomization: A hybrid approach to solve consensus," *SIAM Journal on Computing*, vol. 28, no. 3, 1998.
- [28] H. P. Reiser and R. Kapitza, "Hypervisor-based efficient proactive recovery," in *SRDS*, 2007.
- [29] T. Distler, R. Kapitza, I. Popov, H. P. Reiser, and W. Schröder-Preikschat, "SPARE: Replicas on hold," in *NDSS*, 2011.
- [30] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "TrInc: Small trusted hardware for large distributed systems," in *NSDI*, 2009.
- [31] A. Bessani, M. Correia, T. Distler, R. Kapitza, P. Esteves-Veríssimo, and J. Yu, "Vivisectioning the dissection: On the role of trusted components in BFT protocols," *CoRR*, vol. abs/2312.05714, 2023.
- [32] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Veríssimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 4, 2010.
- [33] P. Thambidurai and Y.-K. Park, "Interactive consistency with multiple failure modes," in *SRDS*, 1988.
- [34] L. Lawnczak and T. Distler, "Hard shell, reliable core: Improving resilience in replicated systems with selective hybridization (extended version)," *CoRR*, vol. abs/2508.10141, 2025.
- [35] J. Kirsch and Y. Amir, "Paxos for system builders: An overview," in *LADIS*, 2008.
- [36] I. Gashi, P. Popov, V. Stankovic, and L. Strigini, "On designing dependable services with diverse off-the-shelf SQL servers," in *Architecting Dependable Systems II*, 2004.
- [37] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Software—Practice & Experience*, vol. 44, no. 6, 2014.
- [38] M. Garcia, A. Bessani, and N. Neves, "Lazarus: Automatic management of diversity in BFT systems," in *Middleware*, 2019.
- [39] J. Ron, D. Gaspar, J. Cabrera-Arteaga, B. Baudry, and M. Monperrus, "Galápagos: Automated N-version programming with LLMs," *arXiv preprint arXiv:2408.09536*, 2024.
- [40] F. M. Avolio, M. J. Ranum, and M. Glenwood, "A network perimeter with secure external access," in *NDSS*, 1994.
- [41] A. Moubayed, A. Refaey, and A. Shami, "Software-defined perimeter (SDP): State of the art secure solution for modern networks," *IEEE Network*, vol. 33, no. 5, 2019.
- [42] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage, "Steward: Scaling Byzantine fault-tolerant replication to wide area networks," *IEEE Trans. on Dependable and Secure Computing*, vol. 7, no. 1, 2010.
- [43] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. ul Haq, M. I. ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran *et al.*, "Windows Azure Storage: A highly available cloud storage service with strong consistency," in *SOSP*, 2011.
- [44] N. Mhaskar, M. Alabbad, and R. Khedri, "A formal approach to network segmentation," *Computers & Security*, vol. 103, 2021.
- [45] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): A survey," *Security and Communication Networks*, vol. 9, no. 18, 2016.
- [46] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE Network*, vol. 8, no. 3, 1994.
- [47] F. Falcão, T. Zoppi, C. B. V. Silva, A. Santos, B. Fonseca, A. Ceccarelli, and A. Bondavalli, "Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection," in *SAC*, 2019.
- [48] S. Mehraj and M. T. Banday, "Establishing a zero trust strategy in cloud computing environment," in *ICCCI*, 2020.
- [49] W. R. Claycomb and A. Nicoll, "Insider threats to cloud computing: Directions for new research challenges," in *COMPSAC*, 2012.
- [50] Elixir, <https://elixir-lang.org/>, last accessed November 2024.
- [51] A. Danial, "cloc – count lines of code," <https://github.com/AIDanial/cloc>, last accessed November 2024.
- [52] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State machine replication for the masses with BFT-SMaRt," in *DSN*, 2014.
- [53] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, 2010.
- [54] A. Charapko, A. Ailijiang, and M. Demirbas, "PigPaxos: Devouring the communication bottlenecks in distributed consensus," in *SIGMOD*, 2021.
- [55] E. Batista, P. Coelho, E. Alchieri, F. Dotti, and F. Pedone, "FlexCast: Genuine overlay-based atomic multicast," in *Middleware*, 2023.
- [56] A. M. Espinoza, R. Wood, S. Forrest, and M. Tiwari, "Back to the future: N-Versioning of microservices," in *DSN*, 2022.
- [57] P. Coelho and F. Pedone, "Geographic state machine replication," in *SRDS*, 2018.
- [58] P. Coelho and F. Pedone, "GeoPaxos+: Practical geographical state machine replication," in *SRDS*, 2021.
- [59] V. Enes, C. Baquero, T. F. Rezende, A. Gotsman, M. Perrin, and P. Sutra, "State-machine replication for planet-scale systems," in *EuroSys*, 2020.
- [60] D. S. Silva, R. Graczyk, J. Decouchant, M. Völz, and P. Esteves-Veríssimo, "Threat adaptive Byzantine fault tolerant state-machine replication," in *SRDS*, 2021.
- [61] M. Correia, D. G. Ferro, F. P. Junqueira, and M. Serafini, "Practical hardening of crash-tolerant systems," in *USENIX ATC*, 2012.
- [62] D. Behrens, D. Kuvaishii, and C. Fetzer, "HardPaxos: Replication hardened against hardware errors," in *SRDS*, 2014.