# FAIL-STOP PROCESSORS: AN APPROACH TO DESIGNING FAULT-TOLERANT COMPUTING SYSTEMS

Pardeep Nehra
Department of Computer Science

**Abstract:** *We are increasingly dependent on services provided by computer systems and our vulnerability to computer failures is growing as a result. We would like these systems to be highly-available: they should work correctly and they should provide service without interruptions. There is a large body of research on replication techniques to implement highly-available systems. The idea is simple: instead of using a single server to implement a service, these techniques replicate the server and use an algorithm to coordinate the replicas. The algorithm provides the abstraction of a single service to the clients but the replicated server continues to provide correct service even when a fraction of the replicas fail. Therefore, the system is highly available provided the replicas are not likely to fail all at the same time.*

**Keywords:** BFT, Byzantine Fault Tolerant Systems, Quorum Replication**.**

**Introduction:** The problem is that research on replication has focused on techniques that tolerate benign faults these techniques assume components fail by stopping or by omitting some steps and may not provide correct service if a single faulty component violates this assumption. Unfortunately, this assumption is no longer valid because malicious attacks, operator mistakes, and software errors can cause faulty nodes to exhibit arbitrary behavior and they are increasingly common causes of failure. The growing reliance of industry and government on computer systems provides the motif for malicious attacks and the increased connectivity to the Internet exposes these systems to more attacks. Operator mistakes are also cited as one of the main causes of failure. In addition, the number of software errors is increasing due to the growth in size and complexity of software. Techniques that tolerate Byzantine faults provide a potential solution to this problem because they make no assumptions about the behavior of faulty components. There is a significant body of work on agreement and replication techniques that tolerate Byzantine faults. However, most earlier work either concern techniques designed to demonstrate theoretical feasibility that are too inefficient to be used in practice, or relies on unrealistic assumptions that can be invalidated easily by an attacker. For example, it is dangerous to rely on synchrony for correctness, i.e., to rely on known bounds on 11 message delays and process speeds. An attacker may compromise the correctness of a service by delaying non-faulty nodes or the communication between them until they are tagged as faulty and excluded from the replica group. Such a denial-of-service attack is generally easier than gaining control over a non-faulty node. This paper describes a new algorithm and implementation techniques to build highly-available systems that tolerate Byzantine faults. These systems can be used in practice because they perform well and do not rely on unrealistic assumptions. The next section describes our contributions in more detail.

## Contributions:

This paper presents BFT, a new algorithm for state machine replication that tolerates Byzantine faults. BFT offers both liveness and safety provided at most 1/3 out of a total of replicas is faulty. This means that clients eventually receive replies to their requests and those replies are correct according to linearizability. We used formal methods to specify the algorithm and prove its safety. Formal reasoning is an important step towards correctness because algorithms that tolerate Byzantine faults are subtle. BFT is the first Byzantine-fault-tolerant, state-machine replication algorithm that works correctly in asynchronous systems like the Internet: it does not rely on any synchrony assumption to provide safety. In particular, it never returns bad replies even in the presence of denial-of-service attacks. Additionally, it guarantees liveness provided message delays are bounded eventually. The service may be unable to return replies when a denial of service attack is active but clients are guaranteed to receive replies when the attack ends. Safety is provided regardless of how many faulty clients are using the service (even if they collude with faulty replicas): all operations performed by faulty clients are observed in a consistent way by non-faulty clients. Since BFT is a state-machine replication algorithm, it has the ability to replicate services with complex operations. This is an important defense against Byzantine-faulty clients: operations can be designed to preserve invariants on the service state, to offer narrow interfaces, and to perform access control. The safety property ensures faulty clients are unable to break these invariants or bypass access controls. Algorithms that restrict service operations to simple reads and blind writes are more vulnerable to Byzantine-faulty clients; they rely on the clients to order and group these simple operations correctly in order to enforce invariants. BFT is also the first Byzantine-fault-tolerant replication algorithm to recover replicas proactively in an asynchronous system; replicas are recovered periodically even if there is no reason to suspect that they are faulty. This allows the replicated system to tolerate any number of faults over the lifetime of the system provided less than 1/3 of the replicas become faulty within a window of vulnerability. The best that could be guaranteed previously was correct behavior if fewer than of the replicas failed during the lifetime of a system. Limiting the number of failures that can occur in a finite window is a synchrony assumption but such an assumption is unavoidable: since Byzantine-faulty replicas can discard the service state, we must bind the number of failures that can occur before recovery completes. To tolerate less than 1/3 faults over the lifetime of the system, we require no synchrony assumptions for safety. The window of vulnerability can be made very small (e.g., a few minutes) under normal conditions with a low impact on performance. Our algorithm provides detection of denial-of service attacks aimed at increasing the window; replicas can time how long a recovery takes and alert their administrator if it exceeds some pre-established bound. Therefore, integrity can be preserved even when there is a denial-of-service attack. Additionally, the algorithm detects when the state of a replica is corrupted by an attacker.

Unlike prior research in Byzantine fault tolerance in asynchronous systems, this paper describes a complete solution to the problem of building real services that tolerate Byzantine faults. For example, it describes efficient techniques to garbage collect information, to transfer state to bring replicas up-to-date, to retransmit messages, and to handle services with non-deterministic behavior. Additionally, BFT incorporates a number of important optimizations that allow the algorithm to perform well so that it can be used in practice. The most important optimization is the use of symmetric cryptography to authenticate messages. Public-key cryptography, which was cited as the major latency and throughput bottleneck in previous systems, is used only to exchange the symmetric keys. Other optimizations reduce the communication overhead: the algorithm uses only one message round trip to execute read-only operations and two to execute read-write operations, and it uses batching under load to amortize the protocol overhead for read write operations over many requests. The algorithm also uses optimizations to reduce protocol overhead as the operation argument and return sizes increase. BFT has been implemented as a generic program library with a simple interface. The BFT library can be used to provide Byzantine-fault-tolerant versions of different services. The paper describes the BFT library and explains how it was used to implement a real service: the first Byzantine-fault-tolerant distributed file system, BFS, which supports the NFS protocol. The paper presents a thorough performance analysis of the BFT library and BFS. This analysis includes a detailed analytic performance model. The experimental results show that BFS performs 2%faster to 24% slower than production implementations of the NFS protocol that are not replicated.

These results support our claim that the BFT library can be used to implement practical Byzantine fault-tolerant systems. There is one problem that deserves further attention: the BFT library (or any other replication technique) provides little benefit when there is a strong positive correlation between the failure probabilities of the different replicas. Our library is effective at masking several important types of faults, e.g., it can mask non-deterministic software errors and faults due to resource leaks. Additionally, it can mask other types of faults if some simple steps are taken to increase diversity in the execution environment. For example, the library can mask administrator attacks or mistakes if replicas are administered by different people. However, it is important to develop affordable and effective techniques to further reduce the probability of 1 /3 or more faults within the same window of vulnerability. In the future, we plan to explore existing independent implementations of important services like databases or file systems to mask additional types of faults.

## Related Works:

Some agreement and consensus algorithms tolerate Byzantine faults in asynchronous systems. However, they do not provide a complete solution for state machine replication, and furthermore, most of them were designed to demonstrate theoretical feasibility and are too slow to be used in practice. BFT’s protocol during normal-case operation is similar to the Byzantine agreement algorithm in. However, this algorithm is insufficient to implement state-machine replication: it guarantees that non-faulty processes agree on a message sent by a primary but it is unable to survive primary failures. Their algorithm also uses symmetric cryptography but since it does not provide view changes, garbage collection, or client authentication, it does not solve the problems that make eliminating public-key cryptography hard. The algorithm in solves consensus more efficiently than previous algorithms. It is possible to use this algorithm as a building block to implement state machine replication but the performance would be poor: it would require 7 message delays to process client requests and it would perform at least three public-key signatures in the critical path. The algorithm uses a signature sharing scheme to generate the equivalent of our quorum certificates. This is interesting: it could be combined with proactive signature sharing to produce certificates that could be exchanged among replicas even with recoveries.

## State Machine Replication:

Our work is inspired by Rampart and Secure Ring, which also implement state machine replication. However, these systems rely on synchrony assumptions for safety. Both Rampart and Secure Ring use group communication techniques with dynamic group membership. They must exclude faulty replicas from the group to make progress (e.g., to remove a faulty primary and elect a new one), and to perform garbage collection. For example, a replica is required to know that a message was received by all the replicas in the group before it can discard the message. So it may be necessary to exclude faulty nodes to discard messages. These systems rely on failure detectors to determine which replicas are faulty. However, failure detectors cannot be accurate in an asynchronous system they may misclassify a replica as faulty. Since correctness requires that less than 1/3 of group members be faulty, a misclassification can compromise correctness by removing a non-faulty replica from the group. This opens an avenue of attack: an attacker gains control over a single replica but does not change its behavior in any detectable way; then it slows correct replicas or the communication between them until enough are excluded from the group. It is even possible for this system to behave incorrectly without any compromised replicas. This can happen if all the replicas that send a reply to a client are removed from the group and the remaining replicas never process the client’s request. To reduce the probability of misclassification, failure detectors can be calibrated to delay classifying a replica as faulty. However, for the probability to be negligible the delay must be very large, which is undesirable. For example, if the primary has actually failed, the group will be unable to process client requests until the delay has expired, which reduces availability. Our algorithm is not vulnerable to this problem because it only requires communication between quorums of replicas. Since there is always a quorum available with no faulty replicas, BFT never needs to exclude replicas from the group. Public-key cryptography was the major performance bottleneck in Rampart and Secure Ring despite the fact that these systems include sophisticated techniques to reduce the cost of public-key cryptography at the expense of security or latency. These systems rely on public-key signatures to work correctly and cannot use symmetric cryptography to authenticate messages. BFT uses MACs to authenticate all messages and public-key cryptography is used only to exchange the symmetric keys to compute the MACs. This approach improves performance by up to two orders of magnitude without loosing security. Rampart and Secure Ring can guarantee safety only if fewer than 1/3 of the replicas are faulty during the lifetime of the system. This guarantee is too weak for long-lived systems. Our system improves this guarantee by recovering replicas proactively and frequently; it can tolerate any number of faults if less than 1/3 of the replicas become faulty within a window of vulnerability, which can be made small under normal load conditions with low impact on performance.

## Quorum Replication:

This work does not provide generic state machine replication. Instead, it offers a data repository with operations to read or write individual variables and to acquire locks. We can implement arbitrary operations that access any number of variables and can both read and write to those variables, whereas in Fleet it would be necessary to acquire and release locks to execute such operations. This makes Fleet more vulnerable to malicious clients because it relies on clients to group and order reads and blind writes to preserve any invariants over the service state. Fleet provides an algorithm with optimal resilience replicas to tolerate d faults but malicious clients can make the state of correct replicas diverge when this algorithm is used. This is interesting but it is not clear whether it will work in practice: a clever attacker can make compromised replicas appear to behave correctly until it controls more than d and then it is too late to adapt or respond in any other way. There are no published performance numbers for Fleet or Phalanx but we believe our system is faster because it has fewer message delays in the critical path and because of our use of MACs rather than public key cryptography. In Fleet, writes require three message round-trips to execute and reads require one or two round-trips. Our algorithm executes read-write operations in two round-trips and most read-only operations in one. Furthermore, all communication in Fleet is between the client and the replicas. This reduces opportunities for request batching and may result in increased latency since we expect that in most configurations communication between replicas will be faster than communication with the client. Therefore, we believe that partitioning the state by several state machine replica groups is a better approach to achieve scalability for most applications. Furthermore, it is possible to combine our algorithm with quorum systems that tolerate benign faults to improve on Fleet's scalability but this is future work.

**Conclusion:** The problem of efficient state transfer has not been addressed by previous work on Byzantine-fault tolerant replication. We present an efficient state transfer mechanism that enables frequent proactive recoveries with low performance degradation. The state transfer algorithm is also unusual because it is highly asynchronous. In replication algorithms for benign faults replicas typically retain a checkpoint of the state and messages in their log until the recovering replica is brought up-to-date. This could open an avenue for a denial-of-service attack in the presence of Byzantine faults. Instead, in our algorithm, replicas are free to garbage collect information and are minimally delayed by the recovery. They are both based on Merkle trees but the read-only SFS uses data structures that are optimized for a file system service. Another difference is that our state transfer handles modifications to the state while the transfer is in progress. Our technique to check the integrity of the replica's state during recovery is similar to those in except that we obtain the tree with correct digests from the other replicas rather than from a secure co-processor. The concept of a system that can tolerate more than d faults provided no more than d nodes in the system become faulty in some time window was introduced. But our algorithm is more general; it allows a group of nodes in an asynchronous system to implement an arbitrary state machine.

## References:

- [BEGx 94] M. Blum, W. Evans, P. Gemmel, S. Kannan, and M. Naor. Checking the Correctness of Memories. Algorithmica, 12:225–244, 1994.
- [CL00] M. Castro and B. Liskov. Proactive Recovery in a Byzantine-Fault-Tolerant System.
- In Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI), San Diego, CA, Oct. 2000.
- [FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382, Apr. 1985.
- [Gei95] K. Geiger. Inside ODBC. Microsoft Press, 1995.
- [GGJR99] J. Garay, R. Gennaro, C. Jutla, and T. Rabin. Secure Distributed Storage and Retrieval. Theoretical Computer Science, 1999.
- [GHMx 90] R. Guy, J. Heidemann, W. Mak, J. Page, T., G. Popek, and D. Rothneier. Implementation of the Ficus replicated file system. In USENIX Conference Proceedings, pages 63–71, June 1990.
- [Gif79] D. K. Gifford. Weighted voting for replicated data. In Proc. of the Seventh Symposium on Operating Systems Principles, pages 150–162, Pacific Grove, CA, Dec. 1979.
- ACM SIGOPS. [GK85] D. Gawlick and D. Kinkade. Varieties of concurrency control in IMS/VS fast path. Database Engineering, 8(2):63–70, June 1985.
- [GMR88] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks. SIAM Journal of Computing, 17(2):281–308, Apr. 1988.
- [Gra00] J. Gray. FT 101. Talk at the University of California at Berkeley, Nov. 2000.
- [Riv92] R. Rivest. The MD5 message-digest algorithm. Internet RFC-1321, Apr. 1992.
- [Rod00] R. Rodrigues. Private communication, 2000.
- [SHA94] National Institute of Standards and Technology (NIST). Announcement of Weakness in Secure Hash Standard, 1994.
- [Spu00] C. E. Spurgeon. Ethernet: The Definitive Guide. O’Reilly and Associates, 2000.
- [Sul00] B. Sullivan. Inside Europe’s cybersleuth central. MSNBC, Oct. 2000.