POLITECNICO DI TORINO
Repository ISTITUZIONALE

Which P Algorithm is Suitable for You? An Empirical Benchmark on Real Network Traces

Original
Which P Algorithm is Suitable for You? An Empirical Benchmark on Real Network Traces / Wang, Y., Schiller, E., Chiasserini, C.F.. - (2026). (30th International Conference on Principles of Distributed Systems (OPODIS 2026) Madrid (Spa) December 9-11, 2026).

Availability:
This version is available at: 11583/3017190 since: 2026-10-29T10:05:56Z

Publisher:
ACM

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

06 November 2026

Which $\Diamond\mathcal{P}$ Algorithm is Suitable for You? An Empirical Benchmark on Real Network Traces

Anonymous author
Anonymous affiliation

Abstract

Unreliable failure detectors (UFDs) are a fundamental abstraction for dependability and fault tolerance in distributed systems, supporting core distributed algorithms, such as consensus. Yet selecting a suitable detector remains difficult because existing implementations are evaluated under different environments, workloads, parameterizations, and quality-of-service (QoS) requirements. We address this problem by systematizing 13 $\Diamond\mathcal{P}$ and closely related $\Diamond\mathcal{P}$-style UFDs and providing a reusable benchmark for their empirical comparison. Our methodology evaluates detectors through common QoS metrics, such as detection time, mistake rate, and query accuracy, and combines same-input replay of network traces with live deployment for detectors whose active probing or cooperative behavior requires online execution. We then formalize failure-detector selection as a constrained algorithm-selection challenge, mapping operational requirements to suitable detector candidates. We instantiate this methodology under controlled variations of communication rate, host load, and crash regime. Applying the selection process shows that no detector is uniformly preferable across the evaluated settings. At relaxed detection-time budgets, 2W [Tomsic *et al.*, IPDPS 2015] is the most frequently selected detector, appearing in 12 of 16 scenario–metric combinations for $T_D \leq 500\,\text{ms}$, whereas stricter budgets yield substantially more scenario- and metric-dependent selections. The benchmark and selection methodology provide a reusable basis for practical detector selection and for establishing stronger analytical baselines for future failure-detection approaches, including learning-based detectors.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms; Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases Empirical evaluation, $\Diamond\mathcal{P}$ failure detectors, practical recommendations

Digital Object Identifier 10.4230/LIPIcs...

Best Student Paper Award: This paper is eligible for the Best Student Paper Award.



© Anonymous author(s);
licensed under Creative Commons License CC-BY 4.0



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consensus is a conceptually simple task: a group of distributed nodes must agree on a single value. Yet deterministic consensus cannot guarantee termination in a fully asynchronous system if even one node may crash [12]. The fundamental difficulty is that a node observing silence cannot determine whether another node has crashed or is merely slow. Consensus therefore requires additional assumptions or information about the system.

One approach is to strengthen the model with timing assumptions, as in partial synchrony [11]. Chandra and Toueg instead separate such assumptions from the upper-layer algorithm through the unreliable failure detector (UFD) abstraction [4]. The distributed algorithm reasons using the UFD's output, while the UFD encapsulates the mechanisms and assumptions used to infer crashes. A UFD may make mistakes provided that its outputs satisfy the completeness and accuracy properties required by its class.

Chandra and Toueg [4] define several UFD classes. In the standard crash-prone setting with a majority of correct processes, the eventual-leader abstraction Ω captures the weakest failure information needed for consensus [3]. We instead focus on the stronger Eventually Perfect class, $\Diamond\mathcal{P}$, which satisfies *strong completeness* (eventually every crashed process is permanently suspected by every correct process) and *eventual strong accuracy* (after some finite time, no correct process is suspected by any correct process) [4]. This per-process trust/suspect behavior makes false suspicions and detection time directly meaningful operational quantities. We therefore use $\Diamond\mathcal{P}$ as our conceptual anchor while also considering closely related detectors that exhibit $\Diamond\mathcal{P}$-style behavior, producing a trust/suspect output after convergence (Table 1).

Table 1 Relationship of the studied failure detectors to $\Diamond\mathcal{P}$. For compactness, we omit the suffix "FD" from most detector names. This naming convention applies to ED-FD [36], Adaptare-FD [9], 2W-FD [34], WD-FD [24], 2E-FD [22], Stab-FD, and Stab^C-FD [31].

Status	Detectors
Proven as $\Diamond\mathcal{P}$ detector	Bertier [1], 2E [22]
Accrual formulation with thresholdable $\Diamond\mathcal{P}$-style behavior	ϕ-FD [16], ED [36], WD [24]
QoS analysis with binary trust/suspect behavior	Chen [7], B-AFD [10] (extended by S-AFD [23])
Engineering-oriented $\Diamond\mathcal{P}$-style behavior	Satzger [30], BDBD [2], Adaptare [9], S-AFD [23], 2W [34], Stab and Stab^C [31]

Scope. The term "failure detector" is broader in systems research than the Chandra–Toueg oracle abstraction studied here. For example, Rapid [33] combines multi-source failure evidence with consistent membership changes, uKharon [14] integrates a multi-level detector with RDMA-based membership and reconfiguration, and FiDe [28] obtains highly reliable microsecond-scale crash reporting through datacenter-specific mechanisms. These systems target interfaces and guarantees that differ from the UFD abstraction considered here. Unless stated otherwise, we therefore use *failure detector* to mean the UFDs within our scope.

The eventual guarantees of $\Diamond\mathcal{P}$ are qualitative: they do not specify how quickly crashes are detected or how frequently correct processes are falsely suspected. Chen *et al.* [7] introduced implementation-independent quality-of-service (QoS) metrics for these operational properties,

allowing detectors to be compared directly through detection speed and accuracy metrics, without embedding each detector into a particular consensus protocol, whose design and assumptions would introduce additional effects. We therefore adopt failure-detector QoS as our common evaluation interface, adapting the metrics to our empirical setting.

Learning-based UFD-style detection has also been explored, from neural-network heartbeat prediction [21] to recent LSTM-based approaches [20]. Such approaches introduce training, inference, and deployment costs and choices absent from conventional UFDs, and existing proposals do not establish that the learned detector satisfies a Chandra–Toueg failure-detector class [25, 37]. Meaningful evaluation of learning-based detection therefore requires comparison against strong UFD alternatives under the same operational conditions and QoS requirements.

Research Gaps and Questions. To our knowledge, no open-source framework implements and evaluates $\Diamond\mathcal{P}/\Diamond\mathcal{P}$-style detectors under a common methodology. Existing evaluations fragmented across implementations, traces, platforms, workloads, parameterizations, and comparison sets, and individual detectors are typically compared against only a small subset of prior work. Consequently, published results cannot be combined into a reliable cross-detector ranking. The same limitation affects ML evaluations: outperforming Chen [7] or another individual baseline does not establish superiority over the best conventional UFD alternative for the same operating point. Moreover, Chen-style QoS metrics characterize detector behavior, but do not tell practitioners which detector to select for a given operational environment. An unsuitable detector may react too aggressively, causing unnecessary recovery or reconfiguration, or too conservatively, delaying fault recovery. We therefore ask:

RQ1: How do the QoS trade-offs of the studied detectors change with network conditions, communication rate, host load, crash frequency, and detection-time requirements?

RQ2: How can the observed trade-offs be mapped to suitable detector candidates under specified deployment and application constraints?

Approach. To address these research gaps, we provide a reusable benchmark for evaluating and selecting among $\Diamond\mathcal{P}$ and $\Diamond\mathcal{P}$-style failure detectors. We systematize the relevant design space and implement 13 detectors under a common QoS evaluation framework. Where possible, detectors are evaluated through offline replay of identical network traces; detectors requiring active probing or cooperation are evaluated through live deployment.

Building on this benchmark, we formalize failure-detector selection as a constrained instance of Rice's Algorithm Selection Problem [27]. Operational conditions characterize the instance, the studied detectors form the algorithm portfolio, and their measured QoS metrics characterize performance. Given a maximum acceptable detection time, we apply an ϵ-constraint formulation [15] to minimize the specified accuracy loss among feasible detectors.

We use a rapid-deployment data-collection tool on PlanetLab platform [8] to demonstrate the methodology and selection process under controlled variations of communication rate, host load, and crash regimes. Our open-source framework and UFD implementations are designed to be reusable and can be deployed on other real-world distributed systems.

Findings. Our results show no one-size-fits-all choice across the evaluated operating conditions and QoS objectives. Performance depends strongly on the required detection-time threshold and accuracy metric, while communication rate and observed execution conditions can further alter the attainable trade-offs (Observations 1–3, Section 5). Nevertheless, some patterns emerge. For example, at relaxed detection-time requirements (*e.g.*, $T_D \leq 500, 2000\,\text{ms}$), 2W is the most frequently selected detector, appearing in 12 of 16

and 10 of 16 evaluated scenario–metric combinations, respectively (Section 6.2). Under stricter detection-time budgets, however, the selected detector depends more strongly on the evaluated scenario and target metrics. These results further motivate our constraint-based selection procedure in Section 6, which maps measured QoS trade-offs and application requirements to suitable detector candidates rather than assuming a universal ranking.

Related Work. Raynal [26] surveys failure detectors as modular oracles characterized by completeness and accuracy, while Chaurasia and Verma [6] survey a broader range of failure models, system assumptions, and implementation strategies. These works organize the literature but do not provide a controlled empirical benchmark for selecting among multiple $\Diamond\mathcal{P}$ or $\Diamond\mathcal{P}$-style implementations.

Empirical failure-detection studies include mobile ad-hoc networks [13]. More recent work includes formal $\Diamond\mathcal{P}$ implementations for specialized communication structures, such as ADD networks [19, 35, 29] and vCube-based hierarchical monitoring [32], while IHFD [5] evaluates hierarchical health monitoring using PlanetLab traces. These approaches make communication topology, channel assumptions, or health aggregation part of the detector design, introducing benchmark dimensions orthogonal to the QoS-oriented design choices studied here. We therefore leave such topology- or architecture-coupled detection to extensions of the benchmark.

Our work complements these efforts by providing a broader reusable implementation corpus, common-input evaluation where possible, and a formal methodology for selecting among detectors according to operational conditions and application requirements.

Contributions and Implications. Our contributions are the following:

1. We provide a reusable benchmarking framework for evaluating $\Diamond\mathcal{P}$/$\Diamond\mathcal{P}$-style failure detectors, together with reusable artifacts for data collection, trace replay, and deployment. We further systematize the relevant design space and provides implementations of 13 detectors spanning timeout-based, accrual, probe-based, cooperative, and self-stabilizing designs (Section 2).
2. We present a common QoS evaluation methodology that combines same-input trace replay with live evaluation where active interaction is intrinsic to the detector (Sections 3–4). We then demonstrate its rapid deployment through a controlled comparison of these detectors on real-world testbeds. We characterize detector performance with communication rate, host workload, crash failures, and detection-time requirements, revealing distinct performance regimes rather than a universally preferable detector (Section 5).
3. We formalize failure-detector selection as a constrained algorithm-selection challenge, building on Rice's Algorithm Selection Problem [27]. This provides a principled basis for mapping operational conditions and empirical QoS measurements to detector choices under explicit application requirements. We use detection time as an ϵ-constraint [15] while separately optimizing mistake rate or query accuracy, and instantiate this formulation to derive a detector-selection process for the evaluated operational space (Section 6).

Together, these contributions have a broader implication for the evaluation of new failure detectors: because the best-performing detector varies with operating conditions and QoS objective, comparisons against a single or few baselines can give an incomplete picture of relative performance. This is particularly relevant to learning-based failure detection, where improvements should be assessed against appropriate UFD alternatives at the same operating point rather than an arbitrarily chosen conventional detector. Our benchmark and selection methodology provide a systematic way to identify such baselines.

For reproducibility, we pledge to release the source code and artifacts upon acceptance, including the data-collection and trace-replay components, detector implementations, deployment scripts, and collected trace dataset.

2 $\Diamond\mathcal{P}$ and $\Diamond\mathcal{P}$-Style Failure Detectors

This section reviews the studied failure detectors. Some are formal $\Diamond\mathcal{P}$ detectors, while others provide closely related trust/suspect behavior or optimize QoS, overhead, or adaptation without proving $\Diamond\mathcal{P}$ properties. Sections 3–5 evaluate them under a common benchmark.

2.1 Monitoring Model and Detector Interfaces

We adopt standard monitoring vocabulary. A distributed system consists of nodes Π, with monitoring nodes $M \subseteq \Pi$ observing monitored nodes $Q \subseteq \Pi$. Nodes may fail by crashing and do not recover. Detectors observe messages that may be delayed or lost. When formal $\Diamond\mathcal{P}$ guarantees are considered, the usual additional assumption is that communication delays eventually become bounded, although the bound does not need to be known [11, 4].

The studied detectors differ in both observation styles and output interfaces. Push-style detectors [7, 1, 16, 36, 24, 30, 34, 22, 31] rely on periodic heartbeat (HB) messages from monitored nodes. Pull-style detectors [9, 10, 23] actively send probes and wait for acknowledgments. Most detectors output a binary trust/suspect value, either for a single monitored node or for a set of nodes. Accrual detectors [16, 36, 24, 30] instead output a scalar suspicion level, leaving applications to choose a threshold for a binary decision.

2.2 Chronological Evolution

Figure 1 presents the main relationships among the studied failure detectors. For each detector, the timeline reports the earliest publication year we found, which may precede the peer-reviewed version. A solid arrow denotes documented technical lineage, *i.e.*, the target detector explicitly builds on or improves a prior detector. A dashed arrow denotes conceptual affinity, *i.e.*, the detectors use related techniques, but the articles do not state a direct lineage. Table 6 complements the timeline by summarizing each detector's design dimension, output interface, and main refinement.

Heartbeat-Based Detection and Timeout Estimation. Chen *et al.* [7] connect $\Diamond\mathcal{P}$-style heartbeat detection with explicit QoS reasoning. Their detector estimates the next heartbeat arrival from recent history, adds a constant safety margin, and suspects the monitored node if the heartbeat misses this freshness point. Later heartbeat-based detectors refine this design by changing the arrival estimator, the safety margin, or the speed/accuracy trade-off.

Bertier [1] keeps the expected-arrival estimator but replaces the constant safety margin with a Jacobson-style dynamic margin, allowing the timeout to react to recent estimation errors. 2W [34] maintains both a short-term and long-term history window, using the short window to react to abrupt delay changes and the long window to retain stable-period information. 2E [22] focuses on reducing estimator cost by using the last observed delay as the next-delay estimate and applying single exponential smoothing to the safety margin. Stab [31] calibrates the safety margin from observed link stability in WAN settings. Its cooperative extension, Stab^C, is discussed below.

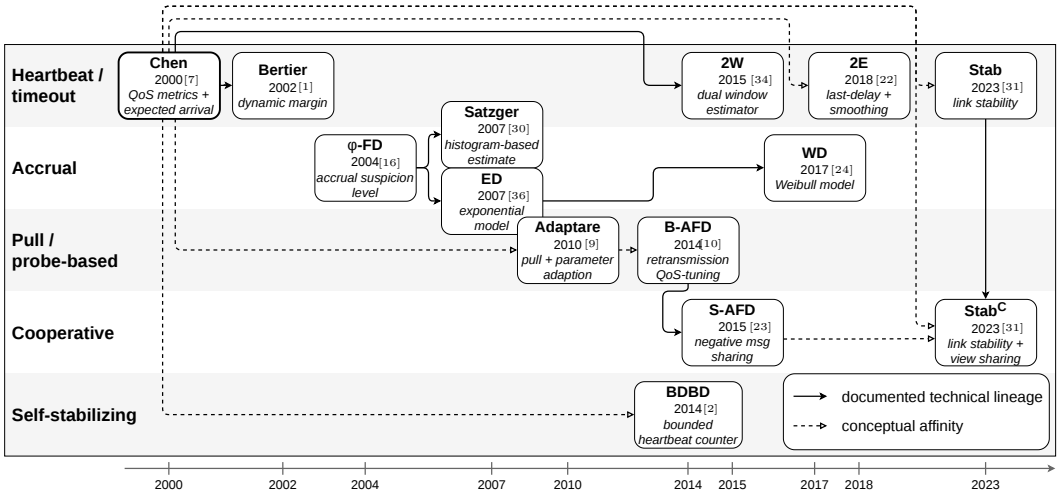


Figure 1 Timeline and main relationships among the studied $\Diamond\mathcal{P}$ and $\Diamond\mathcal{P}$-style detectors. Solid arrows denote documented technical lineage, while dashed arrows denote conceptual affinity.

Accrual Suspicion Level. ϕ-FD [16] shifts from binary timeout decisions to an accrual interface. Instead of returning trust or suspect, it outputs a continuous suspicion level derived from heartbeat inter-arrival behavior. Applications then choose thresholds according to their own speed and accuracy requirements. This separates monitoring from interpretation and allows one detector instance to support multiple application policies.

Satzger [30], ED [36], and WD [24] preserve the accrual interface but change the statistical model used to compute suspicion. Satzger estimates inter-arrival behavior with a histogram-based CDF. ED uses an exponential inter-arrival model, while WD uses a Weibull model intended to better match unstable cloud-computing networks. In this line, the main evolution is the mapping from delayed heartbeats to suspicion levels rather than the detector interface.

Probe-Based Detection. The pull, or probing, strategy predates Adaptare [9]. Bertier *et al.* [1] describe pinging as a classical alternative in which a monitor periodically sends "are you alive?" probes and waits for "I am alive" replies. Among the detectors studied here, however, pull-style detection first becomes a central design choice in Adaptare.

Adaptare moves QoS-oriented detection from passive heartbeat monitoring to active probing. It statistically characterizes the current network environment and adapts the interrogation period and timeout toward a configured QoS target under changing stochastic behavior. B-AFD [10] also uses pull-style detection, but adds a retransmission-based baseline model for lossy P2P networks. Rather than treating one missed probe as sufficient evidence, it models repeated probes within a detection period so that QoS parameters can be tuned under message loss. Since S-AFD [23] extends B-AFD and reports better performance, our empirical evaluation focuses on S-AFD, see [23, Sec. IV].

Cooperative Detection. S-AFD [23] extends the pull-style line with cooperation to reduce system-level detection overhead. Monitors still use B-AFD-style probing parameters, but selected host nodes share negative messages with other interested monitors when repeated failed probes indicate a crash. Stab^C [31] provides a heartbeat-based form of cooperation. The nodes piggyback suspected-node sets and link-stability estimates, allowing monitors to compare their local observations with views from nodes that may have more stable links.

Cooperation therefore plays two roles in the studied detectors: reducing redundant probing in S-AFD and improving suspicion quality in Stab^C.

Self-Stabilizing Failure Detection. Blanchard *et al.* [2] briefly describe the BDBD failure detector in the context of practically self-stabilizing Paxos. Its motivation differs from the QoS-oriented detectors above. Since transient faults may corrupt timestamp-like state, BDBD avoids timestamp comparisons and uses bounded heartbeat counters instead. Each processor repeatedly broadcasts heartbeats, and each receiver keeps a bounded counter per processor, resets the sender's counter upon heartbeat receipt, and increments the other counters up to a fixed maximum W. A processor is reported as failed when its counter reaches W. BDBD is therefore a bounded-state heartbeat variant intended for a self-stabilizing setting, rather than a timeout-estimation algorithm.

3 Evaluation Methodology

The studied detectors differ in interface, assumptions, and formal status. We therefore compare them through their observable suspicion histories. For detectors outputting binary decisions, we record trust/suspect transitions directly. For accrual detectors, we apply explicit thresholds to convert suspicion levels into binary outputs. These thresholds are treated as detector configurations. This gives a common comparison interface across different designs.

3.1 Trace-Based and Live Deployment Evaluation

Our framework uses two complementary evaluation modes: *offline trace-based* and *online live deployment* evaluation. In offline evaluation, a data-collection process records network traces, and a simulator replays those traces to offline detector implementations. This mode gives repeatable inputs and allows multiple detectors and configurations to be compared under the same recorded communication behavior.

In online evaluation, detectors are deployed directly on PlanetLab. We use this mode for Adaptare, S-AFD, and Stab^C, whose behavior depends on active probing or cooperation and therefore cannot be faithfully reconstructed from passive heartbeat traces alone. All other detectors are evaluated offline through trace replay. As discussed in Section 2.2, we evaluate S-AFD rather than B-AFD because S-AFD extends B-AFD with cooperative message sharing and reports better performance.

We use PlanetLab [8] to collect the network traces. Each execution uses four nodes: one monitoring node and three monitored nodes. The monitored nodes periodically send heartbeat messages to the monitoring node. For each experiment configuration, data collection runs for 24 hours, producing a trace for offline replay.

Offline detectors are evaluated by replaying the collected traces through a simulator. Replay preserves the recorded message-arrival behavior and allows the same trace corpus to be reused across detectors and parameter configurations. For detectors that output binary trust/suspect values, the simulator records the resulting suspicion history directly. For accrual detectors, we first apply an explicit threshold to convert the scalar suspicion level into a binary decision. These thresholds are treated as detector configuration parameters.

Online detectors run on the same PlanetLab setup. To make the duration comparable with trace collection, we match the total runtime, *i.e.*, online detectors are executed sequentially for each configuration, with 12 configurations run for approximately two hours each, yielding about 24 hours of live observations in total. This avoids overloading the PlanetLab nodes while still giving each online configuration a nontrivial observation window.

During online runs, we also collect the passive heartbeat stream used by the offline benchmark. Conversely, during trace-collection campaigns, we run the online detectors under the same deployment setup. Thus, online and offline experiments use the same platform and comparable background host-load conditions, while online execution additionally includes the runtime effects of active probing, UDP reception, timer updates, and cooperative protocol processing. These effects cannot be separated from algorithmic effects in the live measurements.

3.2 Crash Failure Simulation

We evaluate responsiveness through repeated crash-stop measurement rounds. In each round, a monitored node crashes by stopping all observable detector communication. The detector then records the elapsed time from crash onset to the first subsequent suspicion. After the round completes, the detector is reset and warmed up before the next round. Thus, each round models a crash-stop failure, while repetition across rounds is used only to obtain multiple detection time measurements.

Failures are injected probabilistically. We uses a sampling probability chosen to produce an average inter-crash interval of about 30 seconds, with each injected crash lasting 30 seconds. After each injected crash, detectors enter a reset and warmup phase before the next crash-injection round. Metrics are not recorded during reset or warmup periods.

In offline experiments, failures are injected into replayed traces. When a node is marked as crashed, all subsequent messages from that node are suppressed for the duration of the crash interval, so the replay reflects what the monitor would observe if the node stopped communicating. Crashes are injected only after an initial warmup period, ensuring that detectors have enough prior observations to establish their normal operating state.

In live deployments, failures are injected by the running detector processes. Each node independently samples whether to enter a crash interval, with the probability scaled by the number of nodes so that the system-wide crash frequency remains approximately one crash every 30 seconds. During the crash interval, the node stops participating normally. After the interval, the experiment enters reset and warmup before metric collection resumes.

The main distinction is therefore where the failure is applied. Offline injection modifies an existing event trace by hiding communication from the crashed node, which gives repeatable and controlled experiments. Crash-injection during live deployment affects live protocol execution, so nodes must coordinate crash-start and recovery events, and metric collection is paused during a warmup period after recovery to avoid measuring transient reset behavior.

4 Evaluation Strategy

We decompose RQ1, which asks how the studied detectors perform under different operational conditions, into the three subquestions below. RQ2 concerns detector selection under specified constraints and is addressed in Section 6.

RQ1.1 How does the communication rate affect the latency/accuracy trade-off of $\Diamond\mathcal{P}$ and $\Diamond\mathcal{P}$-style detectors, with and without frequent crash failures?

RQ1.2 How does the host-side processing load affect detector mistake rates and query accuracy, and how does this interaction change under frequent crash failures?

RQ1.3 How do detectors that require active probing or cooperation behave in live deployment compared with offline detectors under comparable workloads and failure scenarios?

4.1 Evaluation Criteria

We evaluate detectors using QoS metrics adapted from Chen *et al.* [7] and operationalized for our finite executions, repeated crash-injection rounds, and sampled detector histories. *Detection time* T_D is the elapsed time from an injected node crash until the detector first suspects the node. *Mistake rate* λ_M is the rate of false-suspicion transitions, *i.e.*, transitions from trusting a correct node to incorrectly suspecting it. *Query accuracy probability* Q_A is the fraction of sampled times for which the detector output agrees with the known state of the monitored node.

T_D captures responsiveness, whereas λ_M and Q_A capture complementary aspects of accuracy. In an environment with injected crash-failures, T_D is measured from crash onset to the first suspicion. When there is no crash-failure, we estimate T_D from the detector's timeout or threshold-induced suspicion point.

We compute λ_M as the number of false-suspicion transitions divided by the effective metric-collection runtime. We compute Q_A as the fraction of correct detector states among all query samples. In live deployment, query accuracy is sampled once per second. The metrics exclude the reset and warmup periods. For readability, figures and tables report $Q_E = 1 - Q_A$, so lower values indicate better performance for T_D, λ_M, and $1 - Q_A$.

For a fixed scenario and error metric, we call a detector configuration optimal if no other evaluated configuration achieves both no larger T_D and no larger error, with a strict improvement in at least one of the two quantities. The resulting set of optimal configurations defines the Pareto frontier, which we compute separately for (T_D, λ_M) and $(T_D, 1 - Q_A)$.

Offline replay can produce multiple values for the same metric across monitored nodes and independent crash-stop rounds. We report the average over values obtained under the same detector, message rate, host-load setting, crash environment, and detector configuration.

4.2 Evaluation Environment

All experiments use four PlanetLab nodes: two in Sweden, one in Poland, and one in Belgium. Each non-cooperative execution uses one monitoring node and three monitored nodes. For cooperative detectors, all nodes perform both roles. Trace collection and live deployment follow Section 3.

We vary three experimental factors: communication rate, host load, and crash regime. The communication rate is either 100 Hz or 10 Hz, corresponding to inter-message intervals of 10 ms and 100 ms. For push-based detectors, this interval is the heartbeat interval, and for pull-based detectors, it is the probing interval.

We use three host-load settings: *minimal*, *moderate*, and *noticeable*. Under minimal load, nodes exchange only detector-related messages. Under moderate and noticeable load, they also compute and attach host-load indicators, such as CPU and memory usage, adding approximately 1 ms and 20 ms of computation, respectively.

Crash behavior follows Section 3.2. Offline detectors are evaluated under no-crash (NC) and crash-injected (CI) executions. In particular, CI replays the same trace as NC with crash injection enabled. NC isolates false suspicions, while CI measures behavior under repeated independent crash-stop rounds. Online detectors are evaluated only under CI because live deployments are non-replayable and time-limited. We allocate the online budget to CI to preserve per-configuration observation time while capturing both detection time and false suspicions during live probing or cooperation.

Detector configurations are selected to expose the trade-off between detection time and accuracy. For timeout-based detectors, we vary parameters affecting timeout or detection

latency; for accrual detectors, we vary the suspicion threshold converting scalar suspicion levels into binary decisions. Online detectors use compact configuration sets so that each configuration can run for about two hours. Detailed configurations are provided in Table 7.

Offline detectors are evaluated over the collected traces, while online configurations are executed sequentially to avoid overloading the nodes. Two S-AFD results are omitted because extreme parameter values yielded no valid detection-time measurements.

4.3 Experiment Plan

We organize the evaluation into three experiments using the QoS metrics from Section 4.1.

Experiment 1. Experiment 1 addresses RQ1.1. We evaluate the offline heartbeat-based and accrual detectors at 100 Hz and 10 Hz under NC and CI with moderate host load. For each configuration, we measure average λ_M and $1 - Q_A$ as functions of T_D, then compare the best attainable accuracy across detection-time thresholds.

Experiment 2. Experiment 2 addresses RQ1.2. We fix the heartbeat rate at 100 Hz and evaluate the same offline detectors under minimal, moderate, and noticeable host load in NC and CI, comparing λ_M and $1 - Q_A$ across detection-time thresholds.

Experiment 3. Experiment 3 addresses RQ1.3. We compare detectors requiring active probing or cooperation with Pareto-optimal offline configurations under moderate host load, CI, and 100 Hz and 10 Hz communication rates. We use the same QoS metrics and detection-time thresholds as Experiments 1 and 2. As discussed in Section 3.1, we evaluate S-AFD instead of B-AFD.

5 Experimental Results

We report the results of the three experiments from Section 4.3. We conclude each experiment with a boxed observation that states the corresponding answer to RQ1.

5.1 Experiment 1: Impact of Heartbeat Rate

To answer RQ1.1, we compare offline detectors at 100 Hz and 10 Hz under no-crash (NC) and crash-injected (CI) environments. Figure 2 shows λ_M and $1 - Q_A$ as functions of T_D under CI environment. Full plots for both CI and NC environments are provided in Appendix B, since they are visually similar. We focus on configurations on the Pareto-frontier, since they capture the best observed trade-offs between detection speed and accuracy.

The expected latency/accuracy trade-off appears in both environments, with larger detection times generally corresponding to lower mistake rates and lower query error probabilities. The heartbeat rate primarily affects the short detection times. At 100 Hz, the earliest observed detections occur between 10 ms and 20 ms, whereas at 10 Hz they occur around 50 ms. This follows from the shorter inter-heartbeat interval, which allows monitors to use shorter waiting periods before suspecting a node.

Table 2 lists the best λ_M and $1 - Q_A$ values under increasingly relaxed detection-time thresholds $T_D \leq 50, 100, 200, 500$, and 2000 ms. NC and CI differ mainly at short detection times. In CI, some suspicions coincide with injected crash-stop rounds and are therefore correct rather than false, whereas in NC, every suspicion is necessarily false. This explains why CI can yield lower or equal λ_M and $1 - Q_A$ values for $T_D \leq 200$ ms. As T_D increases, NC and CI converge.

For strict detection budgets, 100 Hz improves accuracy substantially. In NC at $T_D \leq 100$ ms, 10 Hz gives $\lambda_M = 7.27$ and $1 - Q_A = 0.16$, while 100 Hz gives $\lambda_M = 0.072$ and

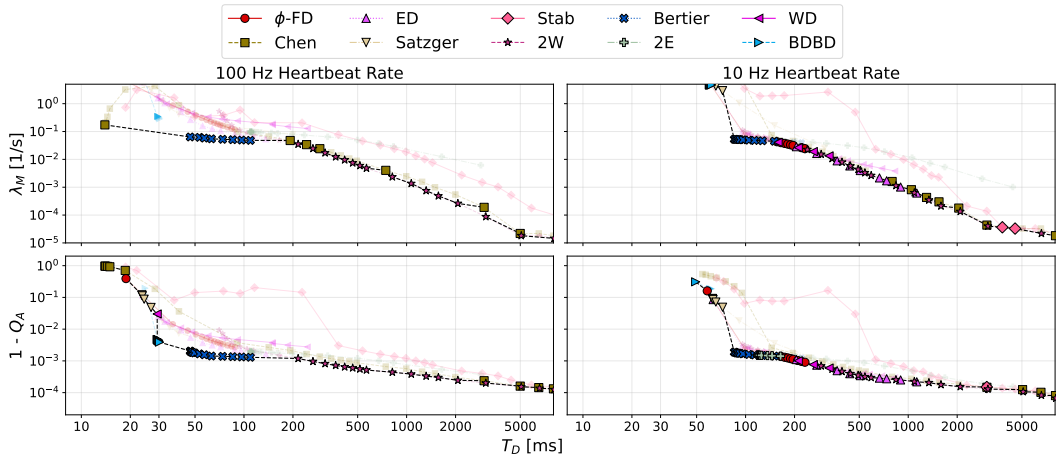


Figure 2 QoS metrics (λ_M and $1 - Q_A$) for offline detectors at 100 Hz and 10 Hz under CI environment. All axes use log scale.

Table 2 Lowest λ_M and $1 - Q_A$ achieved under each heartbeat rate, crash environment, and detection-time threshold.

HB	Env.	Metric	Detection Time Threshold				
			$T_D \leq 50$	$T_D \leq 100$	$T_D \leq 200$	$T_D \leq 500$	$T_D \leq 2000$
100 Hz	NC	λ_M	0.173	0.072	0.048	0.0074	$4.9 \cdot 10^{-4}$
100 Hz	CI	λ_M	0.064	0.049	0.048	0.0074	$4.9 \cdot 10^{-4}$
10 Hz	NC	λ_M	N.A.	7.27	0.047	0.0058	$2.1 \cdot 10^{-4}$
10 Hz	CI	λ_M	7.51	0.050	0.032	0.0044	$2.1 \cdot 10^{-4}$
100 Hz	NC	$1 - Q_A$	0.0039	0.0021	0.0013	$6.0 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
100 Hz	CI	$1 - Q_A$	0.0018	0.0013	0.0013	$6.0 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
10 Hz	NC	$1 - Q_A$	N.A.	0.16	0.0016	$4.0 \cdot 10^{-4}$	$1.8 \cdot 10^{-4}$
10 Hz	CI	$1 - Q_A$	0.31	0.0017	0.0011	$3.5 \cdot 10^{-4}$	$1.8 \cdot 10^{-4}$

$1 - Q_A = 0.0021$, corresponding to reductions of 99% and 98.7%. This advantage diminishes once $T_D \geq 200$ ms. At $T_D \leq 500$ ms in NC, 10 Hz reaches $\lambda_M = 0.0058$ and $1 - Q_A = 4.0 \cdot 10^{-4}$, compared with $\lambda_M = 0.0074$ and $1 - Q_A = 6.0 \cdot 10^{-4}$ at 100 Hz. Thus, when applications can tolerate longer detection times, a lower heartbeat rate can be comparable to or better.

▶ **Observation 1** (RQ1.1 – communication-rate effect). *In our traces,* 100 Hz *significantly improves accuracy primarily under strict detection-time budgets. In NC at* $T_D \leq 100$ ms*, increasing* 10 Hz *and* 100 Hz *reduces* λ_M *by 99.0% and* $1 - Q_A$ *by 98.7%. Once the detection-time budget is relaxed, this advantage disappears. In NC at* $T_D \leq 500$ ms*,* 10 Hz *reduces* λ_M *by approximately 22% and* $1 - Q_A$ *by 33% relative to* 100 Hz*. Thus, a higher message rate mainly improves accuracy in the short-latency operating environment rather than uniformly improving attainable accuracy.*

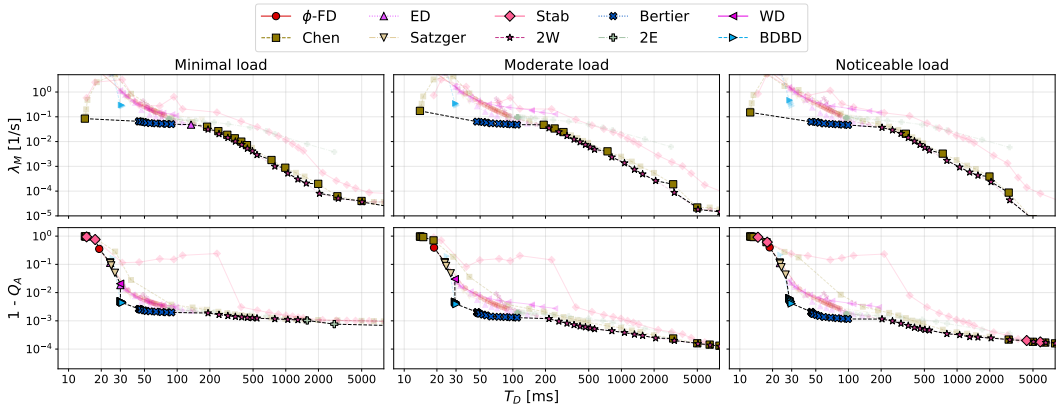


Figure 3 QoS metrics (λ_M and $1 - Q_A$) for offline detectors under minimal, moderate, and noticeable host load under CI environment. All axes use log scale.

5.2 Experiment 2: Impact of Host Load

To answer RQ1.2, we fix the heartbeat rate at 100 Hz and compare minimal, moderate, and noticeable host-load settings. Figure 3 reports λ_M and $1 - Q_A$ under CI. The full results for both CI and NC are found in Appendix B. We use 100 Hz because shorter heartbeat intervals make host-side processing effects more visible.

Table 3 Summary statistics of heartbeat inter-arrival times in milliseconds under different host loads and 100 Hz heartbeat rate.

Load	Mean	Median	Std. dev.	Variance	Min	Max
minimal	10.105	10.084	43.608	1901.642	0.036	219720
moderate	10.141	10.077	14.427	208.146	0.050	25423
noticeable	10.135	10.073	10.993	120.837	0.039	4267

Moderate and noticeable load behave similarly in both environments, whereas minimal load differs. In particular, the minimal-load query-error curves flatten earlier, around $1 - Q_A = 10^{-3}$, while moderate and noticeable load converge closer to 10^{-4}. A plausible explanation is the greater heartbeat inter-arrival variability in the minimal-load trace. As shown in Table 3, the three traces have similar mean and median inter-arrival times, but the minimal-load trace has much higher variance (1901.6 ms^2, compared with 208.1 ms^2 and 120.8 ms^2) and a larger maximum gap (219.7 s, compared with 25.4 s and 4.3 s). Because $1 - Q_A$ measures time spent in an incorrect state, these rare long gaps can prolong false suspicions until the next heartbeat, explaining the higher minimal-load plateau.

Table 4 reports the best λ_M and $1 - Q_A$ under NC and CI environments for the detection-time threshold $T_D \leq 2000$ ms. We omit shorter thresholds because they follow the trends observed in earlier experiments: CI mainly improves performance at short detection times, while NC and CI converge as the detection-time threshold increases.

The clearest host-load effect appears in query error. At $T_D \leq 2000$ ms, $1 - Q_A$ converges to approximately 0.001 under minimal load, compared with $3.0 \cdot 10^{-4}$ under moderate load and $2.6 \cdot 10^{-4}$ under noticeable load, a reduction of up to 74%. Mistake rate is less uniform. At the same threshold, moderate and noticeable load increase λ_M by up to 145% relative to minimal load. Figure 3 nevertheless suggests that, for even larger T_D, the minimal-load

Table 4 Lowest mistake rates λ_M and query error probabilities $1 - Q_A$ achieved under each host-load setting, crash environment, and detection time threshold $T_D \leq 2000$ ms.

Load	Env.	λ_M	$1 - Q_A$
minimal	NC	$2.0 \cdot 10^{-4}$	0.0011
minimal	CI	$2.0 \cdot 10^{-4}$	0.001
moderate	NC	$4.9 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
moderate	CI	$4.9 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
noticeable	NC	$3.9 \cdot 10^{-4}$	$2.6 \cdot 10^{-4}$
noticeable	CI	$3.9 \cdot 10^{-4}$	$2.6 \cdot 10^{-4}$

curve flattens above 10^{-5}, while the other settings converge at values around or below 10^{-5}.

▶ **Observation 2** (RQ1.2 – host-side load effect). *Across the collected traces, increasing the configured host load does not produce a monotonic degradation in detector accuracy. At $T_D \leq 2000$ ms, the moderate- and noticeable-load traces exhibit substantially lower query error (up to 74% in CI and 76% in NC) than the minimal-load trace, while their mistake rates can be higher (up to 145%). Table 3 shows that the minimal-load trace also contains substantially heavier inter-arrival variations. We therefore interpret the result as evidence that inter-arrival variability can dominate the configured host-load level, rather than as evidence that additional host load improves failure detection.*

5.3 Experiment 3: Comparison of Online and Offline Detectors

To answer RQ1.3, we compare the online detectors (Adaptare, S-AFD, and Stab^C) with the Pareto-optimal offline detectors under moderate host load and CI, see Figure 4.

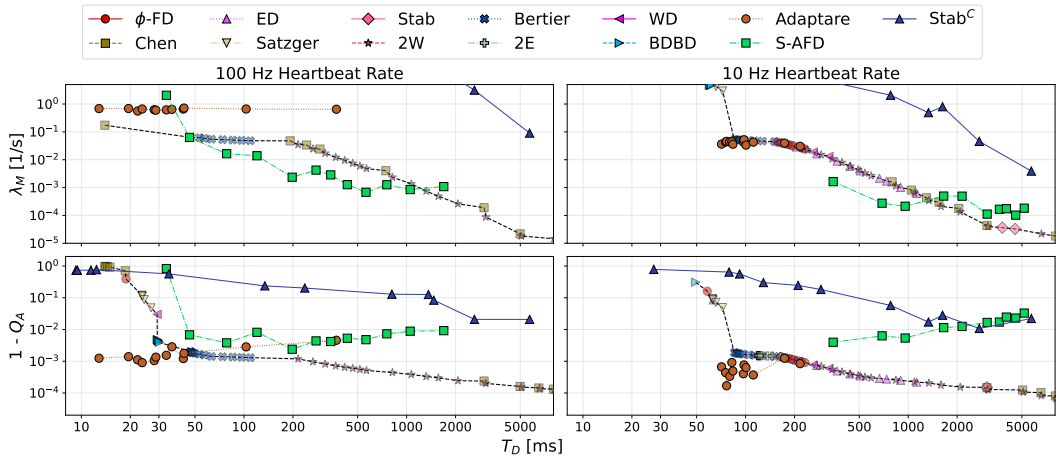


Figure 4 QoS metrics (λ_M and $1 - Q_A$) for online detectors and Pareto-optimal offline detectors at 100 Hz and 10 Hz under CI. All axes use log scale.

At 100 Hz, S-AFD performs well on mistake rate. It attains lower observed λ_M than most offline detectors for $T_D > 50$ ms and degrades mainly for $T_D > 1000$ ms. At $T_D \leq 500$ ms, S-AFD reduces λ_M by more than 80% relative to the second-best detector. However, it does not improve query error relative to the strongest offline detectors.

Adaptare shows the opposite trade-off at 100 Hz. It provides fast detection and low query error, with approximately $1 - Q_A = 0.0013$ at $T_D \approx 15$ ms, but its mistake rate remains high for most tested configurations, around $\lambda_M \approx 0.6$. Stab^C has mistake rates and query error probabilities that are often an order of magnitude higher than those of other detectors at similar detection times, showing that it does not perform well under our setting.

These differences may reflect both the active/cooperative algorithms themselves and execution effects present in live deployment. At 100 Hz, scheduling delays and network jitter are large relative to the message interval and can produce premature timeouts or frequent false suspicions. S-AFD may be less affected because its parameters control retry spacing within a detection period rather than a sustained decision rate. Namely, it sends a bounded number of probes, stops after the first acknowledgment, and suspects only after repeated failures within the detection period.

At 10 Hz, all three online detectors show lower mistake rates. S-AFD remains strong in terms of λ_M, reaching below 10^{-3} for $T_D > 500$ ms, but its earliest detection time increases to approximately 340 ms. Its query error also worsens as detection time increases, likely because longer detection periods extend incorrect-state durations. Stab^C slightly improves its mistake rate but remains worse than stronger alternatives.

Adaptare performs particularly well among the online detectors at 10 Hz. It achieves $1-Q_A$ between 10^{-3} to 10^{-4} for most configurations, with detection times around 70 to 150 ms. Its mistake rates also are comparable to or better than the Pareto-optimal offline detectors at similar detection times. Relative to the offline detectors at $T_D \leq 100$ ms, Adaptare reduces $1 - Q_A$ by up to approximately 90% while maintaining comparable mistake rates.

▶ **Observation 3** (RQ1.3 – online detectors). *In the live CI experiments, S-AFD provides the strongest mistake-rate behavior at* 100 Hz*, reducing* λ_M *by more than 80% at* $T_D \leq 500$ ms*, whereas Adaptare performs strongly at* 10 Hz *around* $T_D \leq 100$ ms*, reducing query error by up to approximately 90% while maintaining comparable mistake rates. No live detector is uniformly best across message rates and accuracy metrics. Because these detectors are executed live rather than replayed on identical traces, the comparison characterizes their deployed behavior under comparable conditions, but does not isolate algorithmic effects from live scheduling and network jitter.*

6 From Benchmark Results to Detector Selection

RQ2 asks how empirical QoS trade-offs can be translated into suitable detector candidates under specified deployment and application constraints. We formulate this as Rice's *Algorithm Selection Problem*, which selects an algorithm from a portfolio based on problem-instance characteristics and observed algorithm performance [27, 18, 17]. Here, the portfolio is the studied failure detectors, and an instance is characterized by deployment conditions such as communication rate, host-load setting, and crash regime. We use this framework as a decision model for the evaluated deployment scenarios.

6.1 Selection Method

Failure-detector selection involves competing QoS objectives: shorter detection time can trade off against accuracy. We therefore apply the ϵ-constraint approach [15], treating the application's maximum acceptable detection time as a constraint and optimizing an accuracy metric among configurations satisfying it rather than combining detection time and accuracy in an arbitrary weighted score.

Table 5 Detector-level selection results by scenario, accuracy metric, and detection-time budget. For each threshold, the table reports the detector achieving the lowest error among Pareto-optimal configurations satisfying the detection-time constraint.

Scenario	Metric	Detection Time Threshold				
		$T_D \leq 50$	$T_D \leq 100$	$T_D \leq 200$	$T_D \leq 500$	$T_D \leq 2000$
100-min-NC	λ_M	Chen	ED	Chen	Chen	Chen
	$1 - Q_A$	BDBD	ED	Bertier	2W	2W
100-min-CI	λ_M	Bertier	Bertier	2W	2W	Chen
	$1 - Q_A$	Bertier	Bertier	2W	2W	2E
100-mod-NC	λ_M	Chen	ED	Chen	2W	2W
	$1 - Q_A$	BDBD	ED	Bertier	2W	2W
100-mod-CI	λ_M	S-AFD	S-AFD	S-AFD	S-AFD	2W
	$1 - Q_A$	Adaptare	Adaptare	Adaptare	2W	2W
10-mod-NC	λ_M	N.A.	ϕ-FD	ϕ-FD	2W	2W
	$1 - Q_A$	N.A.	ϕ-FD	ϕ-FD	2W	2W
10-mod-CI	λ_M	BDBD	Adaptare	ϕ-FD	S-AFD	2W
	$1 - Q_A$	BDBD	Adaptare	Adaptare	Adaptare	Adaptare
100-not-NC	λ_M	Chen	ED	Bertier	2W	Chen
	$1 - Q_A$	BDBD	ED	Bertier	2W	2W
100-not-CI	λ_M	Bertier	Bertier	Bertier	2W	Chen
	$1 - Q_A$	Bertier	Bertier	Bertier	2W	2W

Let $s = (r, l, c)$ denote an evaluated deployment scenario, characterized by communication rate r, host load l, and crash regime c. Let $\mathcal{A}$ be the portfolio of benchmarked detector algorithms, and let $\Theta_a(s)$ be the set of configurations of detector a evaluated in scenario s. The available candidate set is therefore $C_s = \{(a, \theta) : a \in \mathcal{A}, \theta \in \Theta_a(s)\}$. Let B be the maximum acceptable detection time and $m \in \{\lambda_M, 1 - Q_A\}$ be the application's accuracy error metric to minimize. We select $X^*(s, B, m) = \left(\arg\min_{x \in \mathcal{C}_s : T_D(x) \leq B} m(x)\right)$. Thus, $X^*(s, B, m)$ is the detector–configuration pair that minimizes the chosen accuracy error metric m among all candidates whose detection time does not exceed B.

Equivalently, the search may be restricted to the corresponding Pareto frontier because a dominated configuration cannot improve the constrained objective. Each x is a detector–configuration pair. Since RQ2 concerns detector choice rather than parameter tuning, Table 5 reports the detector identity associated with the selected pair, while the benchmark retains the configuration required to reproduce the measurement.

6.2 Applying the Method to Our Benchmark

We apply the selection method to the scenarios evaluated in Section 5. For each scenario and accuracy metric, we (i) compute the corresponding Pareto frontier, (ii) retain points satisfying the detection-time threshold $T_D \leq B$, and (iii) report the detector associated with the lowest remaining error. We apply this procedure for $B \in \{50, 100, 200, 500, 2000\,\text{ms}\}$. If no evaluated configuration satisfies a budget, the result is reported as N.A. Table 5 reports the resulting detector-level selections. Notably, no evaluated case yielded multiple candidates, so we do not specify a tie-breaking convention.

Using Table 5, the selection is direct when an application specifies its accuracy objective

and operational requirements. For example, we select using λ_M if mistake rate is the primary concern, or $1 - Q_A$ if time spent in an incorrect state is more important. In particular, when different detectors are preferred by the two metrics, the appropriate choice depends on the application's accuracy objective.

Applying the selection rule yields two recurring patterns for our results. First, at higher detection-time thresholds, 2W is the most frequently selected detector: in 12 of 16 scenario–metric combinations at $T_D \leq 500\,\text{ms}$ and 10 of 16 at $T_D \leq 2000\,\text{ms}$. Second, under stricter detection-time budgets ($B \leq 200\,\text{ms}$), no detector is uniformly preferred: the selected candidate depends on the evaluated deployment conditions and, in several cases, on whether the application minimizes mistake rate or query error.

Answer to RQ2. We formulate failure-detector selection as a constraint-based algorithm-selection challenge [27]. Given an operational scenario, a detection-time threshold, and an accuracy objective, we restrict the corresponding Pareto frontier to feasible configurations and select the detector with minimum error. Applying this method to our benchmark yields the detector-level selections in Table 5. In particular, 2W is selected most frequently at relaxed detection-time thresholds, whereas stricter thresholds yield scenario- and metric-dependent choices. The same selection procedure can be rerun using benchmark measurements collected from other deployment environments.

7 Conclusion

Three decades after Chandra and Toueg introduced unreliable failure detectors [4], many practical implementations have emerged, but their evaluations remain fragmented across environments, workloads, parameterizations, and comparison sets. Until now, this fragmentation hindered both detector selection for a given deployment and the establishment of strong analytical baselines for newer approaches, including ML-based detectors.

We addresses this gap through a combined systematization, empirical benchmark, and detector-selection methodology. We systematize 13 $\Diamond\mathcal{P}$ and $\Diamond\mathcal{P}$-style detectors under a common QoS evaluation framework comparing detection time and accuracy. We formalize detector selection as a constraint-based algorithm-selection challenge [27], mapping measured QoS trade-offs to candidates under explicit deployment conditions, detection-time requirements, and accuracy objectives. We demonstrate rapid deployment of this methodology on PlanetLab, combining same-input trace replay with live experiments for detectors requiring online execution for probing or cooperation. Our results reinforce that detector selection should be conditioned on operational requirements rather than reduced to a universal ranking. No detector is uniformly preferable across the evaluated detection-time and accuracy objectives: relaxed detection-time threshold yield more consistent choices, whereas stricter thresholds require scenario- and metric-specific selection. Our benchmarking framework and selection procedure can be rerun in other deployment environments, providing both a reusable basis for practical UFD selection and a broader baseline for evaluating future failure-detection approaches, including learning-based detectors.

References

1. Marin Bertier, Olivier Marin, and Pierre Sens. Implementation and performance evaluation of an adaptable failure detector. In *DSN*, pages 354–363. IEEE Computer Society, 2002.
2. Peva Blanchard, Shlomi Dolev, Joffroy Beauquier, and Sylvie Delaët. Practically self-stabilizing paxos replicated state-machine. In *NETYS*, Lecture Notes in Computer Science, pages 99–121. Springer, 2014.
3. Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.
4. Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.
5. B. Chaurasia, A. Verma, and P. Verma. IHFD: An improved heartbeat-based failure detector of SVG class for partially synchronous hierarchical distributed systems. *Complex & Intelligent Systems*, 2026. doi:10.1007/s40747-026-02423-z.
6. Bhavana Chaurasia and Anshul Verma. A comprehensive study on failure detectors of distributed systems. *Journal of Scientific Research*, 64:250–260, 07 2020.
7. Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera. On the quality of service of failure detectors. *IEEE Trans. Computers*, 51(5):561–580, 2002.
8. Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.
9. Monica Dixit and António Casimiro. Adaptare-FD: A dependability-oriented adaptive failure detector. In *SRDS*, pages 141–147. IEEE Computer Society, 2010.
10. Jian Dong, Xiao Ren, Decheng Zuo, and Hongwei Liu. An adaptive failure detector based on quality of service in peer-to-peer networks. *Sensors*, 14(9):16617–16629, 2014.
11. Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.
12. Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
13. Roy Friedman and Galya Tcharny. Evaluating failure detection in mobile ad-hoc networks. *Int. J. Pervasive Comput. Commun.*, 5(4):476–496, 2009. doi:10.1108/17427370911008857.
14. Rachid Guerraoui, Antoine Murat, Javier Picorel, Athanasios Xygkis, Huabing Yan, and Pengfei Zuo. ukharon: A membership service for microsecond applications. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 101–120. USENIX Association, 2022.
15. Yacov Y. Haimes, Leon S. Lasdon, and David A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(3):296–297, July 1971. doi:10.1109/TSMC.1971.4308298.
16. Naohiro Hayashibara, Xavier Défago, Rami Yared, and Takuya Katayama. The Φ accrual failure detector. In *SRDS*, pages 66–78. IEEE Computer Society, 2004.
17. Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evol. Comput.*, 27(1):3–45, 2019. doi:10.1162/EVCO_A_00242.
18. Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Mag.*, 35(3):48–60, 2014. doi:10.1609/AIMAG.V35I3.2460.
19. Saptaparni Kumar and Jennifer L. Welch. Implementing ♢P with bounded messages on a network of *ADD* channels. *Parallel Process. Lett.*, 29(1):1950002:1–1950002:12, 2019. doi:10.1142/S0129626419500026.
20. Xiaonan Li and Olivier Marin. Towards implementing ML-based failure detectors. *CoRR*, abs/2210.00134, 2022.

21 Fábio Lima and Raimundo Macêdo. Adapting failure detectors to communication network load fluctuations using snmp and artificial neural nets. In *Dependable Computing*, pages 191–205. Springer Berlin Heidelberg, 2005.

22 Jiaxi Liu, Zhibo Wu, Jian Dong, Jin Wu, and Dongxin Wen. An energy-efficient failure detector for vehicular cloud computing. *PLOS ONE*, 13(1):1–17, 01 2018.

23 Jiaxi Liu, Zhibo Wu, Jinghui Lan, Jian Dong, Jin Wu, and Jiaxin Yu. A low-overhead cooperative failure detector. In *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*, pages 811–815, 2015.

24 Jiaxi Liu, Zhibo Wu, Jin Wu, Jian Dong, Yao Zhao, and Dongxin Wen. A Weibull distribution accrual failure detector for cloud computing. *PLOS ONE*, 12(3):1–16, 03 2017.

25 Wenbin Qi and Jingyi Zhao. Machine learning applied to failure detection. In Huajun Dong and Shijie Jia, editors, *Sixth International Conference on Computer Information Science and Application Technology (CISAT 2023)*, volume 12800, page 1280021. International Society for Optics and Photonics, SPIE, 2023.

26 Michel Raynal. A short introduction to failure detectors for asynchronous distributed systems. *SIGACT News*, 36(1):53–70, March 2005.

27 John R. Rice. The algorithm selection problem. *Adv. Comput.*, 15:65–118, 1976. doi:10.1016/S0065-2458(08)60520-3.

28 Davide Rovelli, Pavel Chuprikov, Philipp Berdesinski, Ali Pahlevan, Patrick Jahnke, and Patrick Eugster. Fide: Reliable and fast crash failure detection to boost datacenter coordination. In *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7-9, 2025*, pages 765–788. USENIX Association, 2025.

29 Laine E. Rumreich and Paolo A. G. Sivilotti. An eventually perfect failure detector on ADD channels using clustering. In *Networked Systems - 10th International Conference, NETYS 2022, Virtual Event, May 17-19, 2022, Proceedings*, volume 13464 of *Lecture Notes in Computer Science*, pages 149–166. Springer, 2022. doi:10.1007/978-3-031-17436-0_11.

30 Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, and Theo Ungerer. A new adaptive accrual failure detector for dependable distributed systems. In *SAC*, pages 551–555. ACM, 2007.

31 Pierre Sens, Luciana Arantes, Anubis Graciela de Moraes Rossetto, and Olivier Marin. Stab-FD: A cooperative and adaptive failure detector for wide area networks. *J. Parallel Distributed Comput.*, 186:104803, 2024.

32 Gabriela Stein, Luiz Antonio Rodrigues, Elias Procópio Duarte Jr., and Luciana Arantes. Diamond-p-vcube: An eventually perfect hierarchical failure detector for asynchronous distributed systems. In *12th Latin-American Symposium on Dependable and Secure Computing, LADC 2023, La Paz, Bolivia, October 16-18, 2023*, pages 40–49. ACM, 2023. doi:10.1145/3615366.3615420.

33 Lalith Suresh, Dahlia Malkhi, Parikshit Gopalan, Ivan Porto Carreiro, and Zeeshan Lokhandwala. Stable and consistent membership at scale with rapid. In *Proceedings of the 2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, pages 387–400. USENIX Association, 2018.

34 Alejandro Z. Tomsic, Pierre Sens, João Garcia, Luciana Arantes, and Julien Sopena. 2W-FD: A failure detector algorithm with QoS. In *IPDPS*, pages 885–893. IEEE Computer Society, 2015.

35 Karla Vargas, Sergio Rajsbaum, and Michel Raynal. An eventually perfect failure detector for networks of arbitrary topology connected with ADD channels using time-to-live values. *Parallel Process. Lett.*, 30(2):2050006:1–2050006:23, 2020. doi:10.1142/S0129626420500061.

36 Naixue Xiong and Xavier Defago. ED FD: Improving the Φ accrual failure detector, 04 2007. URL: https://hdl.handle.net/10119/4799.

37 Yuncheng Yao. MLFD: The implementation and performance evaluation of an LSTM-based, SmartNIC-offloadable failure detector. *SIGMETRICS Perform. Eval. Rev.*, 52(4):25–27, March 2025.

A Detector Detail Tables

Table 6 complements Figure 1 by summarizing the operational design dimensions used later to select evaluation modes and detector configurations. Table 7 complements Section 4 by detailing the configurations used in our evaluation for the detector implementations.

Table 6 Summary of the systematization of the studied failure detectors.

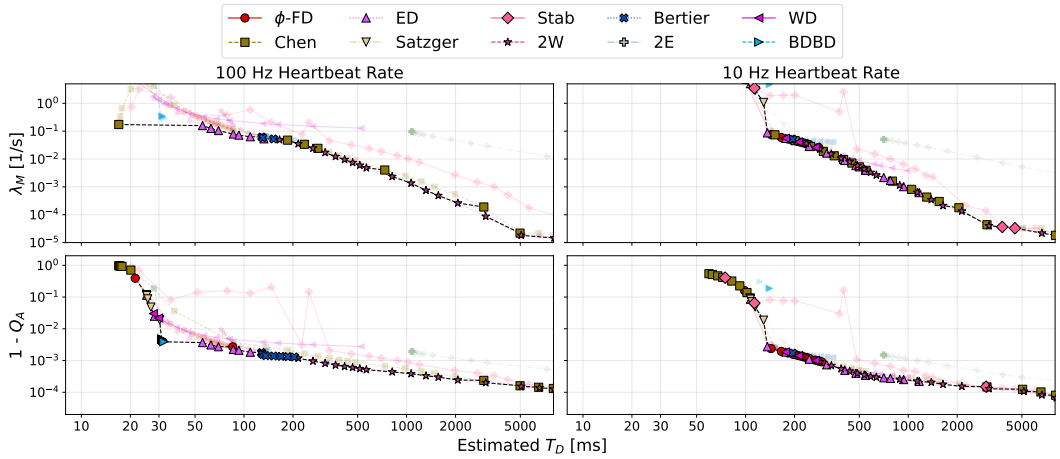
Algorithm	Design dimension	Detection/ output	Main refinement
Chen [7]	HB/timeout	Push, binary	QoS metrics; expected arrival; constant margin
Bertier [1]	HB/timeout	Push, binary	Jacobson-style dynamic margin
ϕ-FD [16]	Accrual	Push, accrual	Continuous suspicion level
ED [36]	Accrual	Push, accrual	Exponential inter-arrival model
Satzger [30]	Accrual	Push, accrual	Histogram/CDF suspicion estimate
Adaptare [9]	Probe with QoS	Pull, binary	Dependability-oriented parameter adaptation
BDBD [2]	Self-stabilizing	Push, binary	Bounded counters; no timestamps
B-AFD [10]	Probe with retransmission	Pull, binary	Retransmission-based QoS optimization
S-AFD [23]	Probe with cooperation	Pull, binary	Negative message sharing
2W [34]	HB/timeout	Push, binary	Short and long windows
WD [24]	Accrual	Push, accrual	Weibull inter-arrival model
2E [22]	HB/timeout	Push, binary	last delay estimate; smoothed margin
Stab [31]	HB/timeout	Push, binary	Stability-calibrated margin
Stab^C [31]	HB/timeout with cooperation	Push, binary	Stability-calibrated margin; cooperative views

Table 7 Parameter configurations used in the evaluation. We vary parameters that primarily affect detection time across various ranges. Notably, Adaptare and S-AFD do not expose an explicit parameter for controlling the message rate. We therefore select different configurations that yield either 10 Hz or 100 Hz probing behavior.

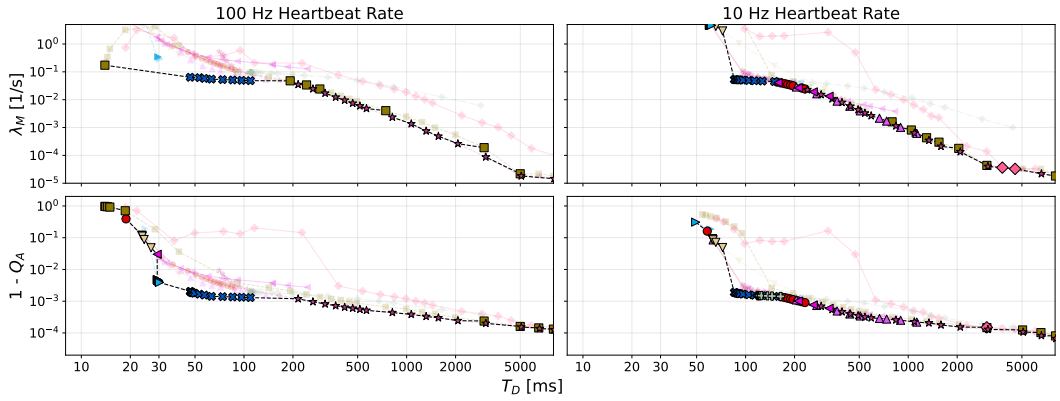
Algorithm	Parameter	No. of config.	Range of values
Chen	safety margin	27	7 to 12 000 ms
Bertier	Jacobson-style variance multiplier	13	0.1 to 30
ϕ-FD	suspicion threshold	14	0.1 to 16
ED	suspicion threshold	12	0.6 to 0.999 99
Satzger	suspicion threshold	12	0.6 to 0.999 99
BDBD	safety margin coefficient	11	0 to 5
2W	safety margin	27	7 to 12 000 ms
WD	suspicion threshold	12	0.6 to 0.999 99
2E	safety margin coefficient β	26	0.01 to 5000
Stab	initial safety margin	27	7 to 12 000 ms
Adaptare	mistake-recurrence target T_{MR}	12	1 to 5000 s
	coverage target (for 10 Hz)	12	0.9 to 0.999 98
	coverage target (for 100 Hz)	12	0.99 to 0.999 998
S-AFD	retry count per round rs	12	3 to 100 tries
	round duration Δ_s (for 10 Hz)	12	0.3 to 10 s
	round duration Δ_s (for 100 Hz)	12	0.03 to 1 s
Stab^C	initial safety margin	12	7 to 3000 ms

B Supplementary Results for Heartbeat Rate and Host Load

This appendix reports supplementary plots and tables for the Section 5. These plots are provided for completeness and reproducibility, and they do not change the interpretation of the main results or the detector-selection guidance. Figure 5 gives the full heartbeat-rate comparison, Figure 6 gives the full host-load comparison, Table 8 presents all detection-time thresholds used to check the robustness of the reported trends, and Table 3 provides the heartbeat inter-arrival statistics for the traces collected under different host loads.

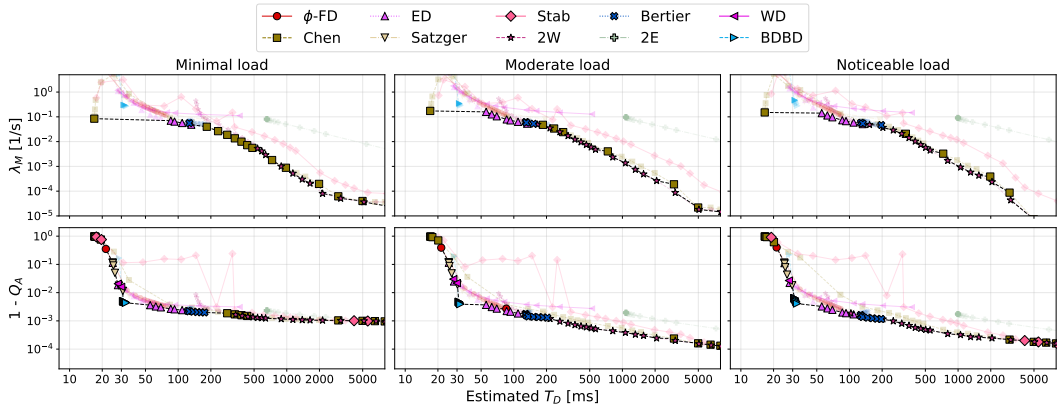


(a) Results under the NC environment.

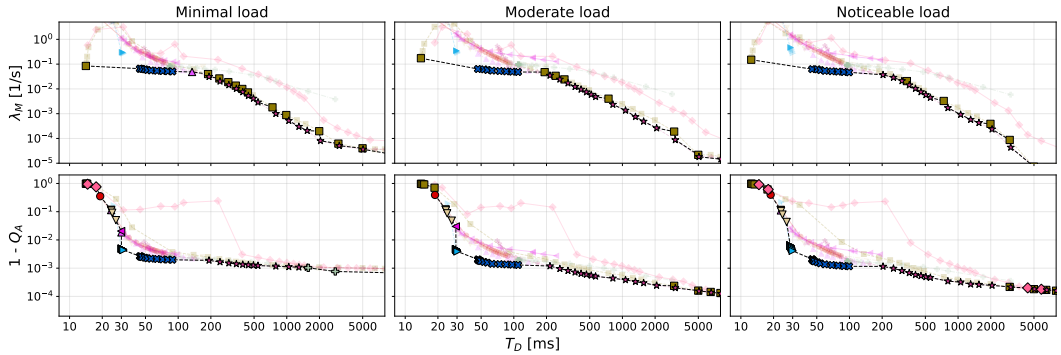


(b) Results under the CI environment.

Figure 5 Experiment 1: QoS metrics (λ_M and $1 - Q_A$) for offline detectors at 100 Hz and 10 Hz under (a) NC and (b) CI environments. All axes use log scale.



(a) Results under the NC environment.



(b) Results under the CI environment.

Figure 6 Experiment 2: QoS metrics (λ_M and $1 - Q_A$) for offline detectors under minimal, moderate, and noticeable host load in (a) NC and (b) CI environments. All axes use log scale.

Table 8 Lowest mistake rates λ_M and query error probabilities $1 - Q_A$ achieved under each host-load setting, crash environment, and detection-time threshold.

Load	Env.	Metric	Detection Time Threshold				
			$T_D \leq 50$	$T_D \leq 100$	$T_D \leq 200$	$T_D \leq 500$	$T_D \leq 2000$
min.	NC	λ_M	0.084	0.064	0.040	0.0056	$2.0 \cdot 10^{-4}$
min.	CI	λ_M	0.062	0.051	0.031	0.0041	$2.0 \cdot 10^{-4}$
mod.	NC	λ_M	0.173	0.072	0.048	0.0074	$4.9 \cdot 10^{-4}$
mod.	CI	λ_M	0.064	0.049	0.048	0.0074	$4.9 \cdot 10^{-4}$
not.	NC	λ_M	0.151	0.067	0.047	0.0075	$3.9 \cdot 10^{-4}$
not.	CI	λ_M	0.063	0.047	0.047	0.0075	$3.9 \cdot 10^{-4}$
min.	NC	$1 - Q_A$	0.0044	0.0026	0.0020	0.0014	0.0011
min.	CI	$1 - Q_A$	0.0024	0.0020	0.0019	0.0013	0.0010
mod.	NC	$1 - Q_A$	0.0039	0.0021	0.0013	$6.0 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
mod.	CI	$1 - Q_A$	0.0018	0.0013	0.0013	$6.0 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
not.	NC	$1 - Q_A$	0.0041	0.0019	0.0012	$5.4 \cdot 10^{-4}$	$2.6 \cdot 10^{-4}$
not.	CI	$1 - Q_A$	0.0017	0.0012	0.0012	$5.4 \cdot 10^{-4}$	$2.6 \cdot 10^{-4}$