Blackbird: A Scalable and Resource-Efficient Framework for Distributed Network Monitoring

Elisa Arnoldi*, Gabriele Merli†, Marco Abbadini‡, Michele Beretta‡,
Dario Facchinetti‡, Matthew Rossi‡, Stefano Paraboschi‡
Università degli Studi di Bergamo, Italy
Email: *e.arnoldi3@studenti.unibg.it, †g.merli2@studenti.unibg.it, ‡*name.surname*@unibg.it

***Abstract*—Modern large-scale network infrastructures require monitoring solutions that ensure operational resilience while imposing minimal overhead. However, existing centralized and static multi-agent approaches often suffer from single points of failure and can generate significant traffic congestion.**

This paper introduces Blackbird, a distributed framework designed to provide a continuous, real-time view of the entire network. Blackbird transforms selected network hosts into self-organizing agents capable of autonomously performing network scans. Within this architecture, a subset of agents act as *aggregators*, responsible for scheduling scans to mitigate burstiness and coordinating the activity of workers. *Workers*, in turn, perform measurements and disseminate the collected data across the system. Preliminary experimental evaluation shows that Blackbird achieves robust horizontal scalability and strong fault tolerance under node failures. Moreover, it maintains a low CPU and memory footprint, which is especially important for resource-constrained edge devices, while preventing network saturation and enabling real-time inspection from any participating node.

***Index Terms*—Distributed Network Monitoring, Scalability, Self-Organizing Systems, Fault Tolerance**

I. Introduction

Modern computer networks are increasingly large, dynamic, and heterogeneous. Except for fully controlled environments, they are typically characterized by a continuously evolving set of hosts. In this context, monitoring the state of the network, inspecting its evolution, and identifying potentially malicious hosts are fundamental security and management tasks.

Traditional monitoring tools [1]–[5] assess the state of the network from a single observation point by performing repeated scans. While this centralized approach simplifies deployment and management, it introduces significant limitations [6]–[8]. To maintain an accurate and up-to-date view, the monitoring node must perform frequent exhaustive scans, generating a continuous high volume of probing traffic and incurring substantial resource consumption in large-scale environments. Conversely, reducing the scanning frequency leads to stale and potentially inaccurate representations. Furthermore, centralized approaches inherently introduce a single point of failure, and may require complex configurations to operate across segmented networks, increasing operational overhead and the risk of misconfiguration.

To address these issues, prior work has explored distributed monitoring approaches [9]–[11]. In such systems, multiple agents deployed across the network perform localized scans and forward the collected data to a central entity for aggregation. This paradigm improves scalability and reduces network congestion by partitioning the scanning workload. However, it also introduces new challenges: each agent has only partial visibility of the network state, and the aggregation process reintroduces a logical single point of failure. When the aggregating node fails, the global network view becomes unavailable until a new aggregator is introduced and the system converges, which may lead to prolonged monitoring gaps.

Contribution. To overcome these limitations, we introduce *Blackbird*, a fully distributed monitoring framework that maintains a consistent and up-to-date view of the network without relying on centralized components. Blackbird organizes nodes into two dynamic roles: *aggregators*, which coordinate the system by scheduling network scans and synthesizing a global view from distributed measurements, and *workers*, which perform network scans, collect measurement data, and disseminate results across the system. Blackbird mitigates network congestion through a novel *wave-patterned* scanning strategy that distributes probing activity across network segments over time. At the same time, it achieves resilience to failures through an adaptive, self-organizing mechanism in which nodes dynamically assume roles based on a quality metric $Q_i(t)$. This enables seamless role reassignment in the presence of failures, improving availability and reducing data loss. We evaluate Blackbird on networks of varying sizes, showing that the system can construct an accurate view of a medium-sized network within seconds, while maintaining a low resource footprint.

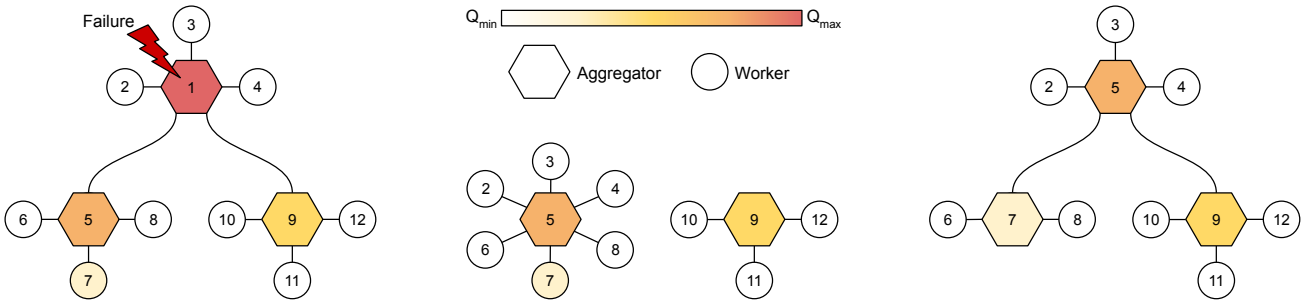
Availability. Blackbird's source code is available open source at https://github.com/Merluz/blackbirdSentinel.

II. Architecture

This section explains how Blackbird organizes agents into aggregators and workers (Section II-A), and how scansions are performed to avoid saturation (Section II-B).

A. Organization of Agents

Our approach requires the network administrator to initially distribute a set of agents into the networks to be scanned by manually installing Blackbird software on hosts, each characterized by possibly heterogeneous hardware resources. After this, the agents send broadcast messages to disclose their presence to other agents, and autonomously separate



(a) Node 1 fails

(b) Orphaned workers select the next highest-quality agent

(c) Node 5 has too many workers, so node 7 is promoted, rebuilding the tree

Fig. 1: Reorganization of the agent tree after a failure. In this example, each aggregator can have at most three workers.

into aggregators and workers. In principle, aggregators are hosts maximizing a quality value Q, which takes into account node-related metrics to estimate reliability and performance of nodes. This metrics weighs current node quality against past values of Q to account for historical reliability. Specifically, given a node i, this is defined as

$$Q_i(t) = \alpha q_i(t) + (1-\alpha)Q_i(t-1) \qquad \alpha \in [0,1]$$
$$q_i(t) = \sum_k w_k m_{i,k}(t) \qquad w_k, m_{i,k} \in \mathbb{R}$$

where $m_{i,k}$ is the aforementioned metric k for a node i, and w_k is a constant chosen to weigh and normalize the value of metric k. $Q_i(t)$ for $t < 0$ is assumed to be 0. A node metric $m_{i,k}$ depends on the specific environment where Blackbird is deployed, and can be chosen by the network administrator. Some examples include heartbeat regularity, connection stability, link quality, or physical system capabilities (e.g., available RAM or processing power), and Blackbird is flexible enough to accomodate any combination of choices.

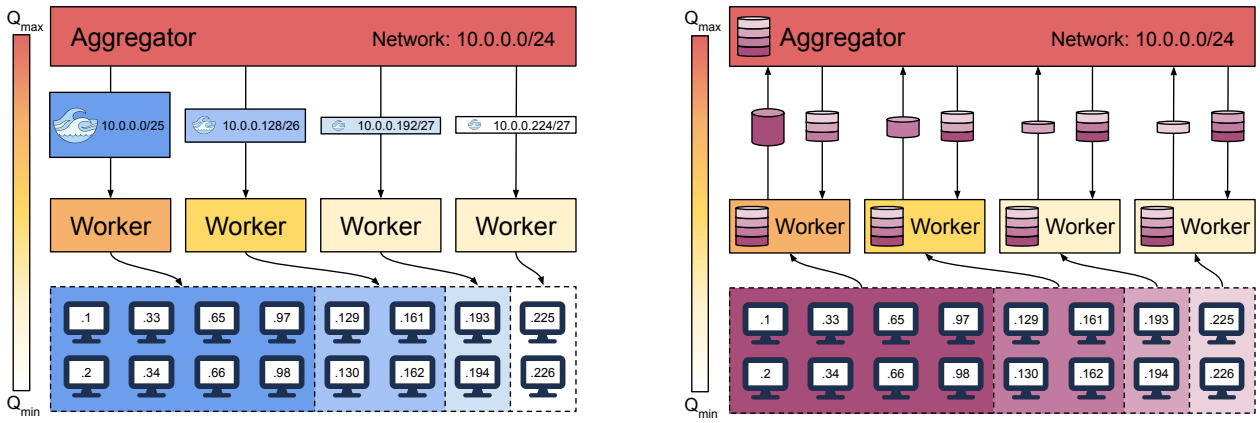
Then, each agent evaluates its own quality metric, and shares it to the other agents in the network. Among a group of agents who successfully established a connection, only the one with the highest Q is elected as aggregator. If a new node join the existing structure, it will be added either as a worker, or replace the existing aggregator based on its Q value. This continues until an aggregator has a certain number of workers W, at which point new joining nodes will become its *sub-aggregators*. Hence, agents will organize into an *agent tree*, where each aggregator manages up to W workers. This number can be defined at system configuration.

The role of aggregators is to gather the network topology information collected by its workers with periodic scans, and to combine it with the information received from other aggregators, ultimately building an accurate global network view. This information is shared with all connected agents, so that precise topology information is never centralized.

The separation between workers and aggregators permits to detect and react to network topology changes at runtime, as all agents periodically update each other. Moreover, this configuration is also fault resilient, as both the role of agents and the structure of the agent tree can change over time. For instance, when a fault occur to an aggregator, or when its Q value reflects a sustained loss in reliability, a new aggregator is elected. To do so, each worker maintains a *failover list*, i.e., a list of other local agents ordered by their Q value descending. Then, repairing the agent tree then involves, for the orphaned agents, simply connecting to the next available local agent and possibly re-organize the structure itself. A simple example is shown in Figure 1. Notably, since collected network data is always shared, no data is lost during failure recovery.

B. Network Scanning

After the agent tree has been generated, workers start recording network topology information with a sequence of scans, continuously sharing the collected data with the connected aggregator. To avoid flooding the network with a huge number of requests, the workers follow a *distributed wave-oriented* scanning strategy, which partitions the address space into manageable segments called *waves*. A *wave* represents a subset of the address space to analyze, i.e., a list of targets to process. By default, waves are equally sized, although the network administrator can tune this parameter setting custom range values. Each wave is not explored sequentially, but in parallel. This is done with the introduction of *chunks*, which represent the number of maximum parallel analyses for each wave. Aggregators determine the waves and chunks assigned to each worker based on their quality metric, with the idea that workers with more resources can scan larger (or more) waves and chunks, as highlighted in Figure 2a. For each target in a wave, workers first determine the reachability using the ICMP protocol, and only when the host is reachable an in-depth scansion is performed. This includes, for example, solving the host MAC address, checking open ports, performing network hardware identification, DNS resolution, and OS fingerprinting. After a target has been analyzed, the worker sends the collected data to the connected aggregator leveraging a persistent WebSocket connection (Figure 2b). Note that a worker may send network data before completely finishing its wave. This allows the aggregator to incrementally merge the information coming from different workers to create a network layout representation, which is in turn shared with connected workers and other aggregators.



(a) Wave distribution and network segment assignment

(b) Network monitoring and data sharing

Fig. 2: Wave-oriented network scanning. Waves are distributed by a top-level aggregator, and are sent to the workers based on their Q value. After receiving the waves, workers start monitoring network segments, sharing results with the aggregator.

This scanning strategy bears several advantages. First, the view of the network is created incrementally, which avoids saturating the network bandwidth, as waves and chunks permit to break down the work into smaller units. Second, it makes it easier to set a global packet rate limit, achieved either by reducing the overall simultaneous parallel chunks, or by waiting for a randomized delay before a wave scan is performed. Finally, this strategy permits to easily detect network changes, since waves can be repeated to identify new or removed hosts.

III. Experimental evaluation

In this section, we evaluate (i) Blackbird's network scanning efficiency, (ii) its ability to reconstruct the agent tree after failures, and (iii) resource footprint on individual nodes.

Network scan times. To evaluate scanning efficiency, we measured the time required to complete full network scans over progressively larger address spaces, ranging from a /24 to a /8 subnet, while varying the number of deployed agents. Each experiment was repeated ten times and results were averaged. Outcomes are shown in Figure 3. Blackbird exhibits high efficiency even for moderately large networks. For example, a /12 network (over 1 million addresses) can be scanned in approximately 30 s with 10 agents, and in under 10 s with 40 agents. In a small /24 network, increasing the number of agents from 10 to 50 yields only a marginal improvement (9.67%). In contrast, for a /8 network (over 16 million addresses), performance gains are significant: doubling the number of agents from 10 to 20 reduces scan time by 45.1%, while using 50 agents results in an overall improvement of 75.2%, reducing the total scan time to just over two minutes.

Agent reliability. We then evaluate the ability of Blackbird to recover from node failures. To this end, we simulate failures affecting a given percentage of nodes selected uniformly at random, and measure the time required to reconstruct the agent tree using the self-organization mechanism described in Section II-A. Results are reported in Figure 4. Despite some variability, reconstruction time shows little dependence on the

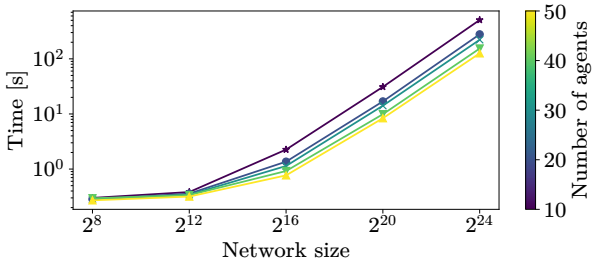


Fig. 3: Average network scan times.

number of failed agents and always remains below 150 ms on average, confirming Blackbird's ability to quickly recover from failures and restore a consistent global network view. Even in extreme scenarios where 90% of nodes fail, Blackbird reconstructs the cluster in about 86 ms on average. This is to be expected: when few agents fail, the agent tree is still mostly intact so reconstruction is trivial; while when most of the agents become unavailable, the short time to reconstruct the tree is due to the low number of remaining nodes.

Resource impact. Finally, we evaluate the resource footprint Blackbird by scanning differently sized networks with ten agents. Table I reports per-agent CPU and memory usage. The results show modest resource requirements: CPU utilization peaks at approximately 35%, while memory usage peeks at less than 103 MiB even in the largest configuration. Notably, resource consumption does not grow proportionally with network size, indicating efficient scalability. These characteristics also make the system suitable for deployment on resource-constrained devices.

IV. Related work

Several centralized network scanning tools have been proposed. For instance, Nmap [1] is a widely used open-source scanner executed from a single node, supporting host discovery and service detection. Unicornscan [4] provides an

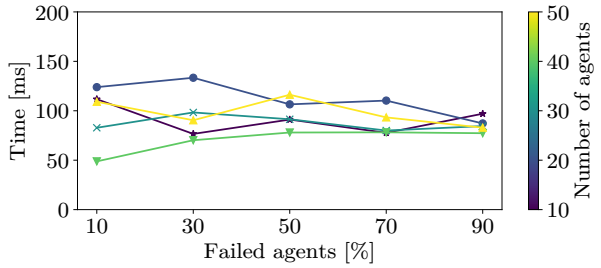


Fig. 4: Average time for the agent tree reconstruction.

asynchronous and stateless port scanner that re-implements the TCP/IP stack in user space, allowing probes to be sent and received without relying on the operating system, thus achieving higher performance than Nmap on large networks. Masscan [5] is an internet-scale TCP port scanner capable of transmitting up to 10 million packets per second, enabling scans of the entire IPv4 address space within minutes. ZMap [2] and its evolution [12] further prioritize high throughput over deep host inspection, introducing optimizations such as lock-free sharding to reach scan rates of up to 10 Gbps. XMap [13] extends this approach to IPv6 and multi-protocol scanning (e.g., TCP, UDP, ICMP), enabling fast discovery of network devices. While these tools significantly improve upon early approaches, they are primarily designed for internet-wide scanning across extremely large address spaces. As such, they are less suited for controlled or private environments, where minimizing overhead and avoiding network disruption are critical. Moreover, these tools often sacrifice reliability by omitting packet retransmissions and are typically optimized for single-port horizontal scans.

Research also considered distributed approaches. In [10], the authors propose a large-scale distributed port scanning system based on a TCP state machine and on task redundancy. Yet, the system still relies on a centralized component. VistaScan [9] is a more recent distributed framework that assigns scanning tasks based on node load using a visibility matrix. Although conceptually related to Blackbird, it is primarily designed for internet-wide port scanning and does not target continuous monitoring in controlled network environments.

A complementary line of work focuses on security-oriented monitoring. SMap [14] enables to study and filter ingress network traffic. LZR [15] detects misconfigured or potentially malicious services. Finally, emerging eBPF-based solutions (e.g., [16]–[18]) improve observability of network communications. In contrast, Blackbird targets trusted environments and focuses on maintaining an accurate and consistent view of the network state, rather than identifying adversarial behavior.

V. Conclusions and Future Work

We presented Blackbird, a distributed framework for network monitoring that provides an accurate global view without centralized components. By combining self-organizing agents, dynamic roles of aggregators and workers, and a wave-patterned scanning strategy, Blackbird achieves scalable and efficient monitoring. Experimental results show that the system scales with network size, recovers from failures in under 150 ms on average, and maintains a low resource footprint, making it suitable for resource-constrained environments. Future work will focus on extending the framework to more dynamic scenarios, including support for additional protocols and adaptive strategies for real-time anomaly detection.

TABLE I: Per-node resource impact of Blackbird.

Network size	CPU		RAM	
	Average	Peak	Average	Peak
2^8	0.34%	0.34%	2.40 MiB	2.40 MiB
2^{12}	1.90%	1.90%	6.97 MiB	6.97 MiB
2^{16}	13.27%	13.47%	33.63 MiB	34.56 MiB
2^{20}	32.22%	32.23%	48.49 MiB	83.01 MiB
2^{24}	34.91%	34.91%	89.60 MiB	102.91 MiB

Acknowledgments

This work was supported by MASE Mission Innovation 2.0 - D.M. 386/2023 under project EVFTEG grant No. F53C25001680001, and by PRIN project NEWTON (2022ZA8T22) funded by the EU - NextGenerationEU.

References

[1] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*, 2009.
[2] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *USENIX Security*, 2013.
[3] Z. Durumeric, D. Adrian, P. Stephens, E. Wustrow, and J. A. Halderman, "Ten Years of ZMap," in *IMC*, 2024.
[4] R. E. Lee, "Unicornscan," https://unicornscan.org/.
[5] R. Graham, "MASSCAN: Mass IP port scanner," https://github.com/robertdavidgraham/masscan.
[6] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "The Matter of Heartbleed," in *IMC*, 2014.
[7] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Professional, 1998.
[8] A. Pras, T. Drevers, R. van de Meent, and D. Pontz, "Comparing SNMP and Web Services for Network Management," *IEEE Communications Magazine*, 2005.
[9] L. Hu, F. Shi, Y. Shen, C. Xu, P. Xue, and B. Guo, "VistaScan: Optimizing Internet-Wide Scanning Through Visibility-Aware Distributed Task Allocation," *Sensors*, 2026.
[10] Z. Li, X. Yu, D. Wang, Y. Liu, H. Yin, and S. He, "Supereye: A distributed port scanning system," in *ICAIS*, 2019.
[11] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure," in *USENIX LADIS*, 2010.
[12] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, "Zippier ZMap: Internet-Wide Scanning at 10 Gbps," in *USENIX WOOT*, 2014.
[13] X. Li, Z. Xie, L. Sun, Y. Qiu, Z. Xu, and Z. Liu, "XMap: Fast Internet-wide IPv4 and IPv6 Network Scanner," arxiv.org/abs/2602.09333, 2026.
[14] T. Dai and H. Shulman, "SMap: Internet-wide Scanning for Spoofing," in *ACSAC*, 2021.
[15] L. Izhikevich, R. Teixeira, and Z. Durumeric, "LZR: Identifying Unexpected Internet Services," in *USENIX Security*, 2021.
[16] M. Abbadini, M. Beretta, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi, "Lightweight Cloud Application Sandboxing," in *IEEE CLOUDCOM*, 2023.
[17] M. Abbadini, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi, "NatiSand: Native Code Sandboxing for JavaScript Runtimes," in *RAID*, 2023.
[18] M. Rossi, M. Beretta, D. Facchinetti, and S. Paraboschi, "POSTER: Transparent Temporally-Specialized System Call Filters," in *ASIACCS*, 2025.