Peer-to-peer Architecture for Collaborative Intrusion and Malware Detection on a Large Scale

Mirco Marchetti, Michele Messori, and Michele Colajanni

Department of Information Engineering
University of Modena and Reggio Emilia
{mirco.marchetti, michele.messori, michele.colajanni}@unimore.it

Abstract. The complexity of modern network architectures and the epidemic diffusion of malware require collaborative approaches for defense. We present a novel distributed system where each component collaborates to the intrusion and malware detection and to the dissemination of the local analyses. The proposed architecture is based on a decentralized, peer-to-peer and sensor-agnostic design that addresses dependability and load unbalance issues affecting existing systems based on centralized and hierarchical schemes. Load balancing properties, ability to tolerate churn, self-organization capabilities and scalability are demonstrated through a prototype integrating different open source defensive software.

1 Introduction

Distributed and collaborative systems are emerging as the most valid solutions to face modern threats coming from multiple sources. To identify network intrusions and new malware as soon as possible, hierarchical architectures for intrusion detection have been proposed, such as [1]. They are able to gather information from a wide network space and allow early detection of emerging threats because they are based on multiple sensors placed in different network segments and on a hierarchical collaboration scheme. This approach allows administrators to deploy timely countermeasures because all the network segments hosting at least one sensor can be alerted about new threats as soon as they are detected in any part of the collaborative system.

The problems affecting existing collaborative solutions based on hierarchical or centralized architectures are well known: peer dependability issues, limited scalability and load unbalance. We present a distributed collaborative architecture that aims to address these main issues through a cooperative peer-to-peer scheme based on a Distributed Hash Table (DHT).

The peer-to-peer architecture proposed in this paper aims to capture and analyze real malware specimens and propose countermeasures instead of just recognizing that a malware is spreading. Moreover, it disseminates network activity reports on the basis of a behavioral analysis of the captured payload, thus being able to provide a description of the malware behavior. The communication

model is different from existing systems because it adopts a publish/subscribe scheme as an option for the distribution of the result of the analysis, while each alert is inserted into an ad-hoc message persistently stored in PAST [14]. This solution guarantees privacy and availability of the information. Other proposals based on peer-to-peer defensive schemes (e.g., [22, 23]) differ from this paper because their focus is on novel algorithms for anomaly detection that should be facilitated by cooperation. On the other hand, our focus is on the software architecture that is flexible enough to work with different algorithms.

Other peer-to-peer schemes (e.g., [24,25,28]) are used to disseminate information about malicious IP addresses through some publish/subscribe model. Our architecture uses a publish/subscribe scheme only for the communication of the analysis results, while events are persistently stored and can be retrieved successively. Other proposals have some peculiarity that is not addressed in this paper. For example, DOMINO [25] is an interesting architecture because its overlay network combines peer-to-peer and hierarchical components and uses Chord [26] to distribute alert information. The main goal of Worminator [27] is to guarantee a high level of privacy of the shared information. It extracts relevant information from alert streams and encodes it in Bloom Filters. This information forms the basis of a distributed watchlist and includes IP addresses and ports. The watchlist can be distributed through various mechanisms, ranging from a centralized trusted third party to a decentralized peer-to-peer overlay network. However, we should be aware that the main goal of these architectures is to compile an updated blacklist of the IP addresses at the origin of some attacks. On the other hand, this paper has a broader scope: our architecture manages IP addresses and other important information, such as binary code of malware, signature of IDS and malware behavior. Moreover, our architecture is sensor agnostic, and is able to support heterogeneous algorithms and techniques for intrusion detection and malware analysis. For these reasons, it differs from Indra [28] that is a distributed intrusion detection architecture that relies on custom sensors.

This paper is organized as follows. Section 2 describes the design of the proposed architecture. Section 3 highlights its main benefits with respect to hierarchical and centralized architectures. Section 4 details the main features of the prototype that is based on open source software. Section 5 reports the experimental results achieved through the prototype. Scalability, load balancing, robustness and self-organization properties at a larger scale with thousands of collaborative nodes are demonstrated through simulation. Section 6 outlines main conclusions and future work.

2 Architecture Design

The main goal of this paper is to design a distributed architecture where each component collaborates to the intrusion and malware detection and to the dissemination of the local analyses including:

- malware behavior,
- malware diffusion,

– network-based attacks,
– diffusion of intrusions,
– identification of suspicious IP addresses,
– identification of the servers from which the malware is downloaded.

The novel architecture should address the main issues of hierarchical collaborative schemes in order to guarantee high scalability, fault tolerance, dependability and self-organization.

To accomplish these goals we propose a flat distributed architecture composed by several cooperating nodes that communicate through a DHT overlay network. An overview of this architecture is shown in Figure 1. Each node, called *collaborative alert agregator*, accomplishes the same high level functions: generation of local security alerts, forwarding of relevant alerts to the other collaborative nodes, analysis of received events and communications of the analysis results. All the communications among the collaborative alert agregators are carried out through a peer-to-peer overlay providing a fully connected mesh network. This solution does not require centralized coordination nor supernodes.

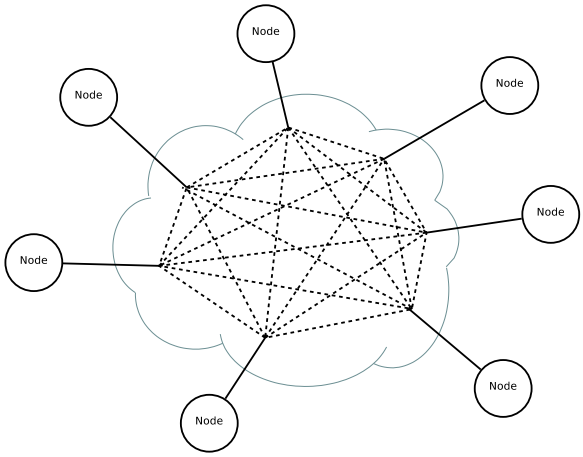

Fig. 1. Node connections to the DHT

The design of a collaborative alert agregator is represented in Figure 2. It is possible to identify three layers: the *sensor layer*, the *local aggregation layer*, and the *collaboration layer*. Each layer is described in the following sections.

2.1 Sensor Layer

The sensor layer provides a node with intrusion detection alerts and malware samples. It consists of one or multiple types of sensors. For example, the current version relies on four classes of sensors.

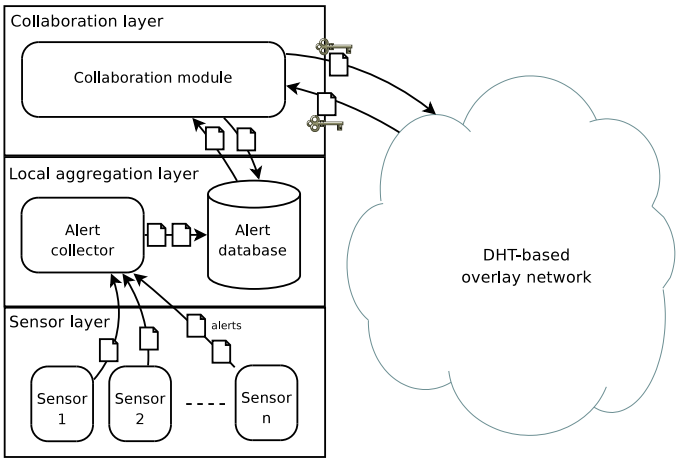

Fig. 2. Design of a collaborative alert aggregator

- **Host IDS**. Intrusion detection system monitoring activities on the same host in which they are deployed.
- **Network IDS**. Sensors placed in strategic positions in order to capture all network traffic and to analyze each packet looking for malicious content. They can be implemented through custom hardware appliances or installed on a general purpose computer. When illicit activities are detected, they generate an alert containing information on malicious network packets (such as a TCP/IP header and a signature identifier) and a description of the attack.
- **Honeypot**. These tools are able to collect malware and to trace malicious activities. They consist of server processes hosted on a vulnerable computer connected to a network. As an honeypot does not provide any useful service, any attempt of reaching it and logging into its services can be considered an attack.
- **Sensor manager**. This class of sensors represents a component of a multi-tier, hierarchical NIDS architecture. A sensor manager forwards information to other managers belonging to upper tiers. It can also aggregate and filter data, thus reducing the number of forwarded alerts and decreasing the network traffic and the computational load of the other components of the hierarchical architecture.

It is not necessary to install the sensors on the same physical machine hosting the local aggregation and the collaboration layers. Sensors can interface with alert collectors installed on remote hosts.

2.2 Local Aggregation Layer

The local aggregation layer is responsible for collecting, filtering and aggregating all the heterogeneous alerts received from the sensors of the lower layer. While it

is possible for the *alert collector* to execute arbitrarily complex aggregation and correlation algorithms, its fundamental task is to pre-process all the received alerts that may be syntactically and semantically heterogeneous. All the alerts are classified and stored in the local *alert database* that is used by the upper layer as the only sensor-independent interface to store and retrieve heterogeneous events.

2.3 Collaboration Layer

The collaboration layer is the only component connected to the collaboration overlay network that is based on DHT. Being part of the overlay network, we can assign a unique node identifier $nodeID$ to each *collaboration module*. The collaboration module has three main purposes.

- It is responsible for retrieving new events that have been stored in the local alert database. These events are submitted to the DHT-based overlay network through a key ($messageId$) that is computed over a significant field of the event. As an example, the key used to submit a malware specimen can be computed by applying a hash function to the malware, while a NIDS alert can be submitted to the overlay network twice, using as keys the signature ID and the IP address from which the illicit network packet originated. The strategy used to determine which fields are involved in key computation can be configured to fulfill specific analysis goals.
- It receives messages submitted by other collaboration modules that are connected to the same overlay network. Each collaboration module is responsible for a portion of the hash space (determined by the $nodeID$ and by the implementation details of the DHT algorithm), and receives all the messages whose $messageId$ fall within that hash space. This design choice allows each collaboration module to receive all the messages that are relevant to one scenario. As an example, this approach allows a single collaboration module to receive all the events caused by the same source, thus achieving an effective network-based and distributed alert aggregation and correlation scheme.
- Each collaboration module is responsible for the dissemination of the analysis results to the other nodes connected to the overlay network. This guarantees the timely disseminations of network activity reports to all the collaborative nodes without any centralized bottleneck.

There are two ways to retrieve new alerts from the database: the collaboration layer reads the collected data at regular interval (pull mode) or it is driven by external calls (push mode). In the implementation presented in Section 4 we use the pull mode.

2.4 Event Processing

The collaboration layer gets new events from the database and processes them sequentially. For each event, it sends a number of messages depending on the

type of event and analysis goals. In the current version of the architecture, each event retrieved from the database may have up to four interesting fields: the malware's binary code, the IP address of the server from which the malware has been downloaded, the IDS signature ID and the IP address of the attacker. For each of these fields a message is created using the hash of its value as $messageId$. This message is received by the node with the $nodeID$ closer to the $messageId$. Before the insertion of a new message, the collaboration module monitors the pre-existence of its $messageId$ within the DHT. If there is not that value, the message is inserted, and the sender node signals the arrival of a new message to the receiver node. Otherwise, if the $messageId$ already exists, the sender node contacts the receiver node and it informs it about the generation of a new event with the same $messageId$.

The receiver node behaves differently depending on the message type. If the message contains a new malware, the receiver node takes care of its behavioral analysis by relying on a local or remote sandbox. In all the other instances, the node executes some anomaly detection strategies that are based on the frequency of received events. It is important to observe that in this paper we do not focus on specific event analysis algorithms, but on the architecture which permits to collaborate and to share information. In particular, the proposed architecture is algorithm agnostic and flexible enough to adopt several different analysis strategies. Finally, the event analysis results are disseminated to all the interested collaborative alert aggregators following a publish/subscribe paradigm [16].

3 Peer-to-peer vs. Hierarchical Architecture

3.1 Fault Tolerance

The completely distributed nature of the proposed architecture is inherently fault tolerant, and lacks single points of failure that are typical of hierarchical and centralized architectures, where alert aggregation, correlation and analysis functions are aggregated in the root node of the architecture [1]. This node represents a single point of failure and when it is unreachable the hierarchical architecture is unable to complete any collaborative task. The effectiveness of hierarchical architectures can be impaired even by failures of the nodes belonging to intermediate layers of the tree. As an example, a failure of one of the tier-1 nodes causes the isolation of the complete sub-tree having the faulty node as its root.

On the other hand, the proposed architecture leverages a completely distributed, network-driven aggregation and correlation technique. Each node is responsible for aggregating and correlating only a small subset of alerts and malware samples. If a node becomes unreachable, only the messages that would have been handled by the faulty node are lost, while all the other nodes are not influenced by the failure. Moreover, depending on the implementation of DHT routing of the overlay network, the collaborative nodes can detect the failure of a peer, and autonomously modify their local overlay routing tables accordingly.

Hence, the proposed architecture is able to autonomously reorganize itself and restoring its efficiency with no human intervention.

Message replication schemes can also be used to reduce the (minimal and transitory) message losses due to the failure of a collaborative node. In the current version, it is possible to set a *replication constant* k denoting that, for each message, k copies are created and maintained by the DTH overlay. One message is sent to the node whose unique identifier $nodeID$ is responsible for the message key. The other $k-1$ messages are sent to the $k-1$ nearest neighbors, thus guaranteeing full reliability for up to $k-1$ failures, because the network would maintain constant the number of replicas through periodic inspections (experimental evaluation of message loss probability for higher number of concurrent faults are presented in Section 5). By tuning the value of k, it is possible to achieve the desired trade-off between overhead and fault tolerance.

3.2 Load Balancing

Hierarchical architectures, such as [1], concentrate malware analysis and alert correlation tasks on the root node, so that they can avoid replicated analyses. As a bad consequence, the computational load on the root is significantly higher than the load of the intermediate nodes, to the extent that much more powerful hardware is necessary to host the root services.

Another advantage of the proposed DHT-based distributed architecture is represented by its intrinsic load balancing properties. Let us consider a scenario in which a network participating to a collaborative hierarchical architecture is targeted by an attacker, while the other participating networks are not. In a similar situation, the load related to alert management and correlation is unevenly distributed because only the nodes related to the attacked network are involved in alert management. Hence, an attacker could easily overload the path connecting the attacked networks to the hierarchy root by attacking few selected networks connected to the same higher-level node.

Uneven load distribution and overload risks are mitigated by the proposed distributed alert aggregation and correlation scheme. As we avoid one centralized aggregator, then there is no single path through which all the alerts generated by a sensor (or a set of sensors in the same network) are transmitted. Alerts gathered by one node in an attacked network are routed to multiple nodes, based on the messageId characterizing each alert. Even if one network (leaf) is heavily attacked, this scenario is well managed and the load is automatically distributed among many nodes through different branches.

3.3 Scalability

Hierarchical architectures are based on a multi-tier management infrastructure connecting the lowest layer alert managers (the leaves of the management tree) to the root manager. Each manager node in the hierarchical architecture is able to aggregate alerts generated by a finite number n of lower-layer sensors or managers on the basis of computational complexity of alert management operations

and on bandwidth constraints. Hence, a hierarchical architecture can be modeled as an n-ary tree, whose number of intermediate elements grows logarithmically with the number of the leaves.

On the other hand, in the proposed architecture, all the alert management operations are distributed among the leaves and there is no need for a separate management infrastructure. This is a huge advantage in terms of scalability and management, because it is not necessary to reconfigure the architecture hierarchy, possibly by adding new layers to the management tree, whenever the number of leaves increases.

3.4 Number of Stored Copies

Another advantage of the peer-to-peer architecture is represented by the smaller number of copies of individual alerts and malware specimens. In a hierarchical architecture, a copy of each different alert and malware specimen is maintained in each node of the management tree by which the alert has been received. Let us consider a tree of managers having order n, l leaves and height $h = \log_n(l)$. If c represents the number of copies of each alert stored by the nodes belonging to the hierarchical architecture, then we have:

$$h \leq c \leq \sum_{i=0}^{h-1} n^i$$

This means that the number of copies c is comprised between the number of manager nodes in the path between the leaf generating the alert and the root of the tree (h) and the total number of manager nodes when the same alert has been issued by all the leaves in the tree ($\sum_{i=0}^{h-1} n^i$). As h grows proportionally to the logarithm of the number of leaves, then the number of copies of each alert (and malware specimen) that a hierarchical architecture needs to maintain grows logarithmically with the number of leaves.

On the other hand, in the peer-to-peer architecture the number of copies of each different alert is determined by the replication factor k, which is a configurable parameter independent of the number of nodes connected to the overlay.

A comparison between the number of stored copies is presented in Table 1. The first row of this table represents the number of copies stored in the peer-to-peer architecture having a replication factor $k = 5$. The number of copies does not depend on any other parameter. The other rows contain the number of copies stored in a hierarchical architecture characterized by a different number of nodes and order. As an example, the second row shows that the number of copies of the same alert in a hierarchical architecture with $l = 1000$ nodes and order $n = 10$ is between 3 and 111, depending on how many leaves issue the same alert.

Architecture	Minimum	Maximum
DHT overlay, $k = 5$	5	5
Hierarchical, $l = 1000$, $n = 10$	3	111
Hierarchical, $l = 10000$, $n = 10$	4	1111
Hierarchical, $l = 100000$, $n = 10$	5	11111
Hierarchical, $l = 8000$, $n = 20$	3	421
Hierarchical, $l = 160000$, $n = 20$	4	8421

Table 1. Number of store copies stored in the peer-to-peer and hierarchical architectures

4 Prototype

The viability of the proposed architecture has been demonstrated through a prototype. In compliance with the architecture description in Section 2, each collaborative alert aggregator consists of different software modules that can be divided in three classes. The first two classes include typical network defense items. The third class includes communication software. The entire prototype is based on open source software.

The current implementation of the collaborative alert aggregator can rely upon heterogeneous sensors, thus being able to detect a wide range of threats. In particular, we used Snort [2] (standard de-facto for signature based network intrusion detection) as a NIDS sensor, and Nepenthes [3] as a low-interaction honeypot.

The alert collector is implemented through the Prelude software [4,5]. All the communications between the Prelude manager and the sensors is based on the Intrusion Detection Message Exchange Format (IDMEF) [6], which is an IETF standard for the generation and management of messages related to security incidents.

The alert collector is configured to store all the collected alerts and malware samples in the local alert database, implemented with MySQL [7].

The collaboration layer is implemented in Java and guarantees a platform independent application. The DHT-based overlay network relies on the FreePastry libraries, a Java implementation of Pastry network [9–13]. These libraries guarantee a useful emulation environment, and implementation of two applications based on Pastry: PAST [14,15], that is the persistent peer-to-peer storage utility used to store information, and Scribe [16,17], used for multicast communications.

The collaboration module can be configured by editing an XML [18] file. The interface with the alert database is implemented through JDBC drivers [19], thus guaranteeing a high interoperability with the most common DBMS.

Events retrieved from the database are classified according to their type, and managed by different classes. The current version of the prototype implements four classes managing four heterogeneous types of event: malware samples, IP addresses related to the server from which the malware is downloaded, IP addresses related to hostile activity and signatureId of alerts generated by a NIDS

sensor. The collaboration module is modular and its functions can be easily extended by adding new classes for the management of other types of event.

After a message has been received, the collaboration module is responsible for its storage, its analysis (possibly leveraging external services, such as Norman sandbox [20] and CWSandbox [21]), and communication of the analysis results to the other nodes. Each node can subscribe to specific areas of interests, thus receiving only a subset of analysis results. In the current implementation, alert storage is handled by PAST, while multicast dissemination of the analysis results relies on Scribe. Finally, a custom application based on FreePastry provides a one-to-one communication service.

PAST is a persistent storage utility distributed peer-to-peer network. It is based on a Pastry layer for routing messages and for network management. We adopt it because it is able to balance the storage space between all the nodes and guarantees high fault tolerance. Each PAST node, identified by an ID of 128 bits ($nodeID$), acts both as a storage repository and as a client access. Any file inserted into PAST uses a 160-bit key, namely $fileId$ ($messageId$ in our case). When a new file is inserted into PAST, Pastry directs the file to the nodes whose $nodeID$ is numerically closer to the 128 most significant bits of $messageId$, each of which stores a copy of the file. The number of involved nodes depends on the replica factor chosen for availability and persistence requirements.

5 Validation and Performance Evaluation

Viability and performance of the proposed architecture have been demonstrated through extensive experiments and simulations. Small scale experiments, through the prototype comprising few tens of nodes, have been carried out by executing several instances of the collaboration module in few hosts and by binding each instance on different port numbers. Tests for large number of nodes include the network emulation environment provided by the FreePastry libraries. This solution allows us to launch up to one thousand nodes on each Java Virtual Machine.

Very large scale simulation involving up to ten thousand nodes are carried out through an ad-hoc simulator. It considers the high-level behavior of the hierarchical and DHT-based architectures and omits the low level details related to the transmission of messages over the network and their storage within the local alert database. Although simplified, the simulator uses the same routing schemes of the prototype, and it has been validated by executing the same (reduced scale) experiments on the prototype and on the simulators and by verifying the complete agreement of the results.

5.1 Dependability and Fault Tolerance

The completely distributed nature of the proposed architecture is inherently fault tolerant, and lacks single points of failure. While the failure of few nodes does not impair the architecture dependability, it is possible that an alert message

sent to a faulty node can be lost. To minimize the chances of loosing alerts, the proposed architecture relies on the message replication scheme described in Section 3. It is possible to configure a replication factor k, so that each message is sent to the k collaborative alert aggregators whose $nodeID$ is nearest to the message key.

The ability of the proposed architecture to sustain faults and node churn is demonstrated through several simulations. In each run we simulate an overlay network consisting of a variable number of collaborative nodes (from 1000 to 10000), and we randomly generate a set of messages. Once guaranteed that each node is responsible for at least one message, we simulate the concurrent failure of a percentage of collaborative alert aggregators, ranging from 1% to 10% of the nodes. Then, we wait for PAST to run a scheduled update, thus restoring k copies of each message not lost due to the concurrent node failures, and we check whether all messages created at the beginning of the simulation are still available. The results are presented in Tables 2 and 3.

In Table 2 we compare networks of different sizes (between 1000 and 5000) where nodes use a replica factor $k = 5$ (5 copies of each message). The number of faulty nodes is denoted as a percentage of the total number of nodes. For each failure rate and for each network size, we run the simulation 100,000 times. The message loss rate is expressed as a percentage of the 100,000 runs in which *at least* one message has been lost. As an example, the cell in the fourth row and second column in Table 2 denotes that a network of 1000 collaborative nodes with 40 concurrent failures (4% for the number of nodes) lost at least one message in only 0.01% of the 100,000 tests.

Concurrent fault rate (%)	1000 nodes	2000 nodes	5000 nodes
1	0	0	0
2	0	0	0
3	0	0	0,01
4	0,01	0,02	0,04
5	0,02	0,04	0,12
6	0,06	0,12	0,24
7	0,11	0,22	0,6
8	0,19	0,42	1,04
9	0,35	0,76	1,8
10	0,58	1,16	2,99

Table 2. Message loss probability for $k = 5$ and for different numbers of nodes and faults

In Table 3 we report the results about the influence of the replica factor k on the probability of losing a message. In these simulations we use a network size of 10,000 collaborative nodes, and we vary both the concurrent failure rate (expressed as percentage of the number of nodes) and the replica factor (with values of k ranging from 4 to 6). As in the previous set of experiments, for each

combination of fault rate and replica factor we run 100,000 simulations. The packet loss probability is expressed as the percentage of simulations in which at least one message has been lost.

Concurrent fault rate (%)	k=4	k=5	k=6
1	0,009	0	0
2	0,16	0,003	0
3	0,735	0,019	0,001
4	2,117	0,075	0,002
5	5,022	0,219	0,015
6	9,732	0,542	0,037
7	16,64	1,186	0,081
8	25,859	2,172	0,159
9	36,682	3,774	0,315
10	48,685	5,904	0,529

Table 3. Message loss probability for a network of 10,000 nodes and for different k

These experiments demonstrate that by using appropriate values of the replica factor k, it is possible to achieve negligible message loss probability even for large networks and (unrealistic) concurrent failures of hundreds of geographically distributed and independent nodes.

5.2 Load Balancing and Scalability

In this section we compare the load of the proposed peer-to-peer architecture against that of the lowest layer of alert manager in the hierarchical architecture presented in [1].

Experiments are carried out by simulating a network of 5000 collaborative nodes, and a hierarchical architecture with the same number of leaves. Each intermediate manager node of the hierarchical network is connected to a random number of elements in the lower level, uniformly chosen between 5 and 20. The resulting tree has 4 layers and 420 low-level managers, directly connected to the leaf sensors. Figures 3 and 4 compares the load distribution among the 5000 collaborative alert aggregators in the DHT-based overlay and the load distribution among the 420 manager nodes of the hierarchical architecture.

Figure 3 represents the best-case scenario for the hierarchical architecture, in which all the 500,000 messages (100 messages for each leaf, on average) are uniformly distributed among the leaves. The uniform distribution among all the leaves simulates an unrealistic scenario in which network attacks are uniformly distributed among all the sensors that generate alerts at the same rate. The two lines of Figure 3 represent the cumulative distribution function (CDF) of the number of messages that each node in the collaborative architecture (line

P2P) and each low-level manager in the hierarchical architecture (line *Hierarchical*) has to manage. The X axis represents the ratio between the number of messages handled by a node and the expected average value (that is, $500,000/5000 = 100$ messages per collaborative node in the collaborative architecture, and $500,000/420 = 1190.48$ messages per manager in the hierarchical architecture). The vertical line represents the ideal load distribution, in which all the nodes handle a number of messages that is equal to the expected average (hence the ratio between the handled messages and the expected ratio is 1).

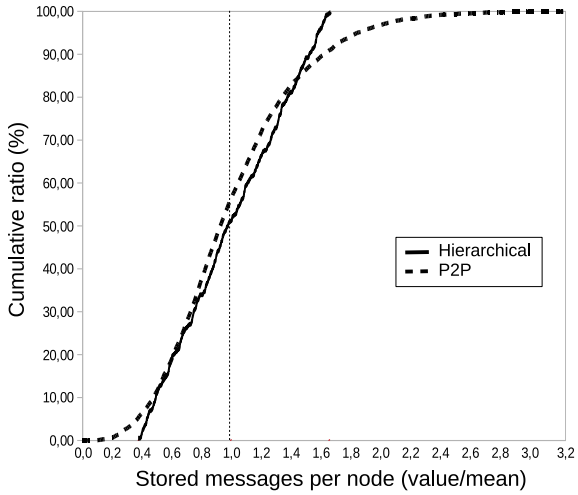

Fig. 3. Load balancing with random inserts

As shown in Figure 3, characterized by a uniform distribution of alerts among sensors, the load of the hierarchical architecture is better distributed than that of the collaborative overlay. Indeed, all the nodes in the hierarchical architecture handle a number of messages between 40% and 150% of the expected average. However, even in the best-case scenario for the hierarchical architecture, the load distribution of the two collaborative architectures is comparable. Indeed, the two distributions behave similarly for about 70% of the nodes; 20% of the nodes in the peer-to-peer architecture are more loaded than the most loaded nodes of the hierarchical architecture, but the highest load is still manageable and limited to 3.2 times than the expected load.

In Figure 4 we report the results of a more realistic scenario, in which 500,000 messages are not generated uniformly by all the collaborative alert aggregators, but they follow a Zipf distribution[1] (known to be a realistic representation of

[1] The used formula is $f(P_i) = \frac{c}{i}$, where i is the rank, P_i indicates the event which occupies the i-th rank (the i-th most frequent), $f(P_i)$ is the number of times (frequency) that the P_i event occurs, c is a constant equivalent to P_i

traffic distribution and load of nodes in the Internet). The two cumulative distributions in this figure show the benefits of the hash-based event distribution algorithm implemented by the DHT-based overlay network. The node managing an alert is determined by the alert content and not by the network generating it, hence the load distribution of the proposed architecture in this scenario is identical to that presented in Figure 3. On the other hand, the load distribution in the hierarchical architectures is highly unbalanced. For example, the most loaded manager handles a number of messages equal to 103 times the expected average.

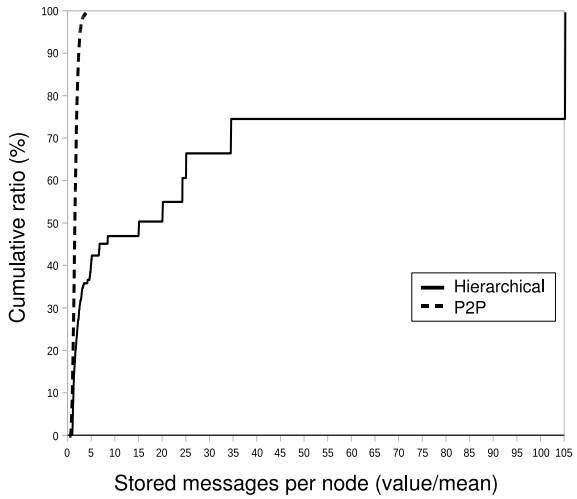

Fig. 4. Load balancing with Zipf distribution based inserts

We consider now an attack scenario in which all the events are generated by one sensor (or by several sensors connected to the same low-level manager). This is the worst case for a hierarchical architecture, because all the alerts must be handled by the same manager node. The results are impressively poor: one manager has to sustain a load 420 higher than that related to the expected average. On the other hand, the proposed architecture preserves the same load distribution of the other two scenarios. This shows the robustness of the peer-to-peer architecture with respect to any attack scenario.

6 Conclusion

In this paper we propose an innovative architecture for collaborative and distributed intrusion detection, malware analysis and alert dissemination.

With respect to previous work in the same field, our architecture represents a more flexible cooperation infrastructure that is able to manage heterogeneous

information, ranging from IP addresses to the complete binary code of malware specimens and IDS alerts. Moreover, our architecture is not focused on a specific analysis algorithm and can leverage heterogeneous analysis engines. Finally, a publish/subscribe scheme is used for efficient and timely dissemination of relevant analysis results to all the collaborative nodes.

Being based on a DHT overlay, the proposed architecture avoids single points of failures, guarantees high scalability, load balancing and self organization capabilities, that allows us to implement a system that may integrate several thousand collaborative nodes. The viability of the proposed architecture is demonstrated through a prototype based on open source software, while large scale results referring to up to 10,000 nodes have been obtained through simulations.

We are now working on a hardened version of the distributed architecture that uses digital certificates and secure communications among the peers. We are also improving the detection algorithm in order to detect polymorphic and metamorphic malware where the cryptographic hashes of the binary code are different for every sample thus preventing the identification of the same threat.

7 Acknowledgments

This research has been funded by the IST-225407 EU FP7 project CoMiFin (Communication Middleware for Monitoring Financial Critical Infrastructures).

References

1. M. Colajanni, D. Gozzi, M. Marchetti, "Collaborative architecture for malware detection and analysis", Proc. of the 23rd International Information Security Conference (SEC 2008), Milano, Italy, Sept. 2008.
2. Snort - the de facto standard for intrusion detection/prevention, http://www.snort.org/
3. Nepenthes - finest collection -, http://nepenthes.mwcollect.org/
4. Prelude, http://www.prelude-ids.com/en/welcome/index.html
5. Prelude Universal SIM project Trac, https://trac.prelude-ids.org/
6. IDMEF standard described in rfc4765, http://www.ietf.org/rfc/rfc4765.txt
7. MySQL, http://www.mysql.com/
8. FreePastry, an open-source implementation of Pastry, http://www.freepastry.org
9. A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, Nov. 2001
10. M. Castro and P. Druschel and A.M. Kermarrec and A. Rowstron, "One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks", Proc. of the 10th SIGOPS European Workshop, Saint-milion, France, Sept. 2002
11. M. Castro and P. Druschel and Y. Charlie Hu and A. Rowstron, "Exploiting Network Proximity in Distributed Hash Tables", Proc. of the International Workshop on Future Directions in Distributed Computing (FuDiCo), Bertinoro, Italy, Jun. 2002

12. M. Castro and P. Druschel and A. Ganesh and A. Rowstron and D. S. Wallach, "Security for structured peer-to-peer overlay networks", Proc. of the 5th Symposium on Operating Systems Design and Implementaion (OSDI'02), Boston, MA, USA, Dec. 2002
13. R. Mahajan and M. Castro and A. Rowstron, "Controlling the Cost of Reliability in Peer-to-peer Overlays", Proc. of the 2nd International Workshop on Peer-To-Peer Systems (IPTPS'03), Berkeley, CA, USA, Feb 2003
14. P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", 8th Workshop on Hot Topics in Operating Systems (HotOS VIII), Schoss Elmau, Germany, May 2001
15. A. Rowstron and . Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP'01), Chateau Lake Louise, Banff, Canadav, May 2001
16. A. Rowstron and A.M. Kermarrec and M. Castro and P. Druschel, "SCRIBE: The design of a large-scale event notification infrastructure", Proc. of the 3rd International Workshop on Networked Group Communication (NGC2001), UCL, London, UK, Nov. 2001
17. M. Castro and M. B. Jones and A.M. Kermarrec and A. Rowstron and M. Theimer and H. Wang and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays", Proc. of the Infocom'03, San Francisco, CA, USA, Apr. 2003
18. Extensible Markup Language (XML), http://www.w3.org/XML/
19. The Java Database Connectivity (JDBC), http://java.sun.com/javase/technologies/database/index.jsp
20. Norman SandBox Information Center, http://sandbox.norman.com
21. CWSandbox, Behavior-based Malware Analysis remote sandbox service, http://www.cwsandbox.org
22. D. J. Malan and M. D. Smith, "Host-based detection of worms through peer-to-peer cooperation", Proc. of the 2005 ACM Workshop on Rapid Malcode (WORM 2005), Fairfax, VA, USA, Nov. 2005
23. C. L. Dumitrescu, "INTCTD: A Peer-to-Peer Approach for Intrusion Detection", Proc. of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), SMU Campus, Singapore, May 2006
24. C. V. Zhou and S. Karunasekera and C. Leckie, "A peer-to-peer collaborative intrusion detection system", Proc. of the 13th IEEE International Conference on Networks (ICON '05), Kuala Lumpur, Malaysia, Nov. 2005.
25. V. Yegneswaran, P. Barford, and S. Jha, "Global Intrusion Detection in the DOMINO Overlay System", Proc. of the ISOC Symposium on Network and Distributed Systems Security (NDSS'04), Feb. 2004.
26. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Application", Proc. of the ACM SIGCOMM2001, San Diego, CA, USA, Aug. 2001.
27. M. E. Locasto and J. J. Parekh and A. D. Keromytis and S. J. Stolfo, "Towards Collaborative Security and P2P Intrusion Detection", Proc. of the IEEE Information Assurance Workshop (IAW'05), Maryland, USA, Jun. 2005
28. Ramaprabhu Janakiraman, Marcel Waldvogel, Qi Zhang, "Indra: A peer-to-peer approach to network intrusion detection and prevention", Proc. of the 12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Linz, Austria, Jun. 2003