The Overlay Scan Attack: Inferring Topologies of Distributed Pub/Sub Systems through Broker Saturation

Leonardo Aniello
aniello@dis.uniroma1.it

Roberto Baldoni
baldoni@dis.uniroma1.it

Claudio Ciccotelli
ciccotelli@dis.uniroma1.it

Giuseppe Antonio Di Luna
diluna@dis.uniroma1.it

Francesco Frontali
f.frontali@hotmail.it

Leonardo Querzoni
querzoni@dis.uniroma1.it

Research Center on Cyber Intelligence and Information Security and
Department of Computer, Control, and Management Engineering Antonio Ruberti
Sapienza University of Rome
Via Ariosto 25, Rome, Italy

ABSTRACT

While pub/sub communication middleware has become mainstream in many application domains, little has been done to assess its weaknesses from a security standpoint. Complex attacks are usually planned by attackers by carefully analyzing the victim to identify those systems that, if successfully targeted, could provide the most effective result. In this paper we show that some pub/sub middleware are inherently vulnerable to a specific kind of preparatory attack, namely the Overlay Scan Attack, that a malicious user could exploit to infer the internal topology of a system, a sensible information that could be used to plan future attacks. The topology inference is performed by only using the standard primitives provided by the pub/sub middleware and assuming minimal knowledge on the target system. The practicality of this attack has been shown both in a simulated environment and through a test performed on a SIENA pub/sub deployment.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network topology*

Keywords

publish/subscribe, topology inference, security, network tomography

1. INTRODUCTION

Publish/subscribe systems can be considered nowadays as one of the mainstream middleware solutions for many-to-many asynchronous communications. In the last five to ten years the rise of large-scale cloud platforms further boosted the adoption of such solutions that can be thus easily considered as a fundamental building block for most of such platforms. Surprisingly enough, despite the growing importance of the role played by publish/subscribe middleware, little attention has been paid by the research community to the security of such systems. This lack of study and analysis, is quite dangerous as it may leave open vulnerabilities in systems many companies blindly rely upon.

A typical strategy employed in many cyber attacks includes three distinct steps: (i) first the attacker starts a reconnaissance phase where he silently studies the target system to understand the most effective strategy to adopt in delivering the attack, then (ii) he puts in place decoy actions aimed at disguising his identity and his real target and finally (iii) he performs the real attack. The reconnaissance phase is often performed using a set of tools and techniques whose main purpose is to collect detailed information about the target systems: characteristics of the software/hardware platforms used by the target, its internal topological structure, its defensive strategies, etc. A typical example of such activity is a port scan attack [12] aimed at identifying hosts on the target domain using software with known vulnerabilities. This kind of "preliminary" attacks are often underestimated by system administrators that consider them only slightly more annoying than email spams. Conversely, the amount of non-publicly available information that can be obtained with these attacks is striking and greatly help the attacker in carefully planning potentially devastating future attacks.

In particular, knowledge about the internal topology of a target system is extremely useful to plan a cyber attack. Studer et al. [19] illustrate the *Coremelt Attack*, which aims at congesting the network core by generating traffic among N malicious clients so that a target link of the system gets saturated. Network topology has to be known in order to properly generate such a traffic. A similar approach is also used in the *Crossfire Attack* [15]. Differently from the Coremelt attack, here the traffic is directed towards a large number of publicly accessible destinations rather than being routed among the malevolent bots themselves. The generated traffic is aimed at congesting some target links, which

DEBS'14, May 26–29, 2014, MUMBAI, India.

ACM 978-1-4503-2737-4/14/05 . . . $15.00.
http://dx.doi.org/10.1145/2611286.2611295.

again requires a certain level of knowledge of the target system.

Starting form this point of view, this paper analyzes a well known wide-area pub/sub middleware, namely SIENA, and shows how it is vulnerable to an attack, the *Overlay Scan Attack*, aimed at inferring the internal topology of a distributed deployment. By assuming only knowledge about a subset of the event brokers (*edge* brokers), the goal of the attack is to infer the full set of brokers (both edge and *internal* or *core* brokers) and the topology interconnecting them.

In particular, the paper introduces an overlay scan algorithm, namely *Real Mapper*, that is able to infer the full overlay network topology of a SIENA deployment by accessing the system with normal clients (i.e. publishers and subscribers) connected to edge brokers. Real Mapper injects properly shaped traffic in the pub/sub system and observes how event notification latency changes along the time. Thanks to these observations, it is possible to infer the position of overloaded internal brokers within the SIENA overlay network. In order to let the reader get through the full details of Real Mapper, we first introduce *Full Mapper*, an overlay scan algorithm that works in ideal conditions such as: (i) all links in the target overlay network have the same latency and (ii) the traffic flowing in the SIENA system is only the one produced by the attacker. Then we remove the previous assumptions and provide the complete description of Real Mapper.

The feasibility of the Overlay Scan Attack through the proposed algorithm is shown both via simulations and via tests performed in a real deployment of the SIENA publicly available prototype [1]. It is important to remark that, while all our tests have been performed considering SIENA as target case study, the Overlay Scan Attack can strike any system that performs event diffusion using spanning trees satisfying the *all-pairs path symmetry* property [5].

After this introduction, the paper continues by reporting the state of the art on this topic in Section 2; then it describes a reference system model in Section 3 and introduces both algorithms in Section 4; Section 5 reports on our experimental evaluation and, finally, Section 6 concludes the paper.

2. RELATED WORK

The issue of topology inference has been tackled in literature in different ways and distinct contexts. Topology inference techniques can be classified as *passive* or *active*.

Passive techniques are oblivious of the layer of the target network and employ non-intrusive measurements to gather required information, such as packets delay. This class of techniques is closely related to the *Distance Realization Problem* [13], which concerns the construction of a network topology on the base of a set of distances among network nodes. It was proved by Dress [10] that for a given distance matrix it is always possible to find an optimal realization. This problem is NP-complete [2], but there exist algorithms that provide an approximate solution within a fixed bound from the optimum [6]. In case the topology is a tree, the problem can be solved optimally with $O(n^2)$ time complexity [7].

Active topology inference techniques are intrusive in the sense that they use probes. These probes are specific for a particular network layer, which implies that each active technique is tailored for a single layer.

Bejerano [4] proposes a technique for inferring the topology of large heterogeneous Ethernet LANs that works at the *data link layer* and exploits the fact that layer-2 elements can provide their Address Forwarding Tables (AFT). Results show that this technique is very successful at discovering the real topology regardless of uncooperative layer-2 elements and its time complexity is $O(n^2)$.

Inference techniques for the *Internet layer* usually employ traceroute-like tools to discover single end-to-end paths. As an example, Jin et al. [14] present the *Isomap Merging Algorithm* that has $O(n^3)$ time complexity and provides an average accuracy of 62%. Such a poor result is mainly due to the limitation of these traceroute-based techniques, which lies in the presence of *anonymous routers* that don't provide the expected information.

Ni et al. [16] try to overcome this limitation by proposing the *network tomography* approach, which only uses end-to-end measurements like packets loss and delay to derive network structure without relying on the cooperation of internal nodes. They devise one algorithm for static topologies ($O(n^2 \log n)$ time complexity) and another one for networks where nodes can join and leave ($O(n^2)$), both with a precision of about 50%. This lack of precision depends on the fact that path lengths derived from loss and delay measurements, if quite small, can be easily distorted. In order to improve the detection accuracy, they propose an hybrid approach that includes traceroute-based measurements.

Topology inference techniques for the *overlay layer* are typically designed for p2p system that are characterized by high nodes turnover, and are usually based on capturing snapshots of network structure. Stutzbach and Rejaie [20] present *Cruiser*, a crawler for inferring the topology of Gnutella network. They show that about 30%-38% of the peers are not reachable, either because they are overloaded or behind a firewall.

Fragouli et al. [11] describe a more general technique which works for multicast overlay networks that implement the *network coding paradigm*[1]. An alternative approach based on gossiping (They employed the Vivaldi algorithm [8]) and clustering (they used agglomerative clustering [9]) is outlined by Schutt et al [18], with focus on identifying the location of data centers and their sizes To the best of our knowledge, none has ever tackled the problem of topology inference in pub/sub systems.

In the field of publish/subscribe systems, Wun et al. [21] present a taxonomy for a specific type of attack against such kind of systems, namely the Denial of Service (DoS). In particular, the authors describe two key characteristics of DoS attacks in the context of pub/sub systems: localization and transmission. The *localization* of a DoS attack concerns the fact that a publication flooding attack at an edge broker can overload that broker, preventing the attack to propagate to the internal brokers. The *transmission* of a DoS attack concerns the possibility to propagate the effect of a DoS attack from an edge node to another remote edge one without affecting the internal nodes along the path. This is due to

[1]The network coding paradigm is based on the idea that independent information flows can be linearly combined throughout the network to give benefits in terms of throughput and complexity.

the overhead introduced at the remote edge node to deliver notifications to all subscribers.

3. SYSTEM MODEL

In this paper we consider a managed wide-area broker-based distributed publish/subscribe systems [3]. The system goal is to efficiently diffuse events issued by content producers (called *publishers*) to a set of recipients (called *subscribers*). An event is a piece of information characterized by a predefined structure (called *event schema*). Such structure can either be very simple like the name of a single topic (*topic-based* event selection model), or based on multiple attributes defined in different domains (*content-based* event selection model). Each subscriber defines through a subscription, by means of constraints expressed on the event schema, a subset of all possible events. We say that an event *matches* a subscription if the event content satisfies the constraints defined by the subscription. The set of recipients for each published event can thus dynamically change on the basis of currently issued subscriptions. A distributed *event notification service* (ENS) constituted by multiple brokers mediates among clients implementing the correct event routing functionalities of the system. The only assumptions we make about the internal functioning of the ENS is that it uses application level multicast trees to diffuse events, and that these diffusion trees satisfy the well-known *all-pairs path symmetry* property [5]. This property states that the event notification path traversed from a source broker B_x to a destination broker B_y by all events published in B_x is equal, but with opposite direction, to path traversed from broker B_y to broker B_x by all events published in B_y. Multicast trees built using shortest paths usually satisfy this property. The SIENA publish/subscribe system is a typical example of an ENS satisfying all the previous assumptions.

We model our publish/subscribe system as a tuple $\mathcal{P} : \langle G(V,E), d, r\rangle$ where: $d : E \rightarrow R^+$ $r : V \rightarrow \{r_{min}, r_{max}\}$ and G is an undirected graph made up of brokers V and application level links E interconnecting them. The functions in the tuple are respectively a delay function d that assigns a given latency to each edge, that is a message will be delayed of $d(e)$ time units when it traverses edge e, and a computational rate r that assigns a maximum message dispatching rate to each broker in the system. Here we assume that brokers are characterized by either a small computational rate r_{min} or a large one $r_{max} \gg r_{min}$; this simplifying assumption is actually satisfied by many systems where computational resources are often homogeneous to facilitate their management. In section 4.1 we will further discuss this assumption and show how it can be relaxed. The tuple represents all the fundamental characteristics of the distributed publish/subscribe system including its topological structure.

Here we assume that the computation time needed on a broker v to correctly manage and route an event is negligible with respect to network latencies as long as the broker is asked to route less than $r(v)$ events per second; when pushed above this threshold we say that the broker is *saturated*, and that its management of an event adds a latency factor that is in the same order of magnitude of network latencies. In particular, according to the queuing theory, we modeled each broker as a single server with a single queue, characterized by a fixed service rate $r(v)$. We also assume that links are symmetric and that available bandwidth is large enough to avoid becoming a bottleneck, that is, computational resource are saturated before network may become saturated.

The set of brokers V is partitioned in two subsets $V = V_e \cup V_i$ where V_e is the set of *edge brokers*, while V_i is the set of *internal brokers*. External brokers are publicly announced by the system owner as they serve as service access points for external clients (both publishers and subscribers), while internal brokers represent core infrastructure components for the systems and are solely dedicated to event routing, therefore their existence is not announced publicly.

The publish/subscribe systems we consider are supposed to be publicly available, thus we can expect application traffic to be present within the ENS.

We assume the presence of an adversary whose purpose is to infer G by only knowing V_e and by using the publish/subscribe system as a normal user. We assume the adversary is able to control at least three clients, and that these clients can be connected at his will to any of the edge brokers. Furthermore, we assume that the clocks at these clients can be synchronized such that their drift is order of magnitude smaller than the latency of any link in E.

4. TOPOLOGY INFERENCE ALGORITHM

This section introduces an algorithm that can be used to implement the Overlay Scan Attack. The algorithm has a single purpose: collect information on the topology characteristics of a distributed event notification service.

We first introduce a preliminary version, called *Full Mapper*, that is based on two simplifying assumptions that will help us introduce the most important characteristics of the algorithm without the need to deal with some tricky issues.

Then we will remove these assumptions and introduce the *Real Mapper* algorithm that fully adheres to the system model introduced in Section 3 and can thus be adopted in realistic settings where link delays may vary.

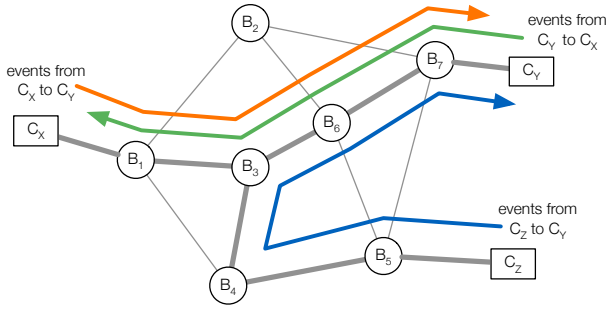
Both the *Full* and the *Real Mapper* algorithms are based on a same idea: by injecting properly shaped application-level traffic in the pub/sub system and by pushing it to the saturation point, i.e. close to the point where at least one broker is working slightly above its maximum computational rate r, it is possible to induce notification delays in the system. The amount of these delays, probed at several points of the system, can be correlated to pinpoint the position of the saturated broker in the overlay network. The iteration of this analysis creates enough information to infer the full internal topology of the system.

4.1 Full Mapper

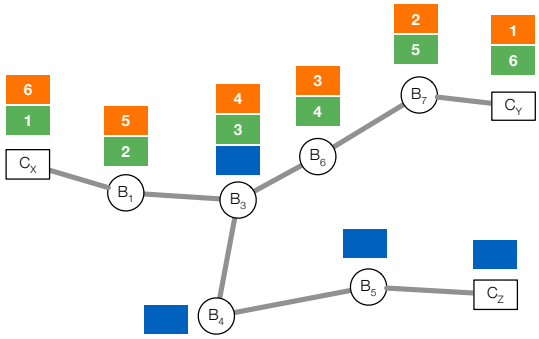
The Full Mapper algorithm works in three phases:

1. Setup and distance estimation
2. Path discovery
3. Path merging

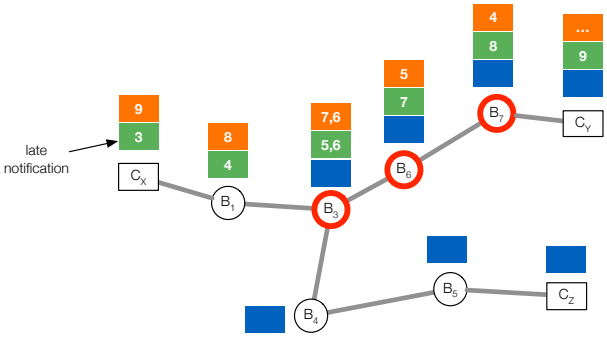
It assumes that all links in the ENS overlay network have the same delay d and that this delay is known by the attacker; the delay between a client controlled by the algorithm and an edge broker is assumed to be negligible with respect to d. Furthermore, it assumes that the only events routed in the ENS are those generated by the attacker. These unrealistic assumptions have been made to simplify the comprehension of the algorithm and will be removed in



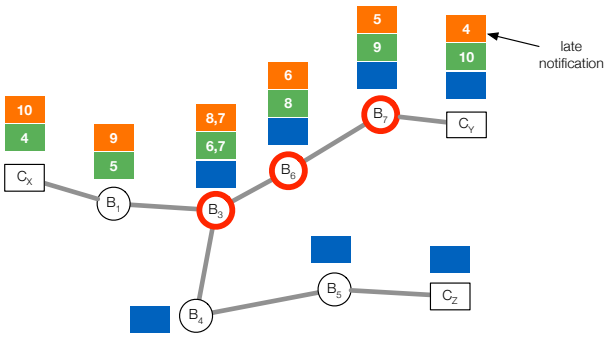
(a) Client initialization and event flows setup.



(b) Flows of events just before broker saturation kicks-in.



(c) Client c_x detects the first late notification (green event with label 3)



(d) Client c_y detects the first late notification (orange event with label 4)

Figure 1: Full Mapper functioning example.

the next Section with the introduction of the Real Mapper algorithm.

Setup and distance estimation.

During this first phase, the algorithm starts by initializing several clients such that each edge broker B_x has attached a client c_x.

Figure 1a shows an example ENS constituted by seven event brokers with three attached clients (c_x, c_y and c_z). Thick lines represent links used by the ENS to route events produced by these clients, while thin lines are links that we suppose are not used in this example.

It then considers each possible pair of clients and configures one as a publisher c_x and the other as a subscriber c_y. Both are configured in such a way that, with high probability, c_y will be notified by the ENS of all and only the events produced by c_x. This can be obtained by issuing a very specific subscription in a section of the event space where other publishers will publish with very low probability. If this statistical information is not directly available to the attacker it can be easily inferred by silently monitoring the set of events published in the system. With clients configured in such a way, it starts publishing from c_x to calculate the average notification latency at c_y.

This procedure is iterated until notification latencies are available for every possible pair $\langle c_x, c_y \rangle$. These values are recorded in a delay matrix D, with size $|V_e|^2$. Note that, due to the initial assumptions, each value in D represents the hop count of the path connecting the two corresponding brokers in the ENS overlay network. Indeed, if we divide the path length (in time units) by the single hop delay d, we obtain the length of the path in number of hops.

Path discovery.

In this second phase the algorithm iterates over all pairs $\langle c_x, c_y \rangle$ and, for each of them, iterates over all the remaining clients selecting one at a time (c_z). For each given triple $\langle c_x, c_y, c_z \rangle$ the algorithm performs the following steps:

1. configures c_x and c_y as both publishers and subscribers such that c_x (resp. c_y) will be notified of the events published by c_y (resp. c_x);

2. the two clients synchronously start to publish events at a rate of $R/3$ events per second (R is an estimation of the maximum dispatching rate); each event is tagged with a sequence number;

3. client c_z and c_y are configured as publisher (the former) and subscriber (the latter) such that c_y will be notified of the events published by c_z;

4. c_z starts to publish events at a rate slightly larger than $R/3$ events per second;

The computational complexity of the algorithm is dominated by the cost of the Path discovery phase, which is proportional to the number of triples to analyze. Since such number is equal to $3\frac{n!}{3!(n-3)!}$, where $n = |V_e|$, the complexity

is $O(n^3)$. While this may sound like an excessive computational cost we remind to the reader that our goal is to show that the attack is, in fact, feasible; conversely, providing an efficient algorithm to perform the attack is out of the scope of this paper.

According to the assumptions reported in Section 3, there are two possible values for r, that the algorithm does not know. However, it is possible to estimate a value for the maximum processing rate by issuing publications to edge brokers at an increasing rate until saturation is detected. In this way, the value R we obtain is such that $r_{min} \leq R \leq r_{max}$. This implies that the algorithm is not able to saturate some brokers, which in turn doesn't allow the algorithm to detect them. We refer to such brokers as *invisible brokers*, and we will discuss in Section 5.1 how they impact on the accuracy of the algorithm.

Figure 1a shows how clients c_x, c_y and c_z are configured to produce three event flows represented in the picture with three different colors. Figure 1b reports events produced by clients c_x, c_y and c_z traveling in the ENS while no broker is saturated. In this case c_x and c_y see events produced by the other client arriving at a constant rate ($R/3$), without any notification delay.

The stream of events injected by c_z produces a saturation in some of the brokers in the overlay network; in particular, all brokers in the subpath that is common to the paths $[B_x, B_y]$ and $[B_z, B_y]$ are saturated as they are asked to manage more than R events per second. Considering again our example in Figure 1b, events produced by c_z that have to be notified at c_y are represented by blue boxes. B_3 is the first broker that gets saturated.

If we now consider only the events that travel from c_x to c_y and vice-versa, it is worth noticing that they are delayed only in a specific section of the path connecting the two clients, which is the subpath between some broker S (the broker at the intersection between the paths $[B_x, B_y]$ and $[B_z, B_y]$ and c_y. We refer to S as the *S-node* of the triple $\langle B_x, B_y, B_z \rangle$. In our example (see Figure 1b) broker B_3 is the S-node. Events published by c_x and c_y that are currently traveling through this broker (event 4 in the orange flow, and event 3 in the green flow) will thus be the first ones whose notification will be delayed on c_x and c_y.

In the example (Figure 1c), we can see that c_x is the first client that receives a late notification (the notification of event 3 in the green flow). The client keeps track of this sequence number as the other client c_y has not seen its late notification, yet.

The difference between the sequence numbers of the first delayed event received by c_x and the first delayed event received by c_y represents the difference between the distances from edge broker B_x to broker S, and from edge broker B_y to S (measured in number of hops).

When client c_y (Figure 1d) receives its late notification (the notification of event 4 in the orange flow) the algorithm can calculate the difference between the two sequence numbers (the difference is 1 in this example) that is exactly the difference between the length of the subpath connecting B_1 to B_3 (i.e. the S-node) and the length of the subpath connecting B_3 to B_7.

The analysis of a single triple thus provides three important information: the existence of a broker S (that may be an internal broker), and the length of two subpaths in the ENS overlay network.

By iterating over all possible client triples, the algorithm builds a set T where each entry includes the triple $\langle B_x, B_y, B_z \rangle$, the related S-node S and the distances $d(B_x, S)$ and $d(B_y, S)$ (measured in number of hops).

The way the path discovery phase works allows us to partially relax the assumption about brokers computational rate. Rather than constraining all the brokers with small computational rate to have the same r, we can indeed derive a range $[r^l_{min}, r^h_{min}]$ of values for r_{min} that still allows our algorithm to work. The value of r^l_{min} can be easily found by searching for the maximum publishing rate for each pair of edge brokers, and then taking the minimum value. In order for our algorithm to work, we have to ensure that no broker gets saturated by two streams, each flowing at $R/3$ events per second. This implies that R has to be set such that $\frac{2}{3}R < r^l_{min}$. On the other hand, we need to saturate any non-invisible broker, i.e. $R > r^h_{min}$. By consequence, the maximum value allowed for r_{min} is $\frac{3}{2}r^l_{min}$, so the range of allowed values is $[r^l_{min}, \frac{3}{2}r^l_{min}]$.

Path merging.

The information contained in T about internal brokers of the overlay contains with high probability a large number of duplicates, i.e. brokers that are listed separately in T, but that actually represent the same internal broker that has been identified in the analysis of two distinct triples. The path merging phase has the aim of getting rid of these duplicates and build the final inferred topology of the ENS overlay network. It consists of three phases.

In the first phase, the set of paths P is built in this way. For each pair of edge brokers $\langle B_x, B_y \rangle$, a path $[B_x, B_y]$ having length $d(B_x, B_y)$ (measured in number of hops) is added to P. Each broker in any path is assigned a unique identifier. During the merging phase, brokers in distinct paths can be recognized as being the same broker. When this happens, the same identifier is assigned.

In the second phase, for each entry of T the paths $[B_x, B_y]$ and $[B_z, B_y]$ are retrieved from P and the following steps are executed

1. the broker in the path $[B_x, B_y]$ at distance $d(B_x, S)$ from B_x is replaced with S
2. the broker in the path $[B_z, B_y]$ at distance $d(B_y, S)$ from B_y is replaced with S
3. all the brokers of the subpath $[S, B_y]$ of the path $[B_x, B_y]$ are replaced with the brokers of the subpath $[S, B_y]$ of the path $[B_z, B_y]$

In the third phase, the inferred graph $G' = (V', E')$ is built as the union of all the paths in P.

4.2 Real Mapper

The Real Mapper algorithm adopts exactly the same strategy as the Full Mapper algorithm, but for some minor modification needed to take into account application-level traffic generated by normal users and the different latencies characterizing the network links.

Application-level traffic can be considered as a sort of steady state noise present in the system, which makes measurements non-deterministic. We cope with this problem by executing more iterations over time for each triple and choosing the most frequent result. In this way, possible deviations due to such noise are likely to be discarded and

	no invisible brokers	with invisible brokers
Full Mapper	*Scenario A* $\circ\ r_{min} = r_{max} = 150\ msg/s$ $\circ\ d = 40\ ms$	*Scenario B* $\circ\ r_{min} = 150\ msg/s$ $\circ\ r_{max} = 1000\ msg/s$ $\circ\ d = 40\ ms$
Real Mapper Each publisher is assigned a set of subscribers according to a Zipf-like distribution. Publications are sent at a rate following a Poisson distribution. The ratio $\frac{\#slow\_channels}{\#fast\_channels}$ is 0.25.	*Scenario C* $\circ\ r_{min} = r_{max} = 150\ msg/s$ $\circ\ d_1 = 4\ ms$ (*fast channels*) $\circ\ d_2 = 60\ ms$ (*slow channels*)	*Scenario D* $\circ\ r_{min} = 150\ msg/s$ $\circ\ r_{max} = 1000\ msg/s$ $\circ\ d_1 = 4\ ms$ (*fast channels*) $\circ\ d_2 = 60\ ms$ (*slow channels*)

Table 1: Simulation Scenarios. For each scenario, maximum broker dispatching rate (r) and link delay (d) are reported.

resulting measurements negligibly impacted. In Section 5.2 we will see in practice how the number of iterations affects the accuracy of the algorithm.

The Real Mapper algorithm gets rid of the assumption on constant network links and assumes a more realistic system model where each link is possibly characterized by a different latency amount. This modification has a deep impact on the algorithm because end-to-end latencies collected in the matrix D cannot be anymore considered the hop length of the corresponding paths.

In order to calculate subpath lengths, the algorithm considers a basic latency time unit u such that the lengths recorded in P are obtained by diving by u, rather than by d. The value of u is a parameter of the algorithm and must be chosen such that none of the ENS links has a latency smaller than u. Otherwise, the inferred paths would have a number of brokers lower than they actually have in practice. On the other hand, choosing u too small negatively affects the precision in inferring the ENS topology because we would be more likely to miss some mergings.

The introduction of this approximation for path lengths will create possible *false brokers* in P, i.e. brokers that actually do not correspond to any real broker in the ENS. During the Path merging phase, only the S-nodes are merged while false brokers are removed. Consequently, the third step of the second phase of the Path merging is not required any more.

Note that due to this modification the algorithm is no more capable of identifying internal brokers with degree 2 as they are non discernible from false brokers. However, it should be noted that the usefulness of such brokers in a ENS is debatable as they dot not increase in any useful way the capacity, performance or reliability of the ENS.

5. EXPERIMENTAL EVALUATION

This section provides an evaluation of the effectiveness of the algorithm introduced in Section 4. The evaluation was firstly performed in size-varying pub/sub systems in a simulated environment (see Section 5.1) to show the sensitiveness of the algorithm to various characteristics of the target system. Then we implemented the algorithm to work on the official prototype of the SIENA publish/subscribe system and run several tests to understand if the Overlay Scan Attack can actually be performed on a real setting (see Section 5.2).

5.1 Simulations

In order to test our algorithm in a simulated environment we implemented it on top of a simple publish/subscribe mechanism based on a distributed set of event routers. For the sake of simplicity we avoided to implement a full-fledged SIENA-like content-based pub/sub system, but rather implemented a simple multi-source multicast mechanism: every broker in the system is equipped with a FIFO queue used to store incoming messages, and every message is always forwarded to the next broker lying on the shortest path toward the destination. Nodes are connected through *delay channels*, a particular class of virtual links with fixed bandwidth but variable delay. The prototype implementation of the algorithms consists of two software modules: (i) a *Client* whose instances connect to edge brokers and act as publisher and/or a subscriber and (ii) the *Controller* that coordinates the clients work. All the code was integrated in the Omnet++ simulation environment.

In our tests we used two kinds of topologies, namely *random topologies* and *cluster topologies*, to test the behavior of the Real Mapper algorithm in different settings. We also varied the value of r in order to include invisible brokers in the test and assess their impact on the algorithm performance. Finally we considered a simplified network model where links can be roughly divided in two families, *slow channels* and *fast channels*, with different latency values. Tests were conducted on both the Full Mapper and the Real Mapper algorithm. Results from the former constitute an interesting baseline for the evaluation of the latter algorithm.

On the basis of these parameters we defined four different test scenarios: two dedicated to Full Mapper, and two dedicated to Real Mapper. The details of these four scenarios are reported in Table 1.

In order to evaluate the effectiveness of the algorithms, we defined several accuracy metrics:

- *broker detection rate (α)*: the ratio between the number of inferred internal brokers and the total number of internal brokers;
- *link detection rate (β)*: the ratio between the number of inferred links and the total number of links;
- *N-degree broker detection rate (η)*: the ratio between the number of inferred internal brokers with degree $N > 2$ and the total number of this kind of brokers;

If invisible brokers are present in the system, it is necessary to introduce an additional metric:

- *graph distance (δ)*: the minimum number of edit operations (addition/removal of brokers and edges) required to transform the inferred topology so that it becomes isomorphic to the original one [17]; if the inferred topology is isomorphic to the original one, then $\delta = 0$.

Beyond the quality of the obtained outcomes, we evaluated performances of our algorithms measuring its *message complexity* and *execution time*.

5.1.1 Simulations on Random Topologies

A random topology is composed by a variable number of internal brokers (from 3 to 15) and edge brokers (up to 20 nodes). We ran simulations to assess the accuracy of Real Mapper, to compare the performance of Full and Real Mapper and to evaluate the impact of invisible brokers on the accuracy of both Full and Real Mapper.

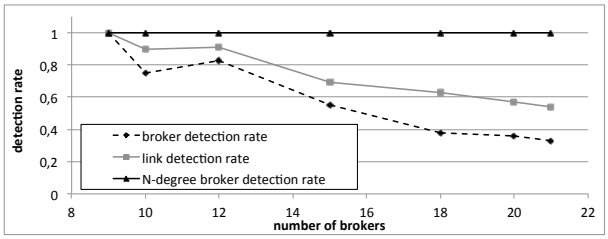


Figure 2: Broker, link and N-degree broker detection rates for the Real Mapper (Scenario C) when executed on a random topology with a fixed number of edge brokers and an increasing number of internal brokers.

Detection Accuracy.

In order to assess the accuracy of the algorithms, we used a random topology with a fixed number of edge brokers and a variable number of internal brokers. While the Full Mapper showed a perfect accuracy in detecting both brokers and links (Scenario A), the Real Mapper exhibited a decreasing accuracy as the number of internal brokers increased (Scenario C, see Figure 2).

This bad performance level was mainly caused by the inability of the algorithm to correctly detect internal brokers with degree 2. The same graph also show how the Real Mapper always correctly detected brokers with degree larger than 2 ($\eta = 1$).

Performance Comparison.

We first compared the performances of Full Mapper and Real Mapper in terms of execution time and message complexity for a random topology with a fixed number of edge brokers as the number of internal brokers increases. As Figure 3 shows, both metrics increase linearly with the number of internal brokers. However, as could be expected, the Real Mapper algorithm sports worse performance on both aspects with respect to the Full Mapper. This performance difference is due to the large number of iterations that the Real Mapper algorithm must perform to take into account variable network latencies.

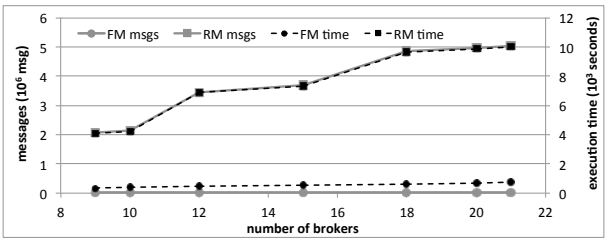


Figure 3: Performance comparison between Full Mapper (FM, Scenario A) and Real Mapper (RM, Scenario C) in execution time and number of messages on a random topology with a fixed number of edge brokers and an increasing number of internal brokers.

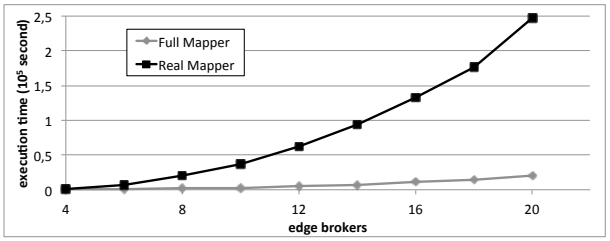


Figure 4: Execution time comparison between Full Mapper (Scenario A) and Real Mapper (Scenario C) on a random topology with a fixed number of internal brokers and an increasing number of edge brokers.

We then investigated how performance are affected by the number of edge brokers, by simulating the execution of the algorithms on a random topology with a fixed number of internal brokers. Figure 4 shows that execution time increases as more edge brokers are included in the topology, and that again Real Mapper performs worse than Full Mapper.

Impact of Invisible Brokers.

We also analyzed how the presence of invisible brokers affects the accuracy of the inferred topology. We built a random topology and started setting brokers with degree larger than 2 in invisible brokers by setting their dispatching rate to r_{max} (see Table 1).

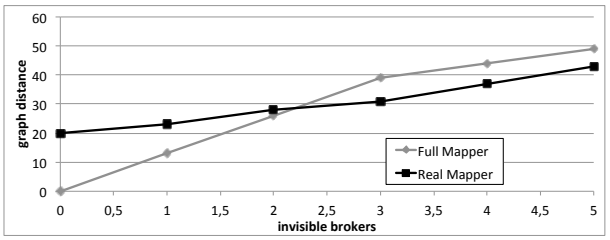


Figure 5: Accuracy comparison between Full Mapper (Scenario B) and Real Mapper (Scenario D) on a random topology with a fixed number of brokers and an increasing number of invisible brokers.

Figure 5 compares the accuracy of the algorithms measured as the distance graph between the original topology

and the inferred one. As expected, the graph distance increases as more brokers become invisible because they cannot be saturated. An interesting result standing out in that figure is that the Real Mapper begins to perform better than the Full Mapper after the insertion of the third invisible broker.

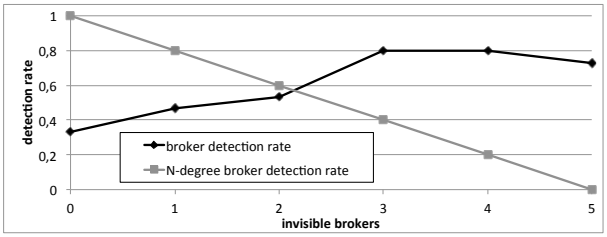


Figure 6: Comparison between *broker detection rate* and *N-degree broker detection rate* of the Real Mapper (Scenario D) on a random topology with a fixed number of brokers and an increasing number of invisible brokers.

The reason why this happens can be explained by looking at Figure 6, where it is shown how α and η vary for the Real Mapper when invisible brokers are added. For each new invisible broker, η decreases because an additional N-degree broker cannot be detected. On the contrary, α increases (this happened three times out of five in our tests) because another broker gets saturated in place of the invisible one, and if such broker is 2-degree (that couldn't be detected before) then the Real Mapper can spot it now.

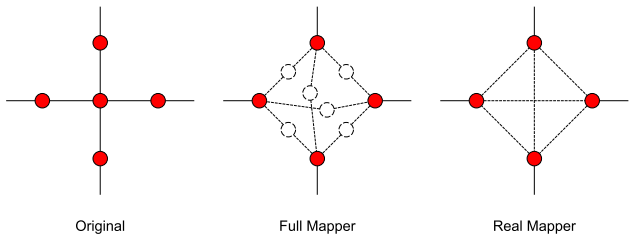


Figure 7: Example of the effect of an invisible broker (the central broker of the original topology) on the topologies inferred by Full Mapper (Scenario B) and Real Mapper (Scenario D). Wrongly inferred brokers and links are dotted.

A practical example where this situation occurs is drawn in Figure 7. In the original topology (the leftmost), the 4-degree broker is invisible and the other 2-degree brokers are visible. Both Full and Real Mapper always saturate one of the 2-degree brokers instead of the 4-degree one. The Full Mapper knows the link delay, so it is aware that another broker lies in between any of the 6 pairs of original 2-degree brokers, but it cannot recognize that such broker is always the same, which leads to the addition of several wrong brokers and links in the inferred topology (the central one in Figure 7). The Real Mapper instead doesn't know the link delay, and adds a certain number of virtual brokers in all the 6 paths between any pair of the original 2-degree brokers. All these virtual brokers are removed because they are 2-degree, and only the links remain (rightmost topology in Figure 7).

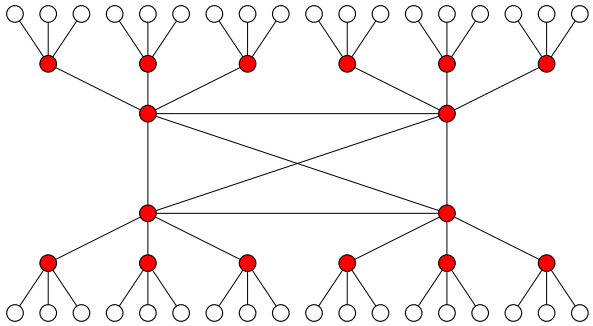


Figure 8: 4-tree-cluster topology with mesh-connected core. Edge brokers are white-filled, internal brokers are red-filled.

5.1.2 *Simulations on Cluster Topologies*

Cluster topologies are composed by tree-structured clusters of brokers. Their structure is meant to mimic many real-world overlay networks where local subnets (clusters) are connected through gateways linked in a *core* structure. We ran tests using 4-cluster topologies that differ in the way the clusters are connected together in the *core* of the network, by using either a bus, a ring or a mesh (see Figure 8). Each single cluster is a tree with 4 internal brokers and 9 edge brokers.

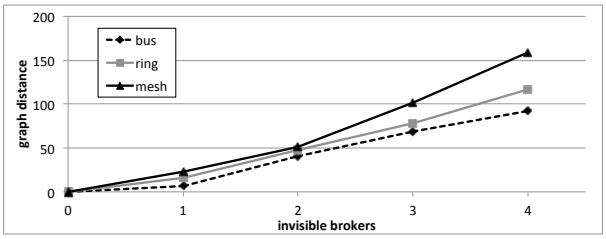
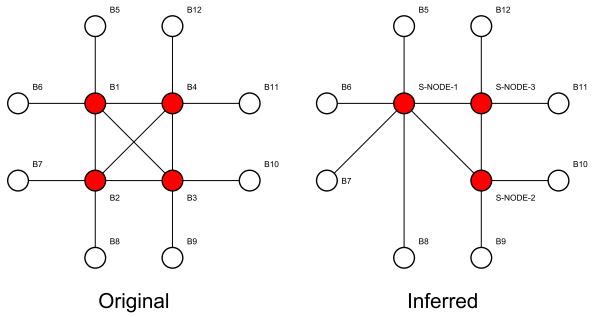
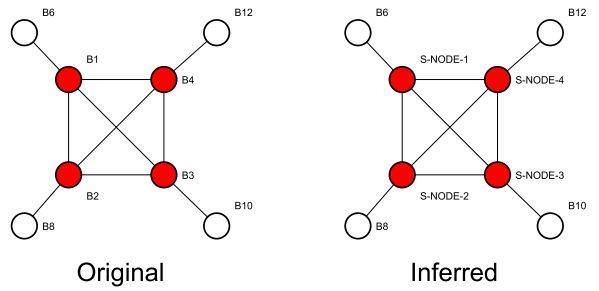


Figure 9: Accuracy comparison of the Real Mapper (Scenario D) on distinct cluster topologies by varying the number of invisible brokers.

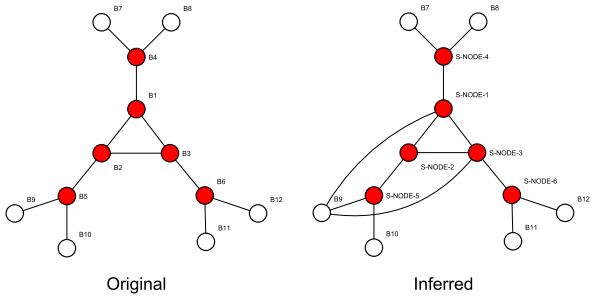
We measured how the accuracy of the Real Mapper is affected by the presence of invisible brokers for the three cluster topologies. Results are plotted in Figure 9. As expected, for all the topologies the graph distance increases as more brokers become invisible. Another result emerging from this simulation concerns the difference in detection accuracy between mesh-connected, ring-connected and bus-connected topologies: the more complex the connection is in the number of links, the less accurate the detection is.



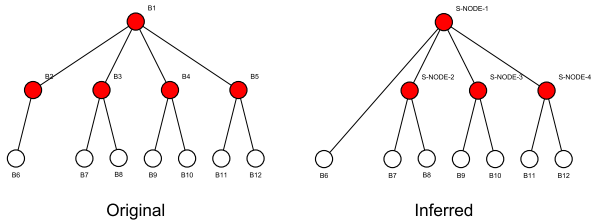
(a) 4-cluster topology with mesh-connected core (Topology $T1$).



(b) 4-cluster topology with mesh-connected core (Topology $T2$).



(c) 3-cluster topology with ring-connected core (Topology $T3$).



(d) 4-cluster topology with bus-connected core (Topology $T4$).

Figure 10: Comparison between original and inferred topology for different scenarios

5.2 Tests on SIENA prototype

In order to assess the practical effectiveness of our algorithms, we implemented the Real Mapper and tested it on a cluster of hosts running the official SIENA prototype [1]. Our goal is providing proofs in support of the feasibility of using the methods we described in Section 4 and simulated in Section 5.1 to gain knowledge of the topology of a real distributed pub/sub system.

We setup 12 virtual machines, each equipped with 2x2.8 GHz CPUs, 2 GB RAM, 5 GB of disk storage, all running Ubuntu 12.04 64-bit as operating system. An instance of SIENA version 2.0.3 was deployed on each machine, and the *DVDRPControl* utility (included in the SIENA software bundle) was used to connect brokers so as to create the desired topologies. In order to emulate a WAN deployment, we used WANem version 2.2 (`http://wanem.sourceforge.net/`) and set link delay to 100 ms.

A number of challenges arise when moving from simulation in an ideal environment to experimental evaluation of a real prototype. The main issue to address regards the presence of relevant noise in the expected link latencies and dispatching delays in the SIENA cluster, which considerably impacts on the accuracy of inferred topologies. Real Mapper deals with this problem by increasing the number of iterations for each triple of edge brokers and considering the most frequent result. Another problem we had to tackle is the presence of a transient state when a new flow of publications is started. During this period, publications are subjected to much higher delays than expected, which can mislead the Real Mapper to believe that a broker is saturated. We had to explicitly code routines to handle these transient states and avoid taking measurements during these periods.

The first topology ($T1$) we tested was a 4-cluster mesh-connected topology where each cluster has one N-degree internal broker and two edge brokers (see Figure 10a). This is a very challenging topology for the Real Mapper because there are many edge brokers, therefore a lot of triples to analyze (they are 168, see Section 4), and a few internal brokers to discover, so several distinct paths have to be correctly merged in order to accurately infer the original topology. This is the reason why the algorithm missed one internal broker and some links.

Then we tested a simpler topology ($T2$) which is mesh-connected like the first one but each cluster has one N-degree internal broker and one edge broker (see Figure 10b). The number of triples to analyze in this topology is one order smaller compared to the previous one (12), which allows the Real Mapper to correctly spot all the internal brokers and all the links.

We also ran the Real Mapper on a 3-cluster ring-connected topology ($T3$) where each cluster has two N-degree internal brokers and two edge brokers (see Figure 10c). The number of triples to analyze is pretty large (60), indeed the Real Mapper introduced two wrong links. Nevertheless, it also managed to correctly detect all the internal brokers.

We finally executed the Real Mapper on a 4-cluster bus-connected topology ($T4$) where each cluster has one N-degree internal node and two edge brokers, apart from one that has a single edge broker, and there is an internal broker playing the role of the bus (see Figure 10d). Although the number of triples to analyze is much larger compared to the previ-

ous topology (105), the algorithm managed to obtain the best result it could get. Indeed, it only missed the unique 2-degree internal broker of the original topology, which is known to be impossible to detect for the Real Mapper.

6. CONCLUSIONS

In this paper we considered overlay-based systems that perform event diffusion using shortest paths and we addressed the problem of inferring the internal overlay network topology when only a few edge nodes of the overlay are known by an attacker. We refer to this as the Overlay Scan Attack. The main result reported here is that properly shaped event flows can be exploited to implement the Overlay Scan Attack successfully. This security weakness has serious practical implications as these information could be exploited by an attacker to plan potentially dangerous attacks. The paper introduced an overlay scan algorithm, namely Real Mapper, and showed how it is able to capture the overlay topology of a SIENA publish/subscribe system. The experimental evaluations presented in Section 5.2 provided evidence about the feasibility of the attack.

The results reported in this paper open several further interesting issues that should be studied further. Firstly, while the tests performed on the SIENA prototype provided encouraging results, they also outlined several difficulties that arise when noise, present in the system, impacts the algorithm's accuracy. It could be interesting to test the Real Mapper algorithm on a real production setting where this problems could be amplified. Furthermore, the feasibility of the Overlay Scan Attack shown in this paper opens up the problem on how system administrators could protect their infrastructures. Given the limited amount of information used by Real Mapper to infer the topology it seems reasonable that introducing properly shaped noise in the system and adequately differentiating its internal characteristics may help to obfuscate the topology. However, this would come at a non negligible cost, creating a set of tradeoffs that are worth being analyzed.

Another possible technique for detecting this kind of scan can be based on the recognition of anomalous client behaviors, i.e. detecting when clients generate event streams in a coordinated fashion aimed at saturating some brokers. broker saturations and the patterns of client publications can be easily tracked and then properly correlated in order to spot suspicious activities, so as to take appropriate countermeasures like blacklisting certain clients.

7. ACKNOWLEDGMENTS

This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research and by the academic project C26A133HZY funded by the University of Rome "La Sapienza".

8. REFERENCES

[1] Siena web site. `http://www.inf.usi.ch/carzaniga/siena/`.

[2] I. Althöfer. On Optimal Realizations of Finite Metric Spaces by Graphs. *Discrete Comput. Geom.*, 3(2):103–122, 1988.

[3] R. Baldoni, L. Querzoni, S. Tarkoma, and A. Virgillito. Distributed Event Routing in Publish/Subscribe Communication Systems. In B. G. H. Miranda, L. Rodriguez, editor, *MiNEMA State-of-the-Art*. Springer Berlin / Heidelberg, February 2009.

[4] Y. Bejerano. Taking the Skeletons Out of the Closets: A Simple and Efficient Topology Discovery Scheme for Large Ethernet LANs. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–13, April 2006.

[5] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A Routing Scheme for Content-Based Networking. In *Proceedings of the 23rd IEEE International Conference on Computer Communications (INFOCOM)*, 2004.

[6] F. Chung, M. Garrett, R. Graham, and D. Shallcross. Distance Realization Problems with Applications to Internet Tomography. *Journal of Computer and System Sciences*, 63(2):432–448, 2001.

[7] J. C. Culberson and P. Rudnicki. A Fast Algorithm for Constructing Trees from Distance Matrices. *Inf. Process. Lett.*, 30(4):215–220, Feb. 1989.

[8] F. Dabek, R. Cox, M. F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. *ACM SIGCOMM*, 2004.

[9] W. H. E. Day and H. Edelsbrunner. Efficient Algorithms for Agglomerative Hierarchical Clustering Methods. *Journal of Classification*, 1984.

[10] A. W. M. Dress. Trees, Tight Extensions of Metric Spaces and the Cohomological Dimension of Certain Groups: a Note on Combinatorial Properties of Metric Spaces. *Adv. in Math.*, 53, 1984.

[11] C. Fragouli, A. Markopoulou, and S. N. Diggavi. Topology Inference Using Network Coding. In *Proceedings of the 44th Allerton Conference on Communication, Control, and Computing*, volume 1, pages 771–779, September 2006.

[12] C. Gates. Coordinated Scan Detection. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, 2009.

[13] S. L. Hakimi and S. S. Yau. Distance Matrix of a Graph and its Realizability. *Quart. Appl. Math.*, 22:305–317, 1965.

[14] X. Jin, W.-P. Yiu, S.-H. Chan, and Y. Wang. Network Topology Inference Based on End-to-End Measurements. *IEEE Journal on Selected Areas in Communications*, 24(12):2182–2195, Dec 2006.

[15] M. S. Kang, S. B. Lee, and V. D. Gligor. The Crossfire Attack. In *Proceedings of the 34th IEEE Symposium on Security and Privacy (SP)*, pages 127–141, Washington, DC, USA, 2013. IEEE Computer Society.

[16] J. Ni, H. Xie, S. Tatikonda, and Y. Yang. Efficient and Dynamic Routing Topology Inference From End-to-End Measurements. *IEEE/ACM Transactions on Networking*, 18(1):123–135, Feb 2010.

[17] A. Papadopoulos and Y. Manolopoulos. Structure-based Similarity Search with Graph Histograms. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications*, pages 174–178, 1999.

[18] T. Schuett, A. Reinefeld, F. Schintke, and M. Hoffmann. Gossip-based Topology Inference for Efficient Overlay Mapping on Data Centers. In

Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P), pages 147–150, Sept 2009.

[19] A. Studer and A. Perrig. The Coremelt Attack. In *Proceedings of the 14th European Conference on Research in Computer Security (ESORICS)*, pages 37–52, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] D. Stutzbach and R. Rejaie. Capturing Accurate Snapshots of the Gnutella Network. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 4, pages 2825–2830, March 2005.

[21] A. Wun, A. Cheung, and H.-A. Jacobsen. A Taxonomy for Denial of Service Attacks in Content-based Publish/Subscribe Systems. In *Proceedings of the 2007 Inaugural International ACM Conference on Distributed Event-Based Systems (DEBS)*, pages 116–127, New York, NY, USA, 2007. ACM.