Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 52 (2008) 1019–1039
Computer Networks
www.elsevier.com/locate/comnet

# Building resilient low-diameter peer-to-peer topologies

Rita H. Wouhaybi [a,b,*], Andrew T. Campbell [c]

[a] Department of Electrical Engineering, Columbia University, New York, NY 10027, USA
[b] Department of Electrical Engineering, Columbia University, Intel Corporation, Corporate Technology Group, Hillsboro, OR 97124, USA
[c] Department of Computer Science, Dartmouth College, Hanover, NH03755, USA

Received 11 June 2006; received in revised form 24 October 2007; accepted 26 November 2007
Available online 8 December 2007

Responsible Editor: L.G. Xue

**Abstract**

As more applications rely on underlying peer-to-peer topologies, the need for efficient and resilient infrastructure has become more pressing. A number of important classes of topologies have emerged over the last several years, all with various strengths and weaknesses. For example, the popular structured peer-to-peer topologies based on distributed hash tables (DHTs) offer applications assured performance, but are not resilient to attacks and major disruptions that are likely in the overlay. In contrast, unstructured topologies where nodes create random connections among themselves on-the-fly, are resilient to attacks but can not offer performance assurances because they often create overlays with large diameters, making some nodes practically unreachable. We propose Phenix, a peer-to-peer algorithm for building resilient low-diameter peer-to-peer topologies that can resist different types of organized and targeted malicious behavior. Phenix leverages the strengths of these existing approaches without inheriting their weaknesses and is capable of building topologies of nodes that follow a power-law while being fully distributed requiring no central server, thus, eliminating the possibility of a single point of failure in the system. We present the design and evaluation of the algorithm and show through extensive analysis, simulation, and experimental results obtained from an implementation on the PlanetLab testbed that Phenix is robust to network dynamics such as bootstrapping mechanisms, joins/leaves, node failure and large-scale network attacks, while maintaining low overhead when implemented in an experimental network.


*Keywords:* Peer-to-peer networks; Resilient networks; System design; Simulations; Experimentation with real networks/testbeds

## 1. Introduction

Over the past several years, we have witnessed the rapid growth of peer-to-peer applications and the emergence of overlay infrastructure for Internet, however, many challenges remain as this new field matures. The work presented in this paper addresses the outstanding problem of the construction of

Corresponding author. Address: Intel Corporation, MC: JF2-55, 2111 NE 25th Avenue, Hillsboro, OR 97142-5961, USA . Tel.: +1 503 712 4887.
*E-mail address:* rita.h.wouhaybi@intel.com (R.H. Wouhaybi).

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.11.018

resilient peer-to-peer networks and their efficient performance in terms of faster response time and low-diameter operations for user queries. Low-diameter networks are often desirable because they offer a low average distance between nodes, often in the order of $O(\log N)$. The two classes of peer-to-peer networks, found in the literature, either offer better resilience to node dynamics such as joins/leaves, node failure and service attacks, as in the case of unstructured networks [15,30], or, they offer better performance as in the case of structured networks [26,31,35]. Because of the inherent tradeoffs in the design space of these different classes of peer-to-peer networks, it is difficult to simultaneously offer better performance and resilience without having to reconsider some of the fundamental design choices made to develop these network systems. We take one such alternative approach and propose a peer-to-peer algorithm that delivers both performance and resilience. The proposed algorithm builds a low-diameter resilient peer-to-peer network providing users with a high probability of reaching a large number of nodes in the system even under conditions such as node removal, node failure, and malicious system attacks. The algorithm does not impose structure on the network, rather, the established graph of network connections has the goal of creating some order from the total randomness found in resilient unstructured networks, such as, Gnutella [15] and KaZaA [30].

Unstructured peer-to-peer networks, such as Gnutella, offer no guarantee on the diameter because nodes interconnect in a random manner, resulting in anything other than an efficient topology. These unstructured systems are often criticized for their lack of scalability [27], which can lead to partitions in the network resulting in small islands of interconnected nodes that cannot reach each other. However, these same random connections offer the network a high degree of resiliency where the operation of the resulting network as a whole is tolerable to node removal and failure. In contrast, structured peer-to-peer networks based on distributed hashing tables (DHTs), such as Chord [31] and CAN [26] have been designed to provide a bound on the diameter of the system, and as a result, on the response time for nodes to perform queries. However, these systems impose a relatively rigid structure on the overlay network, which is often the cause of degraded performance during node removals, requiring non-trivial node maintenance. This results in certain vulnerabilities (e.g., weak points) that attackers can target and exploit. Due to the design of DHTs, these structured topologies are also limited in providing applications with the flexibility of generic keyword searches because DHTs rely extensively on hashing the keys associated with objects [2,9].

These observations motivate the work presented in this paper. We propose Phenix, a scale-free algorithm that constructs low-diameter P2P topologies offering fast response times to users. An important attribute of Phenix is its built-in robustness and resilience to network dynamics, such as, operational nodes joining and leaving overlays, node failures, and importantly, malicious large-scale attacks on overlay nodes. The main design goals of Phenix can be summarized as follows: *to construct* low-diameter graphs that result in fast response times for users, where most nodes in the overlay network are within a small number of hops from each other; *to maintain* low-diameter topologies under normal operational conditions where nodes periodically join and leave the network, and under malicious conditions where nodes are systematically attacked and removed from the network; *to implement* support for low-diameter topologies in a fully distributed manner without the need of any central authority that might be a single point of failure, which would inevitably limit the robustness and resilience of peer-to-peer networks; and *to support* connectivity between peer nodes in a general and non-application specific manner so a wide-variety of applications can utilize the network overlay infrastructure. An important property of Phenix is that it constructs topologies based on power-law degree distributions with a built-in mechanism that can achieve a high degree of resilience for the entire network. We show that even in the event of concerted and targeted attacks, nodes in a Phenix network continue to communicate with a low-diameter where they efficiently and promptly rearrange their connectivity with little overall cost and disruption to the operation of the network as a whole. To the best of our knowledge Phenix represents one of the first algorithms that builds resilient low-diameter peer-to-peer topologies specifically targeted toward, and derived from, popular unstructured P2P network architectures, such as, Gnutella [15] and KaZaA [30].

In this paper, we present the design of the Phenix algorithm, first presented in [34], and evaluate its performance using analysis, simulation, and experimentation. We make a number of observations and

show the algorithm's responsiveness to various network dynamics including systematic and targeted attacks on the overlay infrastructure. We implement and evaluate Phenix using the PlanetLab testbed [24]. Experimental results from the testbed implementation quantify the algorithm's overhead and responsiveness to network dynamics for a number of PlanetLab nodes. The paper is structured as follows. We discuss the related work in Section 2 and present the detailed design and operations of Phenix in Section 3. Section 4 presents a detailed evaluation of the algorithm's operation, followed by Section 5, which presents experimental results from the implementation of Phenix on the PlanetLab platform. Finally, we present some concluding remarks in Section 6.

## 2. Related work

Traditionally, low-diameter networks tend to appear in social networks forming small-world topologies [4], while power-law behavior is often seen in many natural systems as well as man-made environments [1,12,16]. These observations led to a body of work related to analyzing and modeling of such networks [4,8,19,20]. The contribution discussed in [7] on preferential attachment has been influential in our thinking. However, the idea of preferential attachment is used in Phenix as a basis to ensure resiliency in a fully distributed, dynamic peer-to-peer environment. The work on peer-to-peer networks presented in [11] makes use of small-world algorithms based on the proposition by Watts and Strogatz [33] on "rewiring" the network. In [11], the idea of rewiring is applied to a Chord [31] overlay. Pandurangan et al. [22,23] create a low-diameter peer-to-peer network but rely heavily on a central server that is needed to coordinate the connections between peers. This proposal creates a potential single point of failure in the overlay network. The authors also do not address the resilience of such a network in the event of targeted node removal, various attacks, or misbehaving nodes. Under such conditions the performance of the network would likely degrade and deviate from the low-diameter design goal.

A family of structured peer-to-peer topologies relying on DHTs, such as Chord [31], CAN [26], and Tapestry [35], has attracted considerable attention in the P2P and overlay community. However, such networks might be limited because they unduly restrict the queries that the users can initiate (e.g., keyword queries) due to the use of hashing tables to store objects at overlay nodes. These networks also couple the application to the underlying infrastructure layer, which makes them attractive to specific applications, but the infrastructure may need to be revised to support changing needs of users. The idea of differentiating the rank of different overlay nodes (e.g., a super node over a regular node) in a peer-to-peer network has been used by a number of systems in order to achieve better performance. For example, KaZaA [30] uses the notion of "supernodes", and Guntella v.0.6 [15] uses "ultrapeers" [32] as supported by the Query Routing Protocol (QRP) [25]. KaZaA creates supernodes among peers by assigning an elevated ranking to nodes with a faster connectivity such as broadband Internet access. However, the implementation details of these popular P2P schemes are not open or published, which makes it difficult to make a comparative statement on the deployed algorithms. Ultrapeers are a standard feature of Gnutella v.0.6, constituting an essential element of QRP, as mentioned above. Ultrapeers differ from what we propose in Phenix in a number of ways. First, ultrapeers act as servers in a hierarchy that is widely known by all other nodes in the network. As a result of this predetermined hierarchy, ultrapeers create a number of vulnerabilities in the system. If ultrapeers were forcefully removed from the network by an attacker, the system would suffer considerably; potentially fragmenting the remaining nodes into disconnected smaller partitions. Another vulnerability arises when malicious nodes assume the role of ultrapeers and mislead other nodes into relying on them for services. An ultrapeer does not use lower level nodes (also called leaves) to relay traffic to other ultrapeers in the network, rather, ultrapeers interact directly with each other. Such reliance could create disconnected groups of nodes in the event that ultrapeers unexpectedly drop out of the network in an uncontrolled manner due to node failure or forceful removal. Each ultrapeer also keeps state information related to the data held by leaf nodes that are connected to it. Creating such a hierarchy that is closely tied to the application level may call for a complete redesign in the event that the application's needs change or new applications need to be efficiently supported. While some recent work [10,13] have proposed solution to creating low-diameter P2P topologies, our work differs as we assume that nodes would not want to re-wire of flip any of their existing connections. In some applications and uses, re-wiring can be acceptable,

however, we assume that the level of trust among nodes is not high enough in order to make allow for changing of neighbor connections.

In our work, we make a distinction between the type of information carried by packets and the routing decisions that are made. RON [6] and i3 [3] have already been designed based on this approach, where a generic topology is proposed that is independent of the application that makes use of it. Such a topology would be an asset for smart search algorithms [2,9] that direct queries instead of flooding the entire neighborhood of the requesting node. Finally, in the context of security, secure peer-to-peer and overlay networks have been proposed as policies to protect individual nodes against denial of service (DOS) attacks in the SOS [18] and Mayday [5] systems, but not in the context of an overall resilient P2P network architecture. Phenix addresses the resilience of the entire network and not the individual nodes.

## 3. Phenix peer-to-peer networks

We build the case for networks whose degree follows a power-law distribution and the resulting benefits especially for a peer-to-peer topology in Section 3.1. In Section 3.2, we describe Phenix, our proposed algorithm, and present an example. We then discuss additional mechanisms that are implemented on nodes in Phenix to maintain the resilience of the network in Section 3.3. We assume that nodes forming a peer-to-peer topology using Phenix will start by interconnecting randomly, but then after an initial small set of nodes, the rest of the nodes joining will start using Phenix to determine their connections to the existing nodes. Since it is important to determine the tradeoffs in the choice of the size of the initial set of nodes, we look at it in Section 3.4.

### 3.1. Power-law properties

The signature of a power-law or a scale-free network lies in its degree distribution, which is of the form presented in the following equation:

$$p(K) \sim K^{-\gamma}. \quad (1)$$

Many networks tend to have an exponent $\gamma$ close to 2, for example, the Internet backbone connectivity distribution is a power law with an exponent $\gamma = 2.2 \pm 0.1$ [12]. As a result of this distribution some nodes are highly connected and can act as hubs for the rest of the nodes. These nodes and their position in the network contribute to a highly desirable characteristic of these graphs: a low "almost constant" diameter, defined as, the average shortest path between two nodes in the graph. This graph is capable of growing while maintaining a low-diameter hence the name scale-free networks. Typically, peer-to-peer networks suffer from a large diameter, which often causes the generation of more network traffic. This is inefficient because it requires nodes to either increase the radius of a search for an object, or opt for a low radius search, which would limit the probability of finding less popular objects in the network. These design trade offs result in increased signaling or degraded performance. In the light of these observations, it seems natural to construct a peer-to-peer topology that conforms to a power-law for its node degree distribution. However, for a proposed algorithm to be feasible, it must adhere to a number of design restrictions. First, the algorithm should be easy to implement and make few assumptions about the underlying network. Despite the problems associated with Gnutella, its deployment is widespread as a result of the simplicity of the underlying protocol [15]. Next, the algorithm should be fully distributed based on local control information, and not include any centralization of control, which might become a bottleneck or a target for attacks. Finally, the algorithm should be robust to node removal whether random or targeted. This means that the network should not be easily partitioned into smaller sub-networks and should be capable of maintaining a high level of resiliency and low-diameter in the face of node removal. The main motivation behind Phenix is to allow nodes in the network to "organically" emerge as special nodes (called preferred nodes) with a degree of connectivity higher than the average, so that a scale-free topology can be formed. In other words, we do not dictate special nodes or hierarchies in advance for the topology to emerge or the network to function. As shown in [7], such networks appear in nature due to preferential attachment, where newcomers tend to prefer connecting to nodes that already have a strong presence characterized by their high degree, and the dynamic nature of such networks involving growth. By examining social networks, we can observe the following; if someone joins a new social network, the first network of "friends" is pretty much random. However, most people, after seeing that a specific person has more acquaintances and is better connected to a larger

number of members in that specific network, tend to acquire a connection to that person in order to gain better visibility. In fact, [7] shows that if a new node has knowledge of the states of all the existing nodes in the network and their interconnections, it can connect to the nodes with the highest degree giving it the highest visibility and putting it in a place where it is a few hops away from the rest of the network. This will guarantee that the resulting network has a degree distribution conforming to a power-law resulting in a low diameter. However, in a peer-to-peer network having such a global view is practically impossible, since most nodes typically can only see a small fraction of the network, and have to make decisions based solely on local information. We present the detail design of the Phenix algorithm in the next section and show the emergence of a power-law topology through simulation and experimental results in Sections 4 and 5, respectively.

### 3.2. Phenix algorithm design

In what follows, we describe the Phenix algorithm for the simple case where nodes join the network. A node obtains a list of addresses using a rendezvous mechanism by either contacting a host cache server [14] or consulting its own cache from a previous session in a fashion similar to an initial connection, as described in Guntella v0.6 [15]. However, instead of establishing connections to "live" nodes from the returned list, the joining node divides these addresses into two subsets, as expressed in Eq. (2): that is, random neighbors and *friends* that will be contacted in the next step. The reason for this division is to create few random connections through $G_{\text{random},i}$, but then use the second subset $G_{\text{friends},i}$ in order to query them and create preferred attachments as we explain next.

$$G_{\text{host},i} = [G_{\text{random},i}, G_{\text{friends},i}]. \tag{2}$$

Then $i$ initiates a request called a "ping message" to the nodes in the list $G_{\text{friends},i}$, sending a message of the form:

$$M_0 = \langle source = i,\ type = ping,\ \text{TTL} = 1,\ hops = 0\rangle. \tag{3}$$

Each recipient node constructs a "pong message" as a reply containing the list of its own neighbors, increments the hops counter, decrements the TTL, and forwards a new ping message to its own neighbors, as follows: $M_0 = \langle source = i,\ type = ping,\ \text{TTL} = 0,\ hops = 1\rangle$. Each node $j$ receiving such a message will send no pong message in reply, but instead add the node $i$ to a special list called $\Gamma_j$ for a period of time denoted by $\tau$; this is a essential step in order to disable crawling of the network as a possible attack mechanism by malicious nodes.

Following this procedure, the node $i$ obtains a new list of all the neighbors of nodes contained in $G_{\text{friends},i}$ and adds to a new list denoted by $G_{\text{candidates},i}$. Then $i$ sorts this new set of nodes using the frequency of appearance in descending order, and uses the topmost nodes to create a new set that we denote as $G_{\text{preferred},i}$, where $G_{\text{preferred},i} \subseteq G_{\text{candidates},i}$. Basically, this step is a popularity contest where node $i$ is trying to determine the most popular nodes using the obtained friends lists. Thus, the resulting set of neighbors to which $i$ creates connections is $G_i = [G_{\text{random},i}, G_{\text{preferred},i}]$.

Node $i$ opens a servent (server–client) connection to a node $m$ ($m$ is in the list $G_{\text{preferred},i}$) where the word servent is a term denoting a peer-to-peer node, which is typically a server and a client at the same time as it accepts connections as well as initiates them. Then node $m$ checks whether $i$ is in its $\Gamma_m$ list, and if this is the case, increments an internal counter $c_m$ and compares it against a constant $\gamma$. If $c_m \geqslant \gamma$, then $c_m = c_m - \gamma$, a connection is created to node $i$, which we call a "backward connection", and the set of neighbors added as backward edges is updated, as follows: $G_{\text{backward},m} = G_{\text{backward}_m} \bigcup \{i\}$. This backward connection creates an undirected edge between the two nodes $i$ and $m$ ($i \leftrightarrow m$) from the initial directed edge, as $i \rightarrow m$. In addition, $\gamma$ ensures that a node does not add more connections than $d_{in,m}/\gamma$ where $d_{in,m}$ is the in-degree for node $m$, or the number of its incoming connections. The intuition behind this step is to create selective connections from the preferred nodes to the nodes connecting to them, and we achieve this by adding such one connection for every $\gamma$ new nodes connecting to the preferred node.

When node $i$ receives a backward connection from node $m$ it will consider its choice of node $m$ as a good one, and accordingly update its neighbors lists: $G_{\text{preferred},i} = G_{\text{preferred},i} - \{m\}$, and $G_{\text{highly\_preferred},i} = G_{\text{highly\_preferred},i} + \{m\}$. The final list of neighbors for node $i$ is:

$$G_i = [G_{\text{random},i}, G_{\text{preferred},i}, G_{\text{highly\_preferred},i}, G_{\text{backward},i}].$$

The pseudocode for node connections in Phenix is presented in Fig. 1, and an example of the creation of $G_i$ is presented in Fig. 2, for illustration purposes.

In this particular scenario, the existing overlay network is shown in Fig. 2 where the interconnections between nodes are shown with arrows, with the bold arrows representing connections that were created by preferential and backward formation. In the scenario, node 8, wants to join the network and goes through the process shown in Fig. 2. Node 8 starts by obtaining a list of hosts that are present in the network and then divides this list into two sub-lists where $G_{\text{random}} = [1,3]$ and $G_{\text{friends}} = [5,6]$. Then it contacts the nodes contained in $G_{\text{friends}}$ to obtain their lists of neighbors and constructs the following list $G_{\text{candidates}} = [7,2,4,7]$. Sorting the nodes in descending order using their frequency of appearance yields $G_{\text{preferred}} = [7,2]$. Then node 8 constructs the final list $G = G_{\text{preferred}} \bigcup G_{\text{random}} = [7,2,1,3]$ and connects to these nodes. Note, that as node 8 starts its servent sessions with the resulting nodes in $G$ then one or more of them might choose to create a backward connection to node 8 depending on the values of their respective counters $c$.

### 3.3. Network resiliency

According to the Webster Dictionary [21], the word resilience is defined as "an ability to recover from or adjust easily to misfortune or change". Networks with power-law degree distributions are often criticized in the literature for collapsing under targeted attacks. Under such conditions if a small fraction of the nodes with high degrees is removed from the network then the whole network suffers and often becomes disconnected into smaller partitioned fragments, also referred to as "islands" in the literature [7].

Phenix attempts to make connections resilient, protecting the well being of the entire network. We achieve this goal by following a set of guidelines that can be summarized, as follows. First, we attempt to hide the identity of highly connected nodes as much as possible, making the task of obtaining a comprehensive list that contains these nodes practically impossible. The second deterrent deals with neighbor updates, or what we call "node maintenance" (discussed below), where a network under attack can recover when existing nodes rearrange their connections and maintain connectivity.

Note, that we assume that an attacker is powerful enough to force a node to drop out of the network, whether by denial of service attacks or by any other mechanism available, once an attacker acquires the IP address of such a node. In Phenix networks, resiliency implicitly means: the resilience of the whole network consisting of all "live" nodes where their

---

**connect_to_network**($i$) {
  obtain a list of existing nodes $G_{host}$ from web cache;
  divide $G_{host}$ into $G_{random}$ and $G_{friends}$;
  **connect_preferrential**($i$, $G_{friends}$);
  **connect_random**($i$, $G_{random}$); }
**connect_random**($i$, $G$) {
  open connections from $i$ to all nodes in $G$;}
**connect_preferrential**($i$, $G$) {
  let $s$ be the size of $G$;
  $G_{candidates} = \emptyset$;
  **for** ($x = 0; x < s; x++$) {
    $k = G[x]$;
    send $M_0$; where $M_0 = ping\langle i, k, 1, 0\rangle$
    node $k$ sends back the list of its neighbors: $G[k]$;
    $G_{candidates} = G_{candidates} \cup G[k]$; }
  sort $G_{candidates}$ by frequency of appearance and make its elements unique;
  $G = G_{candidates}[0..(s-1)]$;
  node $i$ connects to all nodes in $G[i] = G_{random} \cup G_{preferred}$;
  **if** (( $j$ connects back to $i$) && ($j \in G_{preferred}$)) { // flag $j$ as highly preferred
    $G_{preferred} = G_{preferred} - \{j\}$;
    $G_{highly\_preferred} = G_{highly\_preferred} + \{j\}$; }

---

Fig. 1. Pseudocode for Phenix.

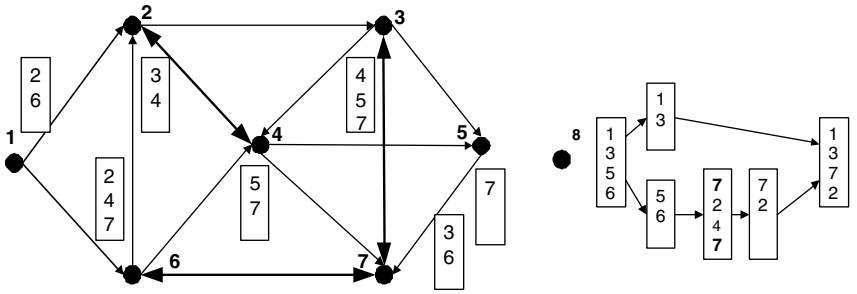



Fig. 2. Example of Phenix overlay construction.

connections form edges in a graph that is as close to a strongly connected graph as is possible, as we will show in Section 4.

### 3.3.1. Hiding node identities

In order to limit the likelihood of a malicious user obtaining a global view of the whole overlay graph (formed by the live nodes) of the network, Phenix supports three important mechanisms. First, a node receiving a ping message $M_0$ will respond with a pong message, and forward a ping message $M_1$ to its neighbors. All nodes receiving $M_1$ will add the originator to a list denoted by $\Gamma_i$. This list supports the notion of either "temporary blocking" or "black listing", where if the same originating node sends a ping message with the intent of "crawling" the network to capture global or partial graph state information, such a message will be silently dropped with no answer/response sent back to the originating node. Black lists can be shared with higher layer protocols to isolate such malicious practices and can serve to isolate such nodes. A mechanism that detects a node crawling the network and silently discards queries will not stop a malicious user, but rather, slow its progress because the malicious node needs to obtain a new node ID (e.g., this would be similar to the Gnutella ID) to continue the crawl of the overlay, or wait for enough time for nodes to purge their black lists $\Gamma_i$. Peer-to-peer networks such as Guntella [15] have proposed including the MAC address as part of the node ID, making it even more difficult for an attacker to obtain a new and distinctly different node ID at a rate fast enough to continue the crawl. It is worth noting that if joins/leaves of an overlay network are dynamic enough then crawling at slower time scales will not yield an accurate view of the network state and topology. Even though such a scheme helps limit the impact that malicious nodes can have, it still does not fully eradicate potential attacks on the network.

Next, Phenix also employs the policy of silently dropping any ping message, similar to the one shown in Eq. (3), whose TTL value is greater than 1. A non-conforming node with malicious intent might generate such a message. Nodes drop these messages without responding to the originator or forwarding such a message to neighbors. This has the effect of eliminating crawling even if the originating node is not on the list $\Gamma_i$ of the receiving node, in contrast to Gnutella where crawling is often practiced.

Third, a node that establishes backward connections to other nodes in the network will not return these connections when it receives a ping in any of its pong reply messages. This policy is not meant to protect the node's $G_{\text{backward}}$ sub-list of neighbors. Rather, it protects the identity of the node itself and any possible preferential status that the node may have, from an attacking node. If an attacker were to receive a long neighbors list from a node, it can infer that such a node is a highly connected node from the size of its neighbors' list. Thus, a node will only return the subset $G_{\text{outside\_world}}$ defined by Eq. (4) in a pong message. In this case, this node does not need to forward $M_1$ to all of its neighbors. Rather, it only forwards $M_1$ to nodes in its $G_{\text{outside\_world}}$ subset since these are the nodes that might risk exposure to an attacker, where,

$$G_{\text{outside\_world}} = [G_{\text{random}}, G_{\text{preferred}}, G_{\text{highly\_preferred}}]. \quad (4)$$

### 3.3.2. Node maintenance mechanism

In the event of an attack, the network needs to be responsive and able to rearrange connectivity in order to maintain strong connections between its nodes. In what follows, we propose a state probing mechanism that makes Phenix responsive to failed

nodes or nodes that drop out of the overlay because of attacks. The number of neighbors of a node $i$, represented by $h_i$, is defined as the summation of the number of neighbors obtained through random, preferred and backward attachments; in other words, the out-degree of the node defined as the total number of outgoing connection for a node $i$. This total number is expressed as $h_i = h_i^{\mathrm{r}} + h_i^{\mathrm{p}} + h_i^{\mathrm{b}}$, where $h_i^{\mathrm{b}} = 0$, if $i \notin$ [preferred nodes]. $h_i^{\mathrm{r}}$, $h_i^{\mathrm{p}}$, and $h_i^{\mathrm{b}}$ represent the number of random, preferential (standard and highly), and backward neighbors, respectively. Nodes examine their neighbors' table in order to make sure that they are not disconnected from the network due to node departures, failures, or denial of service attacks. If the following Inequality $h_i^{\mathrm{r}} + h_i^{\mathrm{p}} <$ threshold is satisfied, signaling a drop below a pre-defined threshold, then node $i$ runs a node maintenance procedure, as described below. During node maintenance, nodes are trying to keep track of how many random neighbors they have versus preferred neighbors. When they notice a drop in preferred nodes, they launch an aggressive algorithm acquiring more preferred nodes than random ones. This is particularly important when the network is under targeted attacks aiming at highly connected nodes, and encourages the rapid promotion of existing nodes to become preferred ones as we will show in Section 4. When the network goes back to a stable state, where preferred nodes are leaving the network gracefully, nodes will slowly change their aggressive strategy and start shifting the ratio of their neighbors to its initial value. What follows next is a formal presentation of this node maintenance algorithm.

If a node on the $i$'s neighbors' list leaves the network gracefully, then it informs all the nodes connecting to it by closing the connections. However, if a node is forcefully removed or fails then node $i$ will be informed of this fact only through probing where a message is sent to its neighbors, as follows: $M_2 = \langle source = i,\ type = ping,\ \mathrm{TTL} = 0,\ hops = 0 \rangle$. In the case where no answer is received after a timeout (which is discussed in Section 5) then the neighboring node is declared down. The number of neighbors before node maintenance can be expressed as follows: $h_i^-(t_n) = h_i(t_{n-1}) - d_i^{\mathrm{r}}(t_n) - d_i^{\mathrm{p}}(t_n) - d_i^{\mathrm{b}}(t_n)$, where, $h_i^-(t_n)$: current number of nodes (prior to the last maintenance run), and $d_i^{\mathrm{r}}(t_n)$, $d_i^{\mathrm{p}}(t_n)$, $d_i^{\mathrm{b}}(t_n)$: the number of neighbors (random, preferential, and backward, respectively) lost since the last node maintenance. Following the node maintenance, we have:

$$h_i(t_n) = \begin{cases} h_i^-(t_n), & \text{threshold} < h_i^-(t_n) - h_i^{\mathrm{b}}(t_n) \leqslant \max, \\ h_i^-(t_n) + u_i^{\mathrm{p}}(t_n) + u_i^{\mathrm{r}}(t_n), & \text{otherwise}, \end{cases} \tag{5}$$

where, $h_i(t_n)$: the number of neighbors after the node maintenance and $u_i^{\mathrm{p}}(t_n)$, $u_i^{\mathrm{r}}(t_n)$: the number of new neighbors added preferentially and randomly, respectively. The ratio of preferential and random neighbors for a node $i$ is presented in the following equation:

$$\alpha_i(t_n) = \frac{h_i^{\mathrm{r}}(t_n)}{h_i^{\mathrm{p}}(t_n)} \text{ and } \frac{h_i^{\mathrm{r}}(t_n)}{\max - h_i^{\mathrm{r}}(t_n)} \leqslant \alpha_i(t_n) \leqslant 1,\ \forall i, n \tag{6}$$

and the initial value of $\alpha$ is expressed by: $\alpha_i(t_0) = 1,\ \forall i$.

The update of neighbors is then performed according to the following equation:

$$u_i^{\mathrm{r}}(t_n) = d_i^{\mathrm{r}}(t_n) \text{ and } u_i^{\mathrm{p}}(t_n) = \begin{cases} \left\lceil \frac{\tau_i(t_n) - \mu^{\mathrm{p}}}{\alpha_i(t_{n-1})} \right\rceil, & d_i^{\mathrm{p}}(t_n) > 0, \\ 0, & d_i^{\mathrm{p}}(t_n) = 0, \end{cases} \tag{7}$$

where $\tau_i(t_n) = \sum_{k=n-l+1} d_i^{\mathrm{p}}(t_k)/l$. $\tau_i^{\mathrm{r}}(t_n)$ is the average number of preferential neighbors that dropped out over the last $l$ node maintenance cycles, measured at time $t_n$, $mu^{\mathrm{p}}$ is the expected value of the number of neighbors that disappeared in one node maintenance cycle. The symbol $\lceil\ \rceil$ rounds up the value to the next highest integer. Therefore, the final number of neighbors is

$$h_i^{\mathrm{p}}(t_n) = \begin{cases} h_i^{\mathrm{p}}(t_0), & u_i^{\mathrm{p}}(t_n) < h_i^{\mathrm{p}}(t_0) - h_i^{-\mathrm{p}}(t_n), \\ h_i^{-\mathrm{p}}(t_n) + u_i^{\mathrm{p}}(t_n), & u_i^{\mathrm{p}}(t_n) < \max - h_i^{-\mathrm{p}}(t_n) - h_i^{\mathrm{r}}(t_n), \\ \max - h_i^{\mathrm{r}}(t_n), & \text{otherwise}. \end{cases} \tag{8}$$

For preferred nodes, we already have the following approximation: $h_i^{\mathrm{b}} = \lceil \frac{n_i - \gamma}{\gamma} \rceil$, where $n_i$ is the number of nodes pointing to node $i$. The preferred node updates its $c_i$ counter, as follows: $c_i = c_i + (\gamma \times d_i^{\mathrm{b}}(t_n))$, while no nodes are added in the backward set during the node maintenance process. Analysis of the effect of $\alpha$ on the network's behavior, particularly when faced with large-scale attacks is discussed in Section 4.

### 3.4. Preferential nodes

In this section, we look at the effect of starting our peer-to-peer network with a subset of node inter-connecting randomly (without implementing

any preferential attachments). After the number of nodes reaches a certain threshold, any incoming node will uses Phenix in order to determine the list of its neighbors. In what follows, we analyze the emergence of nodes with a degree deviating from that of the average of the network. We call such nodes preferred nodes.

Let us assume that we initially have a network of $N$ nodes interconnected randomly. A new node $i$, running the Phenix algorithm wishes to connect to this network. So, $i$ acquires a list of friends using a rendezvous or bootstrapping mechanism similar to the one used by many P2P systems. As described earlier, node $i$ contacts these friends asking for their respective lists of neighbors. The summation of all answers constitutes the list of candidates. It follows that after node $i$ acquires the list of $G_{\text{candidates},i}$, the probability of connecting to a node on the list is directly proportional to the frequency of appearance of that node; that is to say, it is equal to the probability that a node will appear more than once in its list of candidates.

Let, $\mu$ be the average number of neighbors and $N$ the number of nodes in the network. A new node $i$ will connect to $\mu/2$ nodes randomly in $G_{\text{random},i}$, since $\alpha_i(t_0) = 1, \forall i$, and will contact $\mu/2$ nodes requesting a list of their neighbors, which will become $G_{\text{candidates},i}$. Thus, the resulting number of nodes on this latter list is an average of $\mu^2/2$. Since we are interested in nodes appearing more than once on this list (which translates to a higher probability in initiating a connection to one of them), we calculate the probability of a node $j$ appearing at least twice, which is expressed as the summation of the probabilities that $j$ appears $2, 3, \ldots, m$ times, where $m = \mu/2$. This upper bound of $m$ comes from the fact that a node can appear at most once in each list returned by one node of the sub-list $G_{\text{candidates},i}$. Thus the probability of a node appearing twice becomes the probability that it is on two of the lists of nodes in $G_{\text{candidates},i}$, and similarly, three appearances signifies the presence on three lists, and so on until $m$. Therefore, the probability that a node appears at least twice, encouraging a preferential attachment in a Phenix setup is given by the following equation:

$$P(X > 2) = \sum_{i=2}^{m} P(X = i) = \binom{m}{2}\left(\frac{\mu}{N}\right)^2\left(1 - \frac{\mu}{N}\right)^{m-2} + \cdots + \left(\frac{\mu}{N}\right)^m. \quad (9)$$

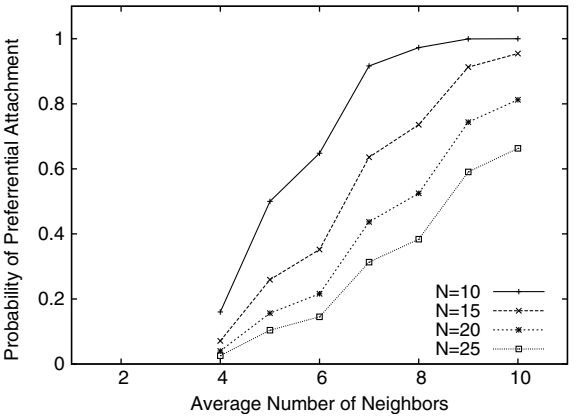



Fig. 3. Probability that a preferred node appears.

Now that we know the value of the probability of a preferential attachment, we are interested in analyzing how fast such an attachment will take place (as the network grows) assuring the evolution of the network graph from a random network to one based on power-laws. Fig. 3 plots the probability derived in Eq. (9) versus the average number of neighbors for different values of $N$, the initial random network. We can observe that it is desirable for the initial network to be small so that preferential attachments start to form as early as possible; for example, given an initial Phenix network of 20 nodes, and 5 neighbors on average the probability of preferential attachment is around 0.1562. This means that with the seventh node joining the network, at least one preferential attachment is formed. It follows that after one preferential attachment forms, the probability of a second preferential attachment increases since the probability of this node appearing more than the others is already biased. Note that N is not the total number of nodes in the final overlay, but only the first initial nodes that come together in the network. Clearly, the overlay network can grow to encompass a much larger number of nodes, and at that time Eq. (4) no longer holds because the connections among nodes is not random, but biased, forming a power-law, as we have just shown in this section.

## 4. Simulation

In what follows, we discuss the results obtained from implementing the Phenix algorithm in a simulation environment based on Java software. We start by examining the emergence of a power-law where nodes enjoy a low-diameter. We then study

different types of attacks on an overlay network using the Phenix algorithm to measure the network's degree of resilience. Finally, we discuss the sensitivity of Phenix to different bootstrapping mechanisms.

### 4.1. Power-law analysis

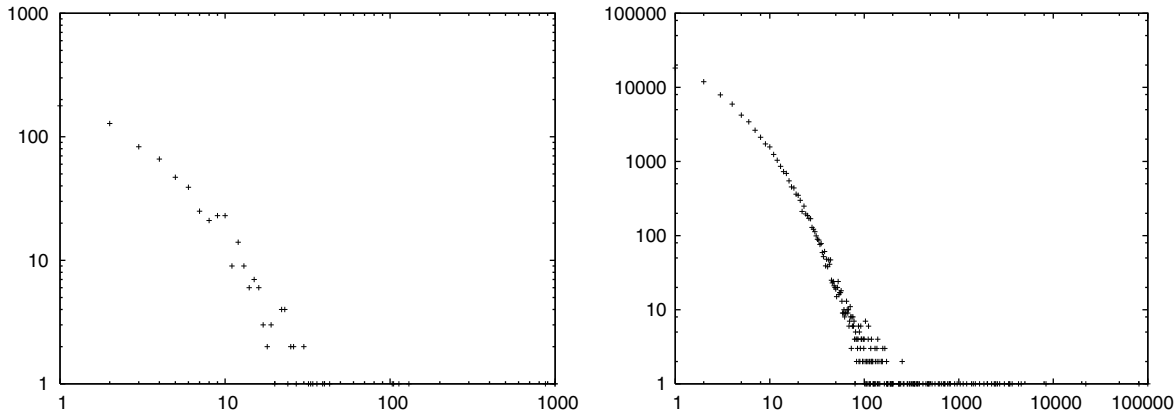
Degree distributions following power-laws tend to appear in very large networks found in nature [7,8]. However, we would like to have an algorithm where such a distribution will be present in networks of modest size. Such an algorithm might be useful in different situations for various applications where an assurance of a large number of nodes might not be feasible. We studied the effect of creating a network of pure joins in order to be guaranteed of the emergence of a power-law in such a simple scenario. The nodes join the network following a normal distribution at simulation intervals, by acquiring neighbors' connections based on the Phenix algorithm. Plotting the degree distribution for the resulting network of a 1000-node on a log–log scale shows a power-law emerging in Fig. 4a. This property is more clearly observed for a network of 100,000 nodes, as observed in Fig. 4b. We further look at the changes in degree distribution and how it affects the power-law in Section 4.3.

The remainder of this discussion shows the low-diameter of the network rather than the degree distribution, since nodes are often more interested in how many nodes they can reach with a small number of hops. In fact, the simulation results show the signature of a scale-free network where the diameter does not increase proportionally to the number of nodes, but instead follows the logarithm of the nodes in the network.

### 4.2. Attack analysis

Next, we study more sophisticated networks where nodes join and leave the network using different scenarios. The attacks analyzed in this section are aggressive and to some extent extreme requiring additions of nodes to the network that probably would not be typical of an attacker in a practical network. However, we chose to include such an analysis in order to test the limit at which the Phenix algorithm is capable of adapting, and the point beyond which the network does not serve its purpose anymore of interconnecting participants to each other.

We consider a number of attack scenarios where an attacker can perform one of three different types of distinct attacks on the network, or a combination of such attack scenarios. The first attack scenario consists of a user that acquires host cache information like a legitimate node might. The attacker contacts these acquired nodes with a $M_0$ message, getting the respective lists of their neighbors, and building his candidate's list, as a result. However, once the attacker has this information it will then attack the nodes appearing in this list more than once, removing them from the network. Such an attacker is limited in its capabilities and resources when compared to the two other scenarios discussed next, because the attacker attempts to target nodes that might have a node degree higher than the aver-




(a) Degree Distribution for 1,000 Nodes (b) Degree Distribution for 100,000 Nodes

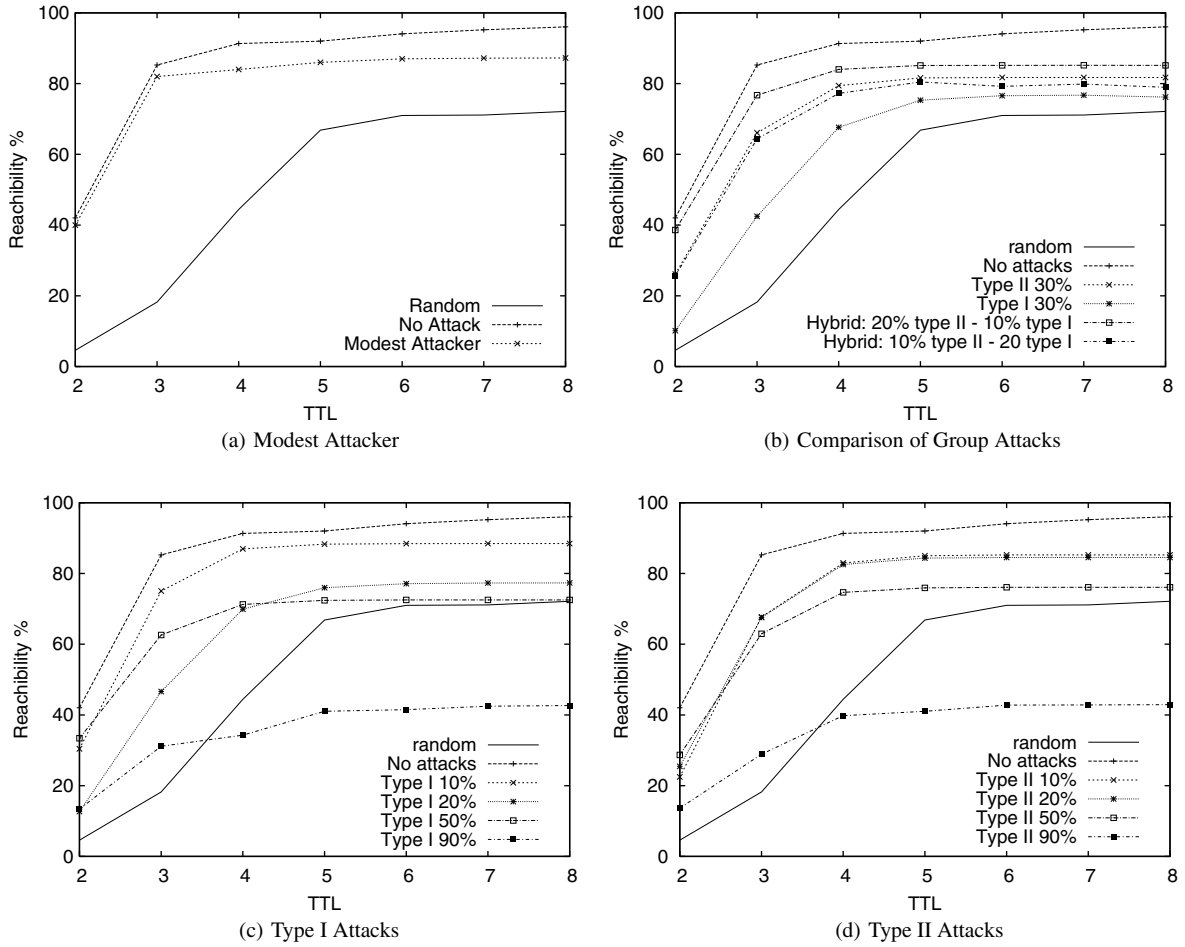
Fig. 4. The degree distribution of a network of nodes running Phenix on a log–log scale.

age without participating in the overall structure. However, such an attacker has a level of sophistication because it is not removing nodes randomly. Rather, the attacker attempts to cause as much disruption as possible by maximizing the damage to the network in creating targeted attacks toward nodes that are important to the network performance, with as little investment as possible. The other two types of attacks are more organized from the attacker's perspective and require adding a large number of nodes to the network. Such an attack option is possible due to the fact that the network is open and welcomes any connection with no prior authentication or authorization. The first of these two additional attacks we denote as a "Group Type I" attack. This attack requires an attacker to add a number of nodes to the network that only point to each other, thus, increasing the probability that they will emerge as preferred nodes in the overlay network. The last type of attack, which we denote as a "Group Type II" attack, consists of adding a number of nodes to the network that would behave like normal nodes do. These last two types of attacks attempt to create anomalies in the network by introducing "false" nodes that remain connected for a prolonged period of time. Such a regime would ensure that other "true" nodes come to rely on these false malicious nodes due to the length of time that the false nodes are available in the network. Under such attack scenarios, these false nodes suddenly disconnect from the overlay network all at the same time with the intention of disconnecting and fragmenting the network into small islands of nodes. We also consider a hybrid attack scenario where the strategy dictates that some of the malicious nodes use the strategy of "Group Type I" and the others use "Group Type II" attacks.

The following simulation results are for an overlay network composed of 2000 nodes. Each node chooses a number of neighbors between 5 and 8, which represents small numbers of nodes, if compared to Gnutella [15], denoted, respectively, by min and max, with equal probability while maintaining $\alpha_i(t_0) \leqslant 1, \forall i$, resulting in an average of $E(\alpha_i(t_0)) = 41/48$ for the whole network. However, this initial state for $\alpha$ will change as nodes join and, most importantly, leave the network, as we will discuss later. At each simulation time interval, the number of nodes joining the network is based on a normal distribution. For the case of nodes leaving the network, we consider three different cases: (i) the departure pattern is based on a normal distribution with a mean $\lambda$ where nodes leaving are randomly selected from the overlay network. This scenario is equivalent to the case where the system faces no attacks, as shown in Fig. 5; (ii) the departure pattern is based on a normal distribution, however, the nodes are removed by sending ping messages creating a sorted list of candidates, and removing preferred nodes from the network (this corresponds to the "modest attacker"); and (iii) represents group attacks as in the case of Group Type I, Group Type II, and hybrid of Group Type I/Group Type II attacks. In this case, a percentage of the nodes (note that different values of this percentage are studied extensively later in this section) represent malicious nodes that conspire together to create the maximum possible damage to the whole structure of the network. The attack proceeds by having nodes at each interval leave the system as if there is no attack scenario until the malicious nodes suddenly drop out of the system, as described earlier. In each case of nodes leaving the system, we compare the performance of the network with a pure random network having the same average number of neighbors across all nodes, taking into consideration the min, max values, and backward connectivity from preferred nodes in a fashion similar to a topology created in the Gnutella network [15].

In all simulations, we start with a small number of nodes $n_{\text{init}} = 20$ that are interconnected randomly to each other with each node maintaining a number of neighbors $\min \leqslant h_i \leqslant \max$. The average rate of nodes arriving (i.e., issuing joins) is greater than the average departure rate, allowing the network to grow to the total number of nodes we would like to examine. In the case of Type I, Type II or hybrid group attacks, the process with which the network is formed starts by adding 50% of the legitimate or "true" nodes in incremental steps. At each step, the number of nodes added is drawn from a normal distribution, in a fashion similar to what would happen in a real P2P network. Following this, the malicious nodes are introduced in a single step giving them enough time to establish a strong presence in the network. We then add the next 50% of the legitimate nodes also in incremental steps. During all the steps, nodes continue to leave the network under a "no attack" situation. Eventually, we remove the malicious nodes, and study the effect on the remaining live nodes.

The metric measured for these networks consists of the percentage of unique reachable nodes in the




(a) Modest Attacker

(b) Comparison of Group Attacks

(c) Type I Attacks

(d) Type II Attacks

Fig. 5. Analysis of attacks on nodes running Phenix.

network versus the number of hops that we also denote by TTL. This measurement will give us an understanding of how many nodes can be reached when an application issues a query on top of the Phenix topology. Also note, that the same can be denoted as a radius because it starts with a node as the center and proceeds to try to cover as much of the network as possible. The figures represent this reachability metric in terms of the percentage of the total number of "live" nodes in the network. We compare the Phenix network under attack to a purely random network (as implemented by the Gnutella v0.6 [15]) because a random topology network is often cited to be the most tolerable to attacks [8]. Also, it is worth noting that the response of the network to various attacks is shown before the nodes run their node maintenance procedure (as described in Section 3.3.2) because the performance of a Phenix network will return back to the case of "no attacks" after a single neighbors maintenance is performed on each node.

Each experiment ran 10 times to ensure that the results stem from the structure and properties of the Phenix algorithm. We then sampled 10% of the nodes and measured the reachability of each of the sampled nodes and calculated the averages for each result. All measurements deviated only a little from the averages presented, proving that the behavior of the distributed algorithm is indeed predictable and reliable.

Fig. 5a shows a comparison of the performance for the first type of targeted attack discussed above, which we denote on the plot as the "modest attacker", versus the "no attack" and random network. We can see that in response to the targeted node removals, the performance of the network degrades but the loss is quite tolerable and still offers a gain over the random topology. Thus, in

this scenario, Phenix has the potential of offering the participating nodes a more efficient overall performance where a node can be reached even with a smaller TTL value.

Fig. 5b shows four different attacks: 30% of both Group Type I and Group Type II attacks, and two hybrid combinations each resulting in a total of 30% malicious nodes in the overlay. In studying such a comparison we were interested in seeing which strategy might be more damaging in fragmenting the network and disconnecting the live nodes. We observed that Group Type I attacks create a larger fragments in the network when introduced as a small percentage, than the same number of nodes running in the Group Type II attack mode. In addition, when we have a smaller percentage of Group Type I nodes backed up by more nodes as Group Type II, the performance of the network degrades the most as the maximum number of nodes reachable drops, as shown in Fig. 5b. This is due to the fact that nodes in Group Type I attacks, point to each other, which means that if we increase their number beyond a certain threshold the probability that they will be chosen by legitimate users as preferential drops. However, Fig. 5b also shows us that across all attack scenarios, the network does not collapse into small islands. A promising result shows the giant component, indicated by the maximum reachability, not dropping below 70% of the remaining "live" nodes under all attack conditions.

Fig. 5c and d show the effect of Group Type I and Group Type II attacks on a Phenix network where the percentage of malicious nodes shown is actually the percentage from the final network. This means that if we have 10% malicious nodes in a 2000-node network then the number of legitimate nodes is 1800. This result implies that for an attacker to launch a 50% attack, he/she has to have the capability of introducing a number of malicious equal to the number of existing nodes in the network that he/she wishes to partition or harm.

In Figs. 5c and d, we can observe that a network under an attack of 50% malicious nodes scenario seems to provide a performance that is better than the 20% malicious nodes attack. This result seems counter-intuitive at first. However, it occurs because the number of nodes in the network becomes half the initial size, as the other half were malicious nodes that dropped out of the network, while the measured reachability is represented as a percentage of the total number of live nodes. Similarly, a network undergoing a 90% malicious node attack seems to reach a constant plateau with a lower TTL value than the initial network for the no attacks scenario, as shown in the figure. This is due to the fact that the structure of the network carries the signature of a power-law like distribution, offering a diameter in the order of $O(\log N)$ where $N$ is the total number of nodes participating in the network. As $N$ drops to 10% of its initial size, the diameter follows by decreasing as well.

Measuring the giant component, which is the largest portion of the network that remains strongly connected, under different group attack scenarios is shown in Fig. 7. If we consider, for example, the 20% attack for both Group Type I and Group Type II modes, we can observe that the giant component still amounts to around 80% of the total nodes of the network. At the same time, an 80% attack results in a giant component composed of 60% of the nodes. One can conclude that in order for a malicious attacker to divide a network of 400 nodes into half, then as many as 1600 nodes have to be introduced into the network for a considerable amount of time. This is a high price to pay to break such a network in two parts as the attacker is adding a number of nodes equal to 400% of the number of nodes in the initial targeted network. Add to this that the network recovers to a giant component in the order of 90% of the total number of nodes after performing one node maintenance interaction. This result looks very promising in terms of Phenix's ability to respond to such attacks.

We ran the same set of simulations where the total number of nodes is 20,000 instead of the 2000 keeping all other parameters identical. In Fig. 6, we present a summary for the hybrid attack

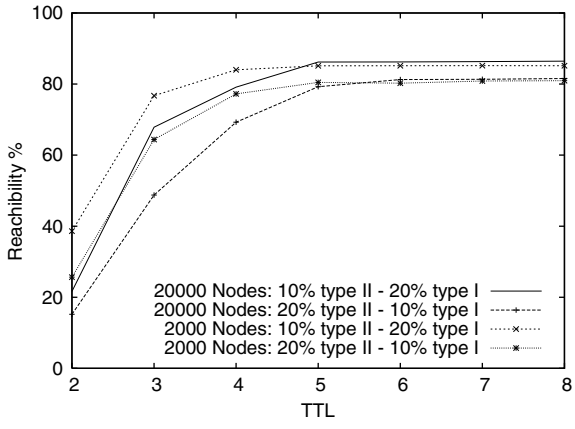



Fig. 6. Hybrid attacks in 2000 and 20,000-node networks.

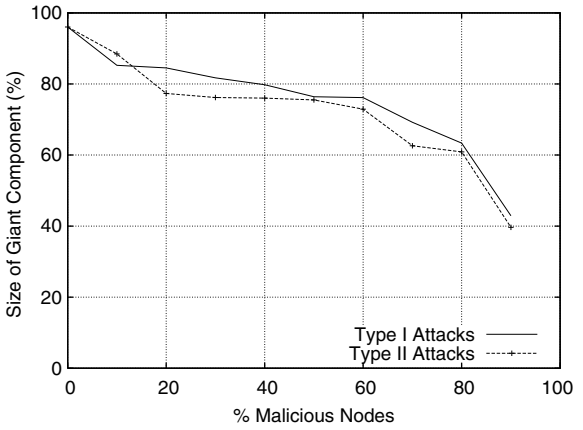



Fig. 7. Giant component.

discussed earlier. The behavior is very similar to that of the previous set of experiments showing that Phenix can provide a high degree of resiliency to the network independent of the total number of nodes in the network. Fig. 6 also shows another signature of a power-law like distribution. A 20,000 node network reaches almost a stable plateau with a TTL larger by 1 hop than the 2000-node network, even though the total number of nodes is 10 times greater.

These plots indicate that increasing the TTL beyond a certain limit does not provide any significant benefit, as can be seen in Figs. 5b and 6. In fact, the number of reachable nodes seems to reach a maximum value beyond which increasing the TTL does not offer a wider variety of nodes reached. For example, it can be seen from Fig. 5c that increasing the TTL from 4 to 5 in a 2000-node network with 10% malicious nodes of Group Type I will increase the reachability from 88.29% to 88.44%. This is a characteristic that can be exploited by applications where a query carrying a large TTL might have its hop decremented by more than 1 at a node receiving it because the gain of a larger TTL is not that significant. Such structure is beneficial in the sense that a reply can be returned to the originating node in a faster period of time because the number of hops is smaller than the random counterpart. An application sitting on top of such a topology might consider not to flood all of its neighbors limiting the generated traffic. Rather, it can direct the search using a smart policy such as GIA [9], for example.

The $\alpha$ parameter introduced in Section 3.3.2 contributes to a fast recovery because most nodes will become quite aggressive in creating highly connected nodes after losing their preferred neighbors. This encourages the promotion of existing nodes to become highly connected nodes and assume the role of preferred nodes. We show the behavior of $\alpha$ in Fig. 8. In this experiment, we use a hybrid attack of 10% Group Type I and 20% Group Type II. We can observe in Fig. 8, that the initial value of the average of $\alpha$ across the entire network is close to 0.7 before introducing malicious nodes. However, when these nodes are added to the network (at time = 60), they create a false sense of stability that can be seen in an increase and almost constant $\alpha$ despite the normal operation of the rest of the network where nodes are joining and leaving. Following the disappearance of the malicious nodes (at time = 180), we observe a sudden drop in $\alpha$ across the entire network, as a sudden change is experienced by most legitimate live nodes. However, as the network goes back to normal operations, $\alpha$ starts to increase again, indicating that the network is in a stable state again. The choice of the $\alpha$ update influenced by Eq. (7) ensures aggressiveness in decreasing it in order to respond as fast as possible to an attack, while the process of increasing it again is more conservative. We assumed any node can handle any traffic offered to it in the work presented, however, in practice this might not be the case and some nodes might refuse to have a higher in-degree than the average.

### 4.3. Sensitivity to bootstrapping mechanisms

In this section, we test the sensitivity of the Phenix algorithm to the use of different bootstrapping

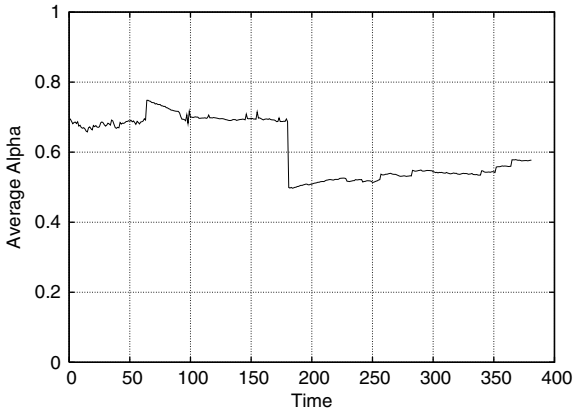


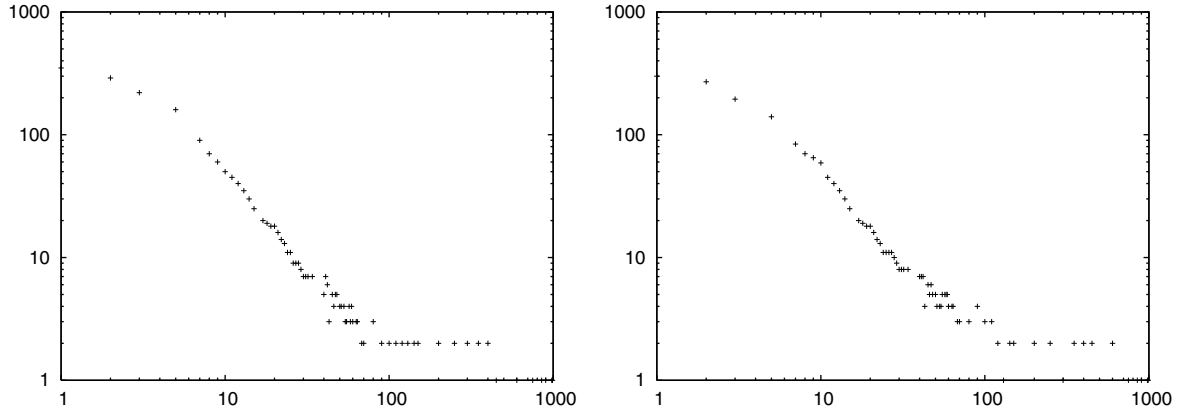

Fig. 8. $\alpha$.

mechanisms. We test mechanisms that are in use in existing peer-to-peer systems, and we compare them to the use of an ideal bootstrap server. We define an ideal bootstrap server as one that is able to return a list of nodes chosen randomly with equal probabilities from all the nodes present in the system, when contacted by a new node that needs to connect to the network. Note that in this section, we are not attempting to propose a scheme for a bootstrap server as it is beyond the scope of our research, however we are testing the dependence of Phenix on the different bootstrapping mechanisms.

We compare an ideal bootstrap server to a system where nodes on their first connection to the network obtain a list of existing nodes as in the case of the ideal bootstrap mechanism, however, we incorporate the idea of caching where a node $i$ saves the addresses of its neighbors $G_i(t_0)$ that it acquired during a previous connection at time $t_0$. Node $i$ favors connecting to the same set of neighbors at a later time $t_n$. This mechanism biases connections to be made to nodes that stay connected to the network for an extended period of time. By testing against this caching mechanism, we want to ensure that Phenix does not compromise its resilience in such a situation. We implement Phenix with 4000 distinct nodes whose session lifetimes follow a distribution similar to observations of empirical data as reported by [28,29]. We measure the degree distribution of all nodes in the network, whenever the size of the network exceeds 2000 nodes. The averaged results over 10 runs are shown in Fig. 9a. We can observe that the system still follows a power-law distribution preserving the desired characteristic of a low-diameter.

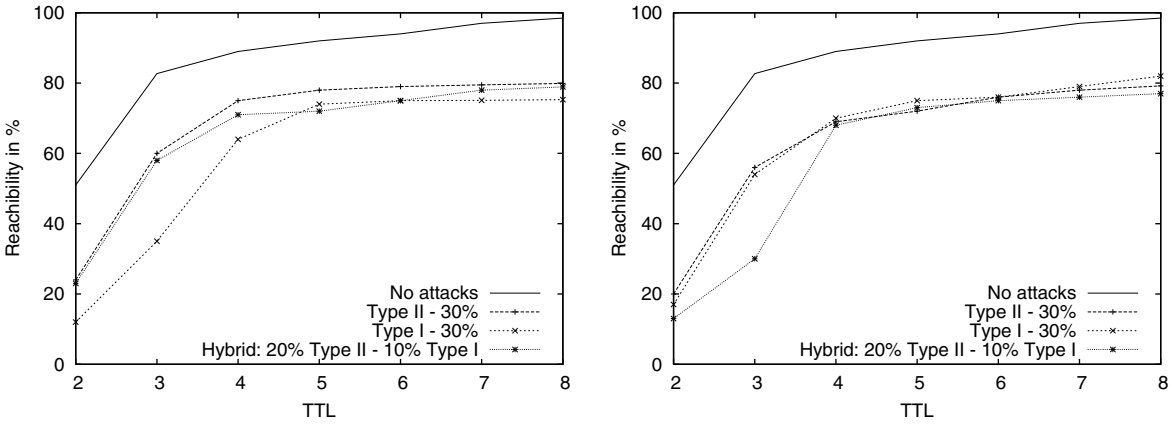
In order to measure the resilience of Phenix with such a bootstrapping mechanism, we repeat the experiment of Group Type I attacks, Group Type II attacks as well as hybrid attacks. The results are shown in Fig. 10a. We can observe that such a bootstrapping mechanism does affect the performance but to a limited extent in the sense that the reachability is lower than that for the case of an ideal random bootstrap server. However, these aggressive attacks did not succeed in dividing the network into separated islands. The reasoning behind this is that under the ideal random bootstrapping, nodes, who emerged as preferred nodes were not necessarily the "oldest" in the system, since no caching is implemented. On the other hand, caching neighbors connections on client nodes changes the system by improving the chances of malicious nodes since they are staying in the system for a prolonged period of time and a returning node is more likely to connect to one of them than to a legitimate node. This adds to the effect of the simultaneous disappearance of malicious nodes helping them create a noticeable void in the overall presence of preferred nodes in the network, thus increasing the diameter. In addressing this void of preferred nodes, the remaining nodes are able to recover to a power-law distribution after one update of their list of neighbors, promoting existing nodes into a preferred status.

Another mechanism of bootstrapping that we test against is when the bootstrap server does not know about all the nodes in the system, but instead has knowledge about a smaller subset that it chooses randomly from. The size of this subset is represented as a percentage of the total number of nodes that we denote by $\rho$. In such a scenario, the




(a) Degree Distribution While Using Caching (b) Degree Distribution With Partial Knowledge

Fig. 9. Degree distribution on a log–log scale.




(a) Group Attacks While Caching

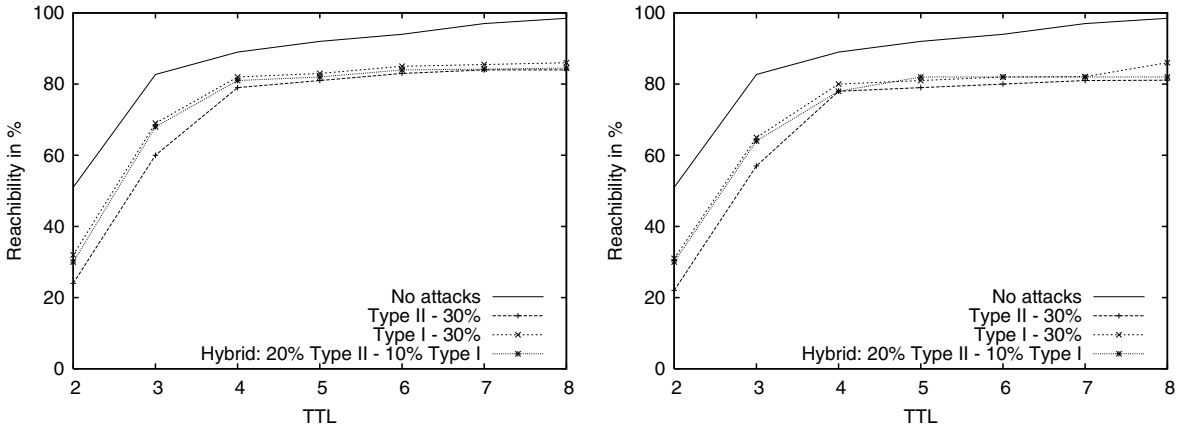
(b) Group Attacks With Partial Knowledge

Fig. 10. Effects of attacks on reachability.

bootstrap server will still return a set of random nodes when contacted by new-coming nodes, however, this set is biased towards nodes that it knows about giving them a higher chance of being in control of which nodes become preferential. One might imagine that a bootstrap server should be able to know about a high percentage of nodes connected to the network since these nodes contact the bootstrap server before connecting to the network, allowing the server to add them to its list. However, this is often not the case due to the fact that nodes might use their cache from a previous session, as presented above, while they have a different IP DHCP-obtained; this will make the bootstrap server oblivious to their presence in the network. Another reason why a bootstrap server cannot obtain full knowledge is due to the use of distributed bootstrapping infrastructure on several servers, which typically do not exchange information among each other for scalability reasons; thus resulting in each bootstrap server having a partial view of the network.

We test Phenix using a network of 2000 nodes that operate with five distinct bootstrap servers. We assume that the initial subset of 20 nodes appearing in the network is known to all five of the bootstrap servers. However, any subsequent arriving node will pick a bootstrap server randomly with equal probabilities, and queries it for random nodes. At that instance, that specific server will add this new node to its list of known nodes. We observe that the degree distribution of this network is still power-law-like as seen in Fig. 9b. We test the reachability of Phenix using such a mechanism under normal operation as well as under attacks for the same setup of 2000 nodes and five mutually independent bootstrap servers. The results are shown in Fig. 10b. Testing this algorithm against malicious attacks of Group Type I, Group Type II, and Hybrid shows that the network remains resilient under the first two cases of attacks, but seems to lose more under the Hybrid attack. Note that under the Hybrid attack the network does not get disconnected but instead its typical diameter increases deviating from a power-law behavior. The reason behind this is that with partial knowledge of nodes, malicious nodes constitute a set of preferred nodes and another set of nodes pointing to them. Thus, if we picture the network where the preferred nodes are in the center, the ones pointing directly to them constitute a circle around them. The Hybrid attack strategy puts malicious nodes in the center as well as a set of nodes around them. Thus, the topology becomes similar to a star topology. As the nodes in the center of the star and a big portion in the first layer disappear, as they are malicious, the network does not have sufficient connections to sustain the power-law distribution; consequently the diameter increases. Note that in our experiments, it took the nodes two rounds of the update mechanism to acquire a power-law distribution back, instead of the regular one round of updates that is sufficient under previous mechanisms and attacks scenarios.

Under such conditions, it seems necessary for the nodes to discover other nodes more aggressively instead of relying on the initial set. In order to alleviate from this issue, we modify the Phenix algo-




(a) Group Attacks With Additional Discovery

(b) Group Attacks With Using 2 Bootstrap Servers

Fig. 11. Effects of attacks on reachability with additions to Phenix.

rithm by adding another mechanism that we call the discovery stage, which takes place during the initial connections stage. In the discovery stage, a node starts by connecting to one of the random nodes in $G_{\text{host},i}$, and sends a special ping message with $\text{TTL} = x$, where $x > 1$ and chosen randomly. Each node $j$ receiving this special message will decrease the TTL by 1 and forward the message to only one of its neighbors also chosen randomly, as long as $\text{TTL} > 1$. If $\text{TTL} = 1$, then the receiving node $j_x$ will send back a list of its neighbors (or a subset, if it is a preferred node) to the sender node $i$. This procedure introduces a more diverse sample that a node can use as a startup point to collect its final list of neighbors, while maintaining restricted crawling capabilities that a malicious node can abuse. In fact, this newly obtained list of neighbors from node $j_x$ will be used by node $i$ as $G_i$ defined in Eq. (2). Note that no matter how deep a node sends a ping message, it will stay in the "circle" of malicious nodes if it had already started with one of them forcing it to connect to the circle as its sole outbound connection to the rest of the network. However, the probability that a node will have all of its initial set of nodes belonging to the set of malicious nodes is quite low. Another mechanism to overcome such situations requires a new node to contact more than one bootstrap server adding to the diversity of its initial set. The results of simulating both of these mechanisms are presented in Fig. 11a and b. In the first technique, nodes send the initial discovery message with $x$ chosen from the set $[2, 3, 4, 5]$ with equal probability. In the second technique, nodes contact two bootstrap servers chosen randomly from the set with equal probabilities. We can observe that the problem shown in Fig. 9b is not replicated under these modifications.

## 5. Experimental testbed results

We implemented Phenix in a real Internet-wide overlay environment running on the PlanetLab experimental testbed [24] for the purpose of measuring the overhead of the algorithm in the face of aggressive node removal scenarios. The code is built on the Open Source Jtella software system [17], a Java API for implementing the Gnutella protocol. We present our results from an implementation and experiment that ran on 81 PlanetLab nodes. We also measured the time needed for the network to recover from an attack targeted at highly connected nodes in the Phenix overlay running on PlanetLab.

### 5.1. Implementation

Each node in our implementation has two layers. The first layer being the Phenix algorithm composed of a servent (server and client) daemon responsible for incoming as well as outgoing connections. The node opens a socket connection waiting for incoming connections from other nodes either sending an $M_0$ (as described in Eq. (2)), or nodes wishing to add this node to their neighbors' list. In terms of the graph, this connection receives and services all the incoming edges pointing to this node. The second type of connection constitutes all the connections that a node opens to other nodes, or the outgoing

connections. As for the second layer, it is purely for experimental purposes, and opens a listening socket interacting with a central control server. The purpose of this latter layer is to be able to monitor the connections of a node in order to observe the progress of the network formation as well as the emerging topology. In addition, the control server can send a stop signal to this layer asking it to remove the node from the overlay network; thus, emulating targeted node removal. The implementation is performed by modifying the JTella API which is a Java module based on Gnutella v0.6 [15]. The modifications are mainly in acquiring hosts and creating outgoing connections, making it conform to the Phenix algorithm, presented in Section 3, instead of the random Gnutella topology.

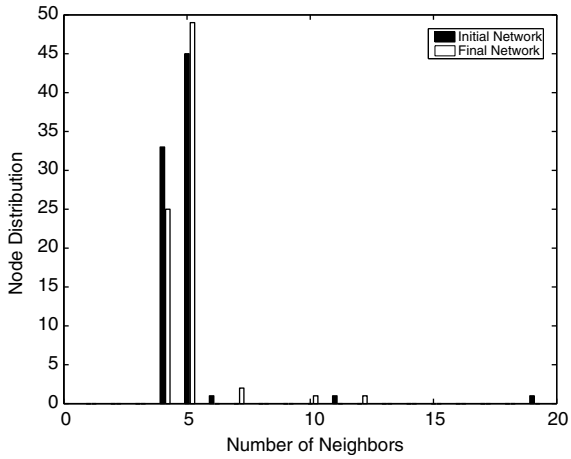



Fig. 12. Out-degree (number of neighbors) distribution.

### 5.2. Degree distributions experiments

The Phenix overlay ran on the 81 PlanetLab nodes spread over 43 sites across eight countries (Australia, Canada, Germany, Hong Kong, Sweden, Taiwan, UK, and US). The network started with $n_{\text{init}} = 10$ nodes interconnected randomly, in order to boot up the process of network formation. After that, nodes started joining at the rate of two nodes every 5 s by contacting the control server, which acts as a bootstrap server and provides the rendezvous mechanism by giving each node a list of four nodes that it can connect to. The generated list of nodes, given as a response for each request, is drawn randomly from nodes that have already joined the system with no bias given towards node location or proximity.

Thus, each starting node contacted the control server to get the initial $G_{\text{host}}$ list, and applied the Phenix algorithm in making its decisions. In the following experiment, we chose the values of 3 and 4 for min and max (lower and upper bounds on the number of initial neighbors for a node, respectively), since the number of nodes (81 nodes) is a small number as compared to the growth of peer-to-peer systems in today's networks. Choosing higher values for min and max would create a network that is closer to a mesh while lower values can easily result in situations where a node might find itself completely disconnected from the rest of the network with the removal of few nodes. Following the complete formation of the network and connections of all nodes, we took a snapshot of the resulting graph by examining the nodes' neighbors' list. Fig. 12 presents the out-degree distribution (or number of formed outgoing connections) for the entire Phenix overlay network. The purpose behind this metric is to examine the number of nodes that emerged as preferential nodes and their respective degrees, as they acquired backward connections, thus, becoming hubs in the overlay network. We can see from the figure that the majority of nodes have between 3 and 4 neighbors, with the exception of three nodes with 5, 10, and 18 connections respectively. Before sending these three nodes the command to close their incoming and outgoing connections, we measured the rtt (round trip time) from the control server to every node in the network in order to see the diversity of the connections. Fig. 13 shows the distribution of rtt for the overlay nodes. We can observe that although the majority of the nodes are within less than 100 msec reach from the control server, some offered a diversity in the

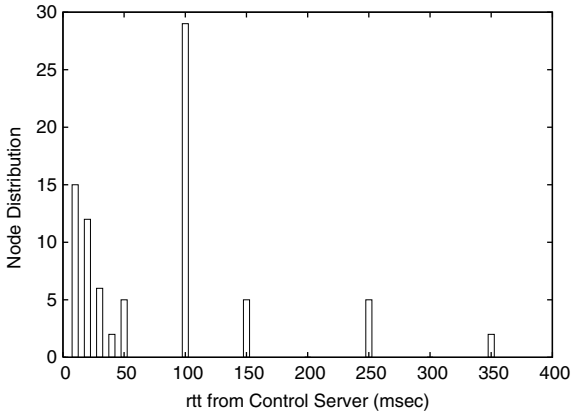



Fig. 13. Round trip time (rtt) distribution of nodes in the testbed.

network where their rtt reached higher values up to 350 msec, thus, offering a degree of heterogeneity for the experiment.

In this experiment we sent the three highly connected nodes (with 5, 10, and 18 connections) a stop signal through their control layer forcing them to close all of their connections. We then waited for the reaction of the rest of the nodes in the Phenix overlay, and measured how long it took them to rearrange their connections and send their new state to the control server. Several factors enter into play when obtaining these results as presented by $t_i$: $t_i = \mathrm{rtt}_j/2 + \zeta_i + \mathrm{rtt}_i + \eta_i + \mathrm{rtt}_i/2$. The total time needed for a node $i$ to inform the control server that it performed the node maintenance, denoted by $t_i$, is the summation of five terms presented above. The first term is the time needed for the stop message to travel from the control server to the node to stop $j$, denoted by $\mathrm{rtt}_j/2$. The second term is the time needed for the node $i$, in the case it is connected to node $j$, to realize that node $j$ is no longer available (or the timeout of the connection, in this case we chose the value to be 1000 msec), denoted by $\zeta_i$. The third term $\mathrm{rtt}_i$ is the time needed for node $i$ to contact the control server requesting the address of one or more nodes that it can connect to, denoted by $\mathrm{rtt}_i$. The fourth term, denoted by $\eta_i$, is the time needed to run the Phenix algorithm, which might involve contacting a friend node in the case of acquiring a preferential node. Finally, the fifth term $\mathrm{rtt}_i/2$ is the time required to send the node maintenance outcome for the control server informing it of the change in the neighbors list. The distribution of time for each of the affected nodes to run this node maintenance mechanism is shown in Fig. 14. We can observe that most nodes returned to a stable state by creating new connections in less than 1 s. Finally, Fig. 12 shows a comparison of the resulting connectivity with the initial overlay graph, where we observe that four new highly connected nodes emerged ensuring the fast recovery of the Phenix overlay with a low-diameter topology.

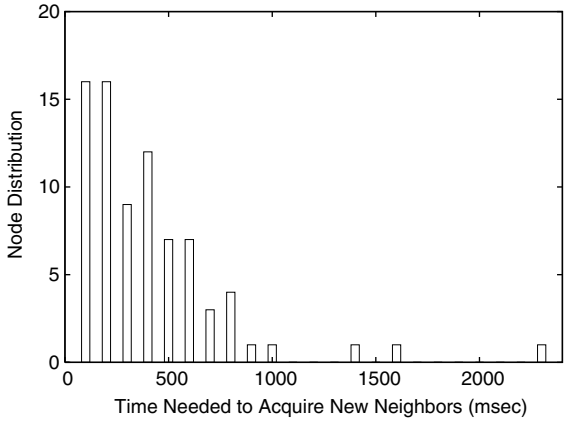



Fig. 14. Node maintenance duration.

## 6. Conclusion

We have presented a fully distributed algorithm called Phenix that creates low-diameter resilient peer-to-peer overlay networks. To the best of our knowledge Phenix represents one of the first contributions that simultaneously supports high performance in terms of low-diameter and fast response times, and is robust to attacks and resilient to various overlay dynamics and node failure scenarios. In this paper, we have shown through analysis, simulation, and from results from an experimental implementation on the PlanetLab overlay that Phenix results in efficient connectivity, offering tolerance to various network dynamics including join/leaves and a wide-variety of simple and more sophisticated node attacks. Because of the rise in number of security attacks and the growing creativity of attackers, the need for resilient overlays that can offer both performance and resilient properties will become necessary particularly for commercial reliable overlays. Phenix supports low-diameter performance and resilience without sacrificing flexibility.

## References

[1] L.A. Adamic, The small world web, in: Proceedings of the third European Conference on Digital Libraries, Lecture notes in Computer Science, vol. 1696, Springer, 1999, pp. 443–452.

[2] L.A. Adamic, R.M. Lukose, B.A. Huberman, Local search in unstructured networks, in: S. Bornholdt, H.G. Schuster (Eds.), Review Chapter to Appear in Handbook of Graphs and Networks: From the Genome to the Internet, Wiley VCH, Berlin, 2003.

[3] D. Adkins, K. Lakshminoarayanan, A. Perrig, I. Stoica, Towards a more functional and secure network infrastructure, UCB Technical Report No. UCB/CSD-03-1242.

[4] L.A.N. Amaral, A. Scala, M. Barthelemy, M. Stanley, Classes of small-world networks, in: Proceedings of the National Academy of Sciences, vol. 97(21), October 2000.

[5] D.G. Andersen, Mayday: distributed filtering for Internet services, in: Proceedings of the Fourth Usenix Symposium on Internet Technologies and Systems, Seattle, WA, 2003.

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, R. Morris, Resilient overlay networks, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), 2001.

[7] A-L Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509.
[8] A-L Barabási, R. Albert, Statistical Mechanics of Complex Networks, Center for Self-Organizing Networks, University of Notre Dame, Notre Dame, Indiana.
[9] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making Gnutella-like P2P systems scalable, in: Proceedings of the 2003 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM Sigcomm 2003), 2003, pp. 407–418.
[10] C. Cooper, M. Dyer, C. Greenhill, Sampling regular graphs and a peer-to-peer network, Combin., Prob. Comput. 16 (2007) 557–593.
[11] S. El-Ansary, L.O. Alima, P. Brand, S. Haridi, Efficient broadcast in structured P2P networks, in: Proceedings of the Second International Workshop on peer-to-peer Systems (IPTPS'03), Berkeley, CA, February 2003.
[12] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the Internet topology, in: Proceedings of the 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM Sigcomm 1999), 1999, pp. 251–262.
[13] T. Feder, A. Guetz, M. Mihail, A. Saberi, A Local Switch Markov Chain on Given Degree Graphs with Application in Connectivity of Peer-to-Peer Networks, Foundations of Computer Science (FOCS 2006), Berkeley, CA, October 2006.
[14] Gnucleus. The Gnutella Web Caching System. http://gnucleus.sourceforge.net/.
[15] The Gnutella RFC. http://rfc-gnutella.sourceforge.net/.
[16] M. Jovanovic, Modeling large-scale peer-to-peer networks and a case study of Gnutella, Master's thesis, University of Cincinnati, 2001.
[17] JTella. http://jtella.sourceforge.net/.
[18] A.D. Keromytis, V. Misra, D. Rubenstein, SOS: secure overlay services, in: Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM Sigcomm 2002), 2002, pp. 61–72.
[19] B.J. Kim, C.N. Yoon, S.K. Han, H. Jeong, Path finding strategies in scale-free networks, Phys. Rev. E 65 (2002) 027103.
[20] P.L. Krapivsky, G.J. Rodgers, S. Redner, Degree distributions of growing random networks, Phys. Rev. Lett. 86 (2001) 5401.
[21] Merriam-Webster online. http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=resilience.
[22] G. Pandurangan, P. Raghavan, E. Upfal, Building low-diameter P2P networks, IEEE J. Selected Areas Commun. 21 (2003) 995–1002.
[23] G. Pandurangan, P. Raghavan, E. Upfal, Building P2P networks with good topological properties, Technical Report, 2001.
[24] PlanetLab. http://www.planet-lab.org/.
[25] Query Routing for the Gnutella Network, Version 1.0, http://www.limewire.com/developer/query_routing/keyword%20routing. htm.
[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM Sigcomm 2001), 2001, pp. 161–172.
[27] J. Ritter, "Why gnutella can't scale. no, really," http://www.darkridge.com/jpr5/doc/gnutella.html, 2001.
[28] S. Saroiu, P.K. Gummadi, S.D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking (MMCN) 2002, San Jose, CA, USA, January 2002.
[29] S. Sen, J. Wang, Analyzing peer-to-peer traffic across large networks, in: Proceedings of the Second ACM SIGCOMM Workshop on Internet Measurement Workshop, Marseille, France, 2002, pp. 137–150.
[30] Sharman Networks LTD. KaZaA Media Desktop. http://www.kazaa.com/.
[31] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM Sigcomm 2001), 2001, pp. 149–160.
[32] Ultrapeers: Another Step Towards Gnutella Scalability. http://groups.yahoo.com/group/the_gdf/files/Proposals/Ultrapeer/Ultrapeers_1.0.htm.
[33] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (1998) 440–442.
[34] R.H. Wouhaybi, A.T. Campbell, Phenix: supporting resilient low-diameter peer-to-peer topologies, in: Proceedings of IEEE INFOCOM'2004, Hong Kong, China, March 7–11, 2004.
[35] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, IEEE J. Selected Areas Commun. 22 (2004) 41–53.

**Rita H. Wouhaybi** is a research scientist at Intel Corporation. She has obtained her Ph.D. from the Electrical Engineering Department of Columbia University. Her research interests include peer-to-peer networks, and game theory. She has also received her Master and Bachelor degrees in Computer and Communications Engineering in 1996 and 1994, respectively, from the American University of Beirut.

**Andrew T. Campbell** recently joined Dartmouth College as an Associate Professor in Computer Science and is a member of the Center for Mobile Computing (CMC) and the Institute for Security Technology Studies (ISTS). Prior to joining Dartmouth Andrew was an Associate Professor of Electrical Engineering at Columbia University (1996–2005) and a member of the COMET Group. His current research interests include the development of resilient sensor networks, intrusion detection systems for WiFi networks, and open spectrum wireless networks.

He received his Ph.D. in Computer Science (1996) from Lancaster University, England, and the NSF Career Award (1999) for his research in programmable wireless networking. Prior to joining academia he spent 10 years working in industry both in Europe and the USA in product research and development of computer networks and wireless packet networks. He spent his sabbatical year (2003–2004) at the Computer Lab, Cambridge University, as an EPSRC Visiting Fellow. In 2005 he and his family relocated from Manhattan to Norwich, Vermont.