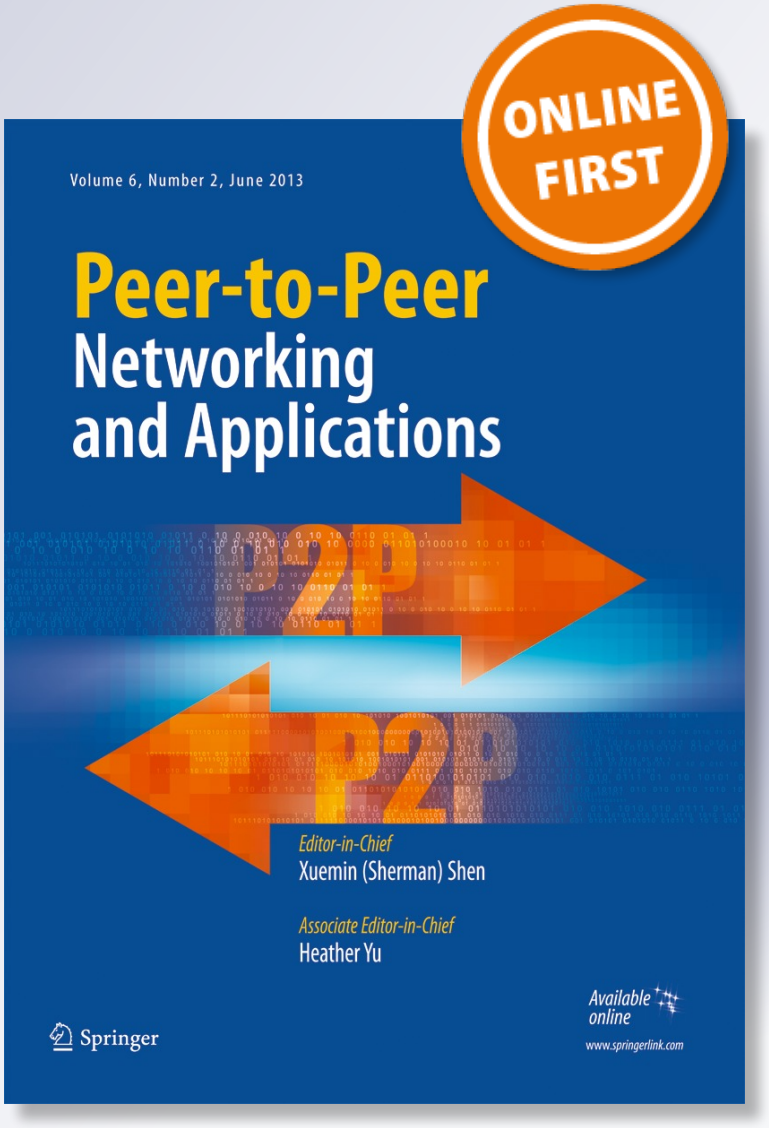
Tsunami: A parasitic, indestructible botnet on Kad

Ghulam Memon, Jun Li & Reza Rejaie

Peer-to-Peer Networking and Applications

ISSN 1936-6442

Peer-to-Peer Netw. Appl.
DOI 10.1007/s12083-013-0202-x



Springer

Peer-to-Peer Netw. Appl.
DOI 10.1007/s12083-013-0202-x

Tsunami: A parasitic, indestructible botnet on Kad

Ghulam Memon · Jun Li · Reza Rejaie

Received: 13 July 2012 / Accepted: 15 February 2013

Abstract While current botnets rely on a central server or bootstrap nodes for their operations, in this paper we identify and investigate a new type of botnet, called *Tsunami*, in which no such bottleneck nodes exist. In particular, we study how a Tsunami botnet can build a parasitic relationship with a widely deployed P2P system, Kad, to successfully issue commands to its bots, launch various attacks, including distributed denial of service (DDoS) and spam, at ease, as well as receive responses from the bots. Our evaluation shows that in a Kad network with four million nodes, even with only 6 % nodes being Tsunami bots, Tsunami can reach 75 % of its bots in less than 4 min and receive responses from 99 % of bots. We further propose how we may defend against Tsunami and evaluate the defense solution.

Keywords Botnet · DHT (distributed hash table) · Kad · DDoS · Spam

1 Introduction

Botnets—i.e., networks of compromised computers called bots—are one of the most challenging network security problems of today's Internet. Their threat is clear: a botmaster can use a command and control (C&C) channel to direct thousands or even millions of bots to launch large-scale security attacks (e.g., DDoS, spam campaign).

The most critical component of botnets is their C&C channel. While botnets have employed several different types of C&C channels, all C&C channels so far rely on a central server or bootstrap nodes to operate, and such "bottleneck" nodes can be the weakest points of a botnet. For example, a centralized botnet (e.g., an IRC botnet or a HTTP botnet) can stop functioning if its central server is located and shut down, and a peer-to-peer botnet (e.g., Nugache [19], Phatbot [2]) will not be able to recruit new bots if its bootstrap nodes are captured.

We introduce a new type of botnet in this paper and draw the research community's attention towards it. Such a botnet will not have a central server or bootstrap nodes, or any types of bottleneck nodes, and every bot will be equal to each other. Without bottlenecks to target, the only way to bring down such a botnet is to remove all the bots, which can become a vast amount of work if such a botnet consists of a very large number of bots. Moreover, this hard-to-destroy feature holds true even if one may identify the bots on this botnet or discover its C&C traffic.

This new type of botnet we introduce has an additional feature. Instead of building *dedicated* C&C channels for communication, this botnet acts as a parasite, and exploits a widely deployed, benign distributed system as its host. It sprinkles its bots randomly across the benign host system, and uses the built-in communication channels of the host system for its own C&C.

G. Memon (✉) · J. Li · R. Rejaie
University of Oregon, 120 Deschutes Hall,
1202 University of Oregon, Eugene,
OR 97403-1202, USA
e-mail: gmemon@cs.uoregon.edu

J. Li
e-mail: lijun@cs.uoregon.edu

R. Rejaie
e-mail: reza@cs.uoregon.edu

Published online: 19 April 2013

While a centralized botnet may also piggyback on other communication platforms, such as the IRC botnet or the HTTP botnet, this new botnet is more advanced by avoiding the need of a central server as required by a centralized botnet. It will still have a botmaster, of course, but unlike the botmaster in a centralized botnet and many P2P botnets, the botmaster here does not need to be always online. It can simply inject its command and run away.

This new botnet is also more dangerous than today's peer-to-peer botnets, which all have their own dedicated network. Not only does the new botnet avoid the need for the bootstrap nodes as required by these peer-to-peer botnets, it also avoids the need for building its own dedicated C&C channels.

We set out to study this new type of botnet in this paper. As the botnet traffic is hidden under the regular traffic, and is similar to a tsunami moving forward under the ocean wave, we call the botnet **Tsunami**. We choose *Kad* to be the host system for Tsunami. Kad is a public network that serves over four million users. It is an implementation of the Kademlia [10] Distributed Hash Table (DHT). Kad is used by the eMule file-sharing system [3]. We study how Tsunami can accomplish a C&C channel without points of failure by running on top of Kad as a parasitic botnet. Specifically, we will study how Tsunami can use Kad to issue its commands to millions of bots, receive responses from them, and conduct surreptitious but damaging functions as needed. Also, using Kad as the host system, we can study the security of Kad, including how a P2P network such as Kad can be exploited and become an involuntary host of a botnet.

We further evaluate Tsunami to show that when 5 % nodes of Kad are Tsunami bots, a command can reach virtually all of them in less than 6 min. Even when Tsunami bots experiences churn (bots arrive and depart all the time), with 6 % Kad nodes that are bots, a Tsunami botmaster can still reach 75 % bots in 4 minutes.

In addition to designing the botnet, we also study how to defend against Tsunami. Since capturing and destroying Tsunami bots is extremely hard and costly, we study how we may capture the Tsunami traffic, mislead the bots, and thus disrupt their C&C communication.

The rest of this paper is organized as follows. We first summarize the related work in Section 2. Then, after providing a background on the Kad system in Section 3, we describe the design of Tsunami in Section 4 and two examples of launching attacks via Tsunami in Section 5. We evaluate Tsunami's performance in Section 6, and describe our current implementation of Tsunami in Section 7. Section 8 discuses our defense strategies against Tsunami, and we conclude the paper in Section 9.

2 Related work

In this section we provide a brief review of existing botnet C&C channels. In particular, we consider three general C&C categories: centralized, peer-to-peer, and hybrid. We highlight the generic disruption techniques for each of them.

Centralized botnets These botnets use a central C&C server. In order to receive new commands, the bots maintain permanent or periodic TCP connections with the C&C server. Several examples of such botnets exist: Mybot, Gobot [13], Mebroot [18] and Machbot [4]. This centralized design, while very efficient, makes it hard for a C&C server to hide its identity. If a single binary that carries the location of the C&C server is captured, the server can be located and shut down. Even if the binary does not possess the location of the C&C server, a bot's traffic pattern may reveal the location.

Peer-to-peer (P2P) botnets The basic goal of P2P botnets is to avoid the shortcomings of centralized botnets. The biggest shortcoming of centralized botnets is the existence of a central point of failure. Examples of real-world P2P botnets include Nugache [19] and Phatbot [2]. Both the botnets form an unstructured P2P network, similar to the Gnutella P2P network [14]. The commands propagate through these botnets via *gossiping*, i.e., each bot passes the command to its directly connected peers. Wang et al. [21] also designed an unstructured P2P botnet where servant bots with static, public IP address are responsible to ensure the connectivity of the botnet.

While P2P botnets eliminate the central point of failure, these botnets are faced with new potential threats: the removal of their bootstrap nodes. The bootstrap nodes are a set of nodes, through which peers can join a P2P network. P2P botnets, similar to all P2P networks, also require the bootstrap nodes. For example, Nugache uses a fixed set of 22 bootstrap nodes; Phatbot uses Gnutella cache servers for bootstrapping; and Wang et al.'s botnet uses servant bots for bootstrapping. Bootstrap nodes are Achilles' heel [7] of P2P botnets: if the bootstrap nodes are captured and compromised, then bots cannot join the botnet.

Hybrid botnets Hybrid botnets combine centralized C&C with peer-to-peer networking. Specifically, hybrid botnets use P2P networks as a directory service for bots to discover the location of the central C&C server. Examples of hybrid botnets include Storm [8], Trojan.Peacomm [7] and Alureon [1].

While Storm and Trojan.Peacomm botnets are significantly different from Tsunami, Alureon bears some resemblance to Tsunami. Alureon is a new botnet, and very little is known about it. Based on the little information that we

have, we know that both Tsunami and Alureon hide their bots in the Kad DHT.

To the best of our knowledge, hybrid botnets are the most sophisticated botnets in existence given their use of public P2P networks and encryption. However, hybrid botnets (including Alureon) still rely on the presence of a centralized C&C server for bots to *pull* commands from. A dedicated defender may continuously monitor a target P2P network and discover the location of the C&C server, which can then be destroyed. Tsunami, on the other hand, uses a *push* model for issuing commands: the C&C server injects a command into a public P2P network. Given the way the commands are propagated through the P2P network, it is challenging to discover the original source of the command. In other words, a Tsunami C&C server hides behind regular bots, which makes Tsunami indestructible.

Hybrid botnets also face the threat of directory poisoning. The location of central C&C server is published at well-known rendezvous locations in the P2P network. The defenders may poison the directory service and change the location of the central C&C server [7].

3 Background and preliminary

Kad is a peer-to-peer (P2P) system based on Distributed Hash Table (DHT). In this section we first introduce DHTs, then describe Kad's features that are relevant to Tsunami.

3.1 Distributed hash table (DHT)

A DHT is a peer-to-peer (P2P) system that provides a lookup service similar to a hash table, where every $(key, value)$ pair is stored at a peer whose ID is closest to key. Different DHTs have different schemes to define what is meant by "closest," such as the XOR distance in Kademlia [10], the numeric difference in Chord [17], and prefix matching in Pastry or Tapestry [15, 22].

DHTs use *iterative routing* to find a peer whose ID is closest to a key K. The iterative routing process of DHTs is demonstrated in Fig. 1. It shows three different types of peers:

1. Peer P_r, which starts the iterative routing process.
2. Peer P_d, which is the closest peer to K.
3. Peers P_i and P_j, which are intermediate peers that route requests from peer P_r towards peer P_d.

Peer P_r initiates the routing process by consulting its own routing table and finds a peer P_i that is closest to the key K. Peer P_r sends a request for K to the peer P_i, as shown in Fig. 1. Peer P_i consults its own routing table, finds the closest possible peers to K, and informs the peer P_r about the newly discovered peers that are closer to K. This process

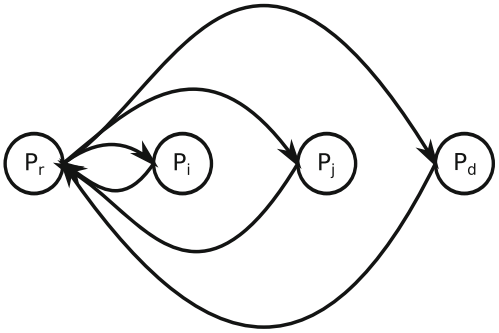


Fig. 1 Iterative routing process in DHTs

continues until the peer P_r finds the peer P_d, which is the closest possible peer to K.

The type of $(key, value)$ pairs in a DHT depends on the usage of the DHT. In a file-sharing DHT, the key is typically the hash of a file and the value is the location of the peer storing the file. To help search files using their keywords, sometimes the key is the hash of a keyword of a file, and the value is the meta-data of files matching the keyword. Call peers that publish content *publishers*, and peers that consume content *consumers*. Using DHT algorithms, a publisher can map a file to a specific peer and store the file in that peer, and a consumer can find the peer that stores the file.

3.2 Kad

Used by eMule [3] and other file sharing applications, Kad is a DHT with the largest user base (more than four million users). It is derived from Kademlia [10] DHT, and uses 128-bit IDs for peers and keys. Since there are 2^{128} peer IDs and there is little chance of collision, every peer chooses their IDs randomly. Kad uses XOR metric to calculate the distance between IDs where an ID can belong to either a peer or a key (i.e., the distance between IDs x and y is $x \oplus y$).

Kad embraces a unique lookup algorithm that can locate nodes "closer" to a target ID hop by hop, and it takes logarithmic steps to finally discover multiple "closest" nodes. Specifically, upon the receipt of a lookup message, a Kad peer will discover a *k-bucket* that contains k contacts closer to the target ID than itself, and forwards the lookup request in parallel to three out of k contacts.

Kad allows a publisher to `publish` content and a consumer to `search` specific content. A `publish` operation includes a *lookup* phase and a *storage* phase. A `search` operation includes a *lookup* phase and a *retrieval* phase.

Lookup phase In this phase a publisher (or a consumer) sends a lookup message to discover the peers whose IDs are the "closest" to the key of the content to be published

(or to be consumed). This phase is supported by the lookup algorithm of Kad.

Storage/retrieval phase In this phase a publisher or a consumer directly contacts the peers discovered in the lookup phase. The publisher stores the content (or information about the content) at the peers, and a consumer retrieves content (or information about the content) from the peers. Kad supports two different kinds of content: keywords and files. Thus there are four different kinds of messages:

- PUBLISH_FILE (PuF) for publishing a file, where the *key* field of the message is the hash of the file, and the *value* field of the message is the publisher's IP address and TCP port number.
- PUBLISH_KEYWORD (PuK) for publishing a keyword, where the *key* field of the message is the keyword's hash, and the *value* field is the meta-data of those files that include the keyword in their names. The meta-data for each file includes their name, hash, size and type.
- SEARCH_FILE (SeF) for searching a file, where the *key* field of the message is the hash of the file. A response to this message carries the IP address and TCP port number of the file's publisher.
- SEARCH_KEYWORD (SeK) for searching the files associated with a keyword, where the *key* field of the message is the hash of the keyword. The response to this message carries the meta-data of relevant files, including their hash. For each desire file, a consumer then can use its hash just retrieved to issue an SeF request to finally get the file.

4 Design: Tsunami botnet as a parasite of Kad

As we discussed in Section 1, the goal of Tsunami is a C&C channel without points of failure, and it accomplishes this goal by being a parasite of the Kad system and leveraging the built-in communication capabilities of Kad. We describe how Tsunami can achieve these capabilities in this section.

4.1 Tsunami objective refined

Every Tsunami bot embeds itself in Kad as a peer node in Kad, and does not even necessarily know any other Tsunami bots. Nevertheless, a Tsunami botmaster must be able to send commands to Tsunami bots, and perhaps also receive responses from the bots. Of course, Tsunami would prefer a good performance such as reaching its bots quickly. Albeit not required, Tsunami bots can also try to stay unnoticeable.

A Tsunami botmaster does not know which peers in Kad are Tsunami bots. In order to reach them, the botmaster can send a message to *every* peer in Kad. However, as Kad uses a 128-bit ID space to represent peers, doing so implies 2^{128} messages. This is not only resource intensive, but will also make the botmaster appear anomalous and get captured.

This problem can be resolved by realizing that a 128-bit ID space can never be fully populated. In fact, Kad uses a large ID space to avoid collisions between the IDs of different peers. We know from our earlier measurement study on Kad [11] that in practice, only the top 22 bits of a peer's ID are used to locate peers. In other words, as long as the top 22 bits are common between a lookup ID and a peer ID, the desired peer will be found. Thus, instead of sending 2^{128} messages, the refined objective of Tsunami is for the botmaster to send 2^{22} messages instead to broadcast its command. Note that 2^{22} is the total number of messages that needs to be sent; individual bots only send a small fraction of these messages, as explained in Section 4.3

Our measurement study on Kad [11] shows that the number of bits required to locate peers is directly proportional to peer population; as peer population increases, more bits are required to locate peers. Thus, Tsunami must adjust the number of messages as the peer population changes, and also must cope with reduced number of command-carrying bits. While we do not provide a technical mechanism to address this problem, we note that only when peer population doubles, one bit needs to be added to the lookup bits (i.e., instead of 2^{22} messages, 2^{23} messages need to be sent). The botmaster may periodically crawl the Kad DHT, and as the peer population doubles, it may increase the number of messages.

4.2 Tsunami command

The Tsunami botmaster issues commands using Kad *lookup* messages. In particular, the target ID in such Kad lookup messages is divided into the following two parts (Fig. 2):

- Lookup Bits: The top 22 bits are used to find Tsunami bots. As we discussed in Section 4.1, using the top 22 bits is sufficient for finding bots.
- Command Bits: The lower 106 bits are used to carry a Tsunami command. For example, it can encode multiple 32-bit IP addresses as the victims of a DDoS attack. The reader may observe that Kad may counter this communication mechanism by always using zeros for lower order 106 bits. However, recall from Section 3 that in Kad each file and keyword is uniquely identified by a 128 bit ID. Decreasing the number of bits by using zeros for lower order bits will lead to collisions.

Upon the receipt of a Kad lookup message that carries a Tsunami command, a benign peer will simply process it as a normal lookup message, while a Tsunami bot must be

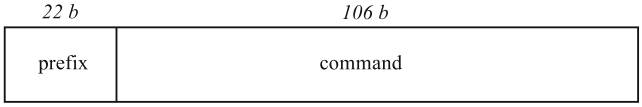


Fig. 2 Tsunami command format

able to recognize it, obtain the command, and act upon the command. To meet this condition, the command bits can carry a command signature.

The botmaster can further encrypt the command bits, such as by using the private key of the botmaster, and a bot can decrypt the command and verify the command is indeed from the botmaster. This will require every bot to have the decryption key, such as the public key of the botmaster, when distributing the binary code of the botnet.

4.3 Sending commands to tsunami bots embedded in Kad

We now describe how a Tsunami command can reach Tsunami bots embedded in Kad. In doing so, Tsunami wants to (1) reach as many bots as possible, (2) be fast in reaching them, and (3) stay relatively quiet in order not to appear suspicious to ISPs.

Tsunami uses a tree-based command propagation mechanism. First, the botmaster sends lookup messages toward a small number (e.g., 100) of target IDs. Every bot that happens to receive a lookup message will repeat the same process. If a bot discovers that a command is a duplicate (such as that caused by a loop), it will simply drop the command. Note that the tree is constructed on-the-fly, and does not require any coordination among the bots. Since the bots receive messages purely by chance, the larger the bot population the higher the reachability of the command (Section 6).

As shown in Fig. 3, this process will involve three different kinds of nodes:

- M represents the bot master.
- B represents a bot.
- K represent a regular Kad peer.

We emphasize that the botmaster of a Tsunami botnet can hide itself in two measures: First, the botmaster hides itself behind regular bots. If a defender captures a Tsunami bot binary and finds out the pattern of Tsunami commands,

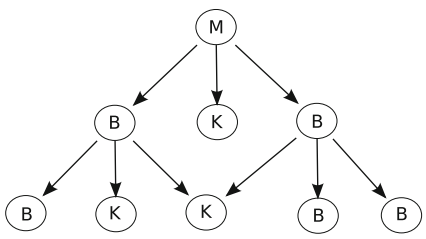


Fig. 3 Sending commands to tsunami bots embedded in Kad

it could detect Tsunami commands and try to trace who sent the commands. However, there is no way to distinguish between a message sent by the botmaster and a message from one of thousands of bots. Furthermore, since lookup messages use UDP and their source IP address can be spoofed without affecting message delivery, the botmaster (and a regular bot as well) can fake its IP address at the last hop of the lookup message.

We introduce two forms of bot-bot communication:

- Whispering: For a bot, lookup messages simply provide a way to find other peers in the network. A bot hopes (without knowing) that some of the discovered peers are bots. As a result, the bot does not have to carry the command bits in its lookup message. It can simply find a peer closest to any random ID. Then it can send the actual command bits to the discovered peer using some other Kad protocol message. This approach saves the bot from exposing itself. During the lookup, at intermediate hops, a bot cannot fake its IP address because it expects replies to progress in routing. As a result, if a lookup message is intercepted in the middle and if the lookup message is carrying command bits, the bot itself can be exposed. This cautious approach, however, may reduce the efficiency of the botnet because after every lookup only one peer is sent the command bits.
- Shouting: Shouting is opposite of whispering. In this approach, the lookup messages issued by the bots carry the command bits. As a result, any bot along the way will learn about the command. This efficiency comes at the possible cost of exposure, as described above.

4.4 Receiving response

A botmaster may wish to receive responses for certain commands, such as the machine information of bots and the success rate of spam. Since Tsunami bots do not know the identity of the botmaster, they cannot directly send responses to the botmaster.

To solve this problem, Tsunami bots use *logical overlay links* that are created during command dissemination. At each step of forwarding a Tsunami command, a bot A will discover at least one bot B: when B forwards a command to A, A immediately learns that B is a bot and is a parent of A. This information creates a logical uni-directional link from A to B. Bot A sends its response to bot B, which collects responses from all its children and propagates aggregated response to its parents. Same as Tsunami command, the response messages are also delivery via Kad protocol messages.

A bot's parent(s) may depart, disconnecting it from the botmaster. However, the botmaster can recreate the overlay by periodically rebroadcasting a command. Each command

may carry a nonce to distinguish command re-broadcasts from original command broadcasts. Bots can respond at the first command-rebroadcast after the fulfillment of the original command.

Command responses can also help the botmaster to estimate the bot population. (As described in Section 6, the botmaster can then determine the number of copies that every bot needs to forward for a command.) Bots communicate this information by embedding it in the aggregated responses, where every aggregated response carries the number of responses merged. Note because the redundancy in the Kad network may artificially inflate the bot population, this approach only provides an *estimate* of bot population.

5 Tsunami attack examples

In this section we describe how Tsunami issues distributed denial of service (DDoS) and spam commands.

5.1 Distributed denial of service (DDoS)

A DDoS command can be delivered using either short commands or long commands.

Short commands A short DDoS command consists of two parts: a 10-bit unique command ID, and at least one target IP address. The command ID helps the botmaster identify the response to a command, as described in Section 4.4. The botmaster requires the command ID to be unique across unanswered commands only. Thus, the botmaster can issue 1024 ($2^{10} = 1024$) distinct commands in parallel.

Tsunami uses lookup messages to deliver short DDoS commands. It embeds the command in the 106-bit suffix of the lookup ID, as described in Section 4.2. It can also embed a Tsunami signature (e.g., a particular bit string) for bots to recognize it is a Tsunami command.

Long commands A long DDoS command consists of four components: a 10-bit unique command ID, IP address(es) or hostname(s) of the target, date and time of the attack, and bot population. As the Kad lookup message does not provide enough space to accommodate all components, Tsunami uses Kad's storage/retrieval-phase messages for these commands. It delivers long commands in two steps. First, it uses lookup messages to find peers for storing commands. Next, it sends PUBLISH messages (i.e., either PuK or PuF) to the discovered peers, with commands embedded in the value portion. Tsunami prefers PUBLISH messages over SEARCH messages, because SEARCH messages do not have a value portion, which limits the space for a command.

Tsunami uses different keys for a given pair of LOOKUP and PUBLISH messages. The keys share the prefix, but they differ in the suffix. The suffix of the lookup key is completely random because it does not carry a command. On the other hand, the suffix of the PUBLISH key must carry the command.

5.2 Spam

Because of the *bulky* nature of spam, issuing spam commands is more challenging than issuing DDoS commands. Modern spam campaigns [9] typically consist of list of email addresses, email template and a collection of values for macros in the text, called a dictionary. Tsunami solves this problem by storing the spam information as key-value pairs in Kad, and broadcasting only the keys to the bots.

The botmaster uses one PuK message for each component of spam information. First, the botmaster randomly chooses three 128-bit IDs K_1, K_2, and K_3 as the keyword hashes for three PuK messages M_1, M_2, and M_3, respectively. The botmaster uses the body of each PuK message to encode spam information. Next, it goes through the lookup phase to find the closest possible peers to K_1, K_2, and K_3, as described in Section 3. After that, it sends M_1, M_2 and M_3 to the discovered closest possible peers. Finally, the botmaster broadcasts K_1, K_2 and K_3 using the same method as DDoS.

As bots receive K_1, K_2 and K_3, they retrieve each component of spam information. First, each bot goes through the lookup phase to find the closest possible peers to K_1, K_2 and K_3. Next, each bot sends SeK messages to discovered peers and retrieves spam information. Finally, they construct spam email messages using email template and email dictionary, and start sending spam (e.g., every bot can randomly choose a pre-determined number of email addresses for sending spam).

6 Evaluation

We have developed a discrete event simulator to evaluate the performance of C&C in Tsunami when it runs on top of Kad. We now present the results for these evaluations.

6.1 Issuing commands

We use "time taken to reach maximum possible bots" as the main performance metric, which is affected by the following two factors:

- Bot population
- Number of lookup messages issued by each bot

During our simulations, we vary these two factors to understand their impact on command dissemination time. We use a real world churn model [20] to simulate the arrival and departure of the bots.

We keep peer population constant at 4,000,000 [11] and lookup latency constant at 8 s [16]. Note that lookup latency takes into account the network latency as well. We vary bot population and number of messages issued by each bot. Bots are placed randomly in Kad, as they would be in a real-world scenario. During our simulations each bot picks up a random id and simulates a lookup. We check if the lookup id is registered as a bot. If it is then it is considered as a successful lookup. Upon receiving a command, the bot simulates more lookups and the process continues until there are no more pending messages in the botnet.

The results of our simulation are shown in Fig. 4. Figure 4a shows the impact of varying % of bot population on command distribution time. The x-axis shows bot population as the percentage of total population. The y-axis shows command distribution time in seconds for reaching 75 % of bots. Each line in the figure represents different number of messages sent by a single bot. If a given line does not show a value for a given j% of bots, it means that with j% bot population and i messages, 75 % of bots are not reachable. Figure 4a demonstrates that increasing the bot population decreases command distribution time. This observation follows the intuition that a larger bot population will distribute commands more efficiently. Figure 4a also shows that increasing the number of messages sent by each bot increases the command distribution time. For example, for a bot population of 7 % or more, sending 50 messages from each bot is the optimal operating point. Sending 60 or 70 messages only increases command distribution time without much gain. Sending more messages from each bot increases the likelihood of discovering already contacted bots, thus wasting the message. Figure 4a also indicates that for 6 % bot population, at least 80 messages must be sent from each bot to reach 75 % of bots. Figure 4a also shows if the bot population is less than 6 % of total population, then 75 % of bots cannot be reached.

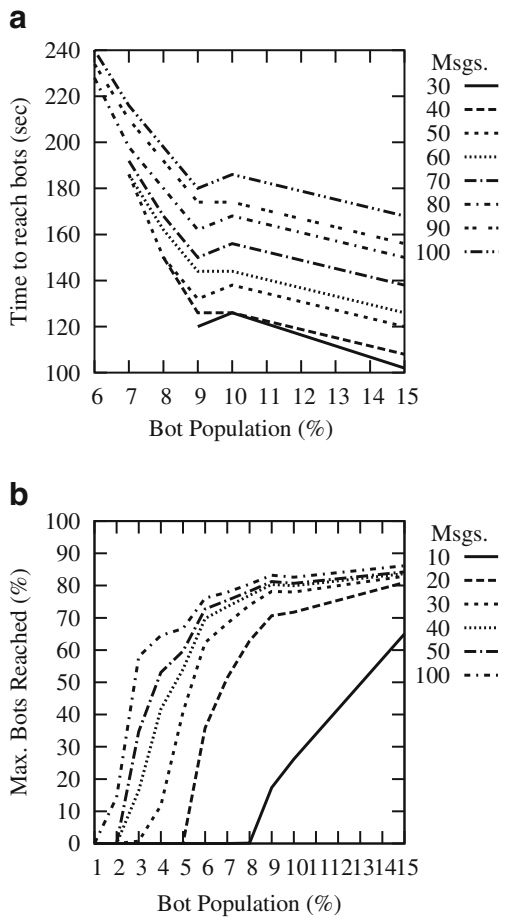


Fig. 4 Evaluation of command dissemination. **a** Impact of bot population on command distribution time. **b** Maximum number of bots reached

Figure 4b shows maximum possible bots that can be reached under the current churn model, while varying the bot population and number of messages sent by each bot. The x-axis shows bot population as the percentage of total population. The y-axis shows the percentage of bots that can be reached. Each line in the figure represents different number of messages sent by each bot. Figure 4b shows increasing the bot population increases the likelihood of reaching more bots. Figure 4b also shows that increasing the number of messages sent from each bot increases command reachability but to a certain point only. For example, if the bot population is 7 % or more of the total population, then sending more than 50 messages from each bot does not increase command reachability. This is because of the message overlap phenomenon described earlier. Finally, we note from Fig. 4b that under the current churn model, slightly more than 80 % of bots can be reached at best. Thus, if maximum bots need to be reached with minimum number of messages, then at least 9 % of total population must be bots and at least 50 messages must be sent from each bot.

In order to get an upper bound on bot performance we also conducted experiments with a constant bot population. The results of this experiment are shown in Fig. 5. This experiment was conducted with 100 messages per bot. We only varied the bot population. Figure 5 shows that as long as 5 % bots are present in the system, all the bots can be easily reached. As bot population increases, the time to reach all the bots decreases.

6.2 Performance of response reachability

Figure 6 shows how many bots are able to respond to a Tsunami botmaster. The x-axis shows bot population as percentage of total population. The y-axis shows those bots that receive a command and have a response-route to the

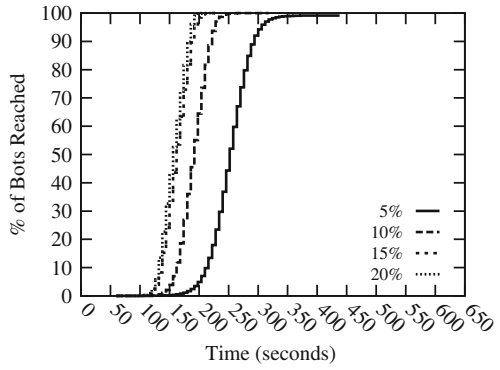


Fig. 5 Simulation results without churn

botmaster. Each line in the figure shows the number of messages sent by each bot. Figure 6 shows that for 30 or more messages, 99 % of the bots have a response route to the botmaster, and can successfully send the response. In other words, the bot population has no impact on response route when each bot sends 30 or more messages. Figure 6 also shows if each bot sends less than 30 messages, then response routes only exist when bot population is greater than or equal to 6 % of total population. If the bot population is less than 6 % and each bot sends less than 30 messages, then under the given churn model, no route exists from bots to botmaster.

7 Implementation of tsunami

In addition to evaluating the performance of Tsunami via simulations (Section 6), we demonstrate its feasibility via emulation over Kad.

We opted for emulation instead of a real-world implementation, because an implementation would disrupt the normal operation of Kad. An implementation would require deployment of modified Kad peers that can identify and propagate malicious commands. Such peers would artificially inflate the peer population, and their sudden departure

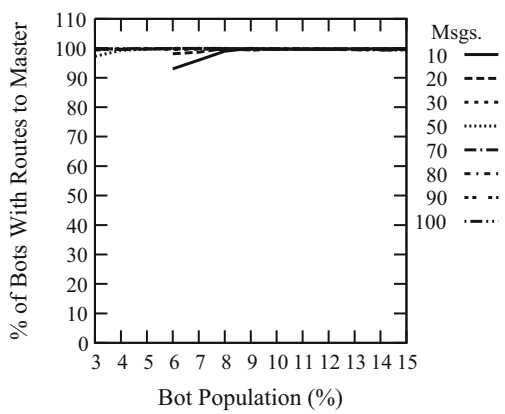


Fig. 6 Successful routes from bots to botmaster

at the end of the experiment would result in stale routing table entries.

In order to avoid above-mentioned problem, we developed an emulator that uses Planetlab [5] nodes as *bot emulators*, which inject malicious commands in the real-world DHT. Specifically, the emulator chooses random Kad peers as *artificial bots*, and assigns one bot emulator to each artificial bot. The botmaster injects a malicious command at a random point in the Kad DHT. As artificial bots receive malicious command traffic, bot emulators propagate commands in the DHT on their behalf. The use of Planetlab nodes helps us in avoiding the artificial inflation of Kad peer population, while still propagating malicious commands in the real-world Kad DHT.

The specific emulation steps are shown in Fig. 7, and described below:

0. Crawler crawls a *zone* of the Kad DHT once every minute, and discovers all the peers in that zone.

   We define a zone as a collection of Kad IDs that share the same x bit prefix. For example, a 12 bit zone, 0xacf, contains all the lookup IDs with 0xacf as prefix. For the emulation, we crawl a 12 bit zone instead of the entire DHT, in order to minimize the resource usage.

   The crawler discovers all the peers in a given zone by issuing lookups for all the Kad IDs in that zone, and finding all the closest peers. Finally, it chooses n% of discovered peers at random as artificial bots. Similar to simulations, we vary n from 1 to 15.
1. The botmaster injects a command in the Tsunami botnet. In a real-world implementation the botmaster would simply issue lookups, as described in Section 4.

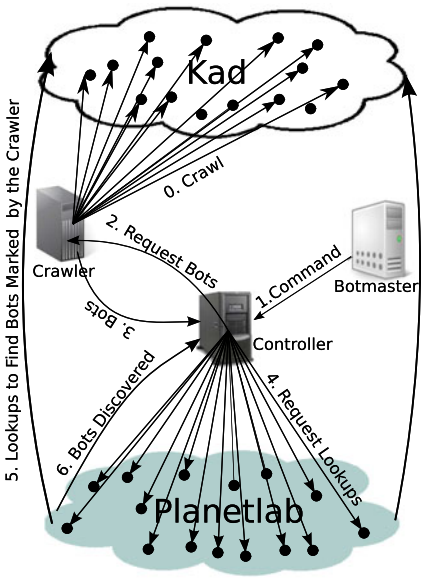


Fig. 7 Tsunami emulator design

However, for emulation, the botmaster requests the *controller* to issue lookups on its behalf. As the name indicates, the controller is the central component of the emulator that coordinates its activity.
2. The controller requests identities of currently chosen bots from the crawler.
3. The crawler sends the Kad ID, IP address and port number of currently chosen bots to the controller.
4. The controller chooses a random Planetlab node to perform lookups on behalf of the botmaster.
5. The chosen Planetlab node performs lookups in the crawled 12 bit zone, and sends the Kad ID, IP address and port number of all discovered peers to the controller.
6. The controller compares the discovered peers from the previous step against the bot identities received from the crawler. If a match is found, the controller assumes that a bot received the command and it will propagate the command further. For each match, the controller picks a random Planetlab node to perform lookups in the crawled 12 bit zone. This process continues until all lookups are complete on all Planetlab nodes and no more unique bots are found.

8 Defense against tsunami botnet

In general, defense against a botnet may use the following approaches: (i) patching the software or hardware vulnerabilities so that the botnet cannot recruit new bots; (ii) protecting probable targets [6] from a botnet attack; and (iii) disrupting the C&C of the botnet once it becomes active. In this paper, we focus on disrupting the C&C. Since unlike today's botnets, Tsunami does not employ bottleneck nodes for us to locate and shut down, we investigate how we may intercept C&C traffic of Tsunami in Kad.

In this section, we first briefly describe our generic technique for capturing DHT traffic. We then use this technique for defending Kad against Tsunami. After that, we evaluate the effectiveness of defense. Finally, we describe the weaknesses in the defense approach.

8.1 Capturing DHT traffic

In order to capture DHT traffic, we use *Montra* [12]. Montra introduces a large number of passive peers—called *monitors*—to capture Kad traffic. The monitors are passive because they do not issue any queries. A single monitor can either capture the traffic received by a single peer, or capture the traffic destined towards a single lookup ID. The two forms of capturing traffic require a different monitor placement strategy. In the next two subsections we describe how each form works.

Monitoring a single peer (peer monitor) In order to monitor a single peer P, Montra divides the lookup traffic that P observes into two types: *destination traffic* if P is the *closest* peer to the lookup ID, and *routing traffic* if P is *not* the closest peer to the lookup ID.

Montra focuses on capturing destination traffic only because it is more tractable. Montra captures a peer's destination traffic by exploiting Kad's *redundancy*. In order to minimize the effect of churn, Kad peers publish and search information at multiple peers that are close to the lookup ID. If a monitor M is placed close to P such that M is one of the closest peers to P's destination traffic, M will receive all of P's destination traffic.

We carefully choose the ID of M to control its placement. In our implementation of Montra, we set $ID(M) = 1 \text{ XOR } ID(P)$. As a result, P and M share a 127-bit prefix. Usually Kad peers share a 22-bit prefix [12]; thus M is the closest peer to P, and for all the traffic that is closest to P (i.e., destination traffic), M is the 2^{nd} closest peer. Such a placement ensures that M receives P's destination traffic.

Monitoring a single lookup ID (ID monitor) Monitoring a lookup ID is simpler than monitoring a peer. We simply set the ID of the monitor M same as the lookup ID in question, thus making M the closest peer to T. As a result, M receives all the traffic destined for T.

8.2 Protecting Kad against tsunami

In order to intercept Kad traffic that carry Tsunami commands, we must first *capture* it and then *redirect* it, as described in the next two subsections.

8.2.1 Capturing malicious traffic

We capture malicious traffic by monitoring all command-carrying lookup IDs (i.e., use ID monitors), as described in Section 8.1. In order to deploy ID monitors, we must find the command-carrying IDs in advance. From the analysis of the bot binary, we know that Tsunami bots issue lookups for 2^{22} lookup IDs, varying high-order 22 bits. Thus, we know all the 2^{22} prefixes. However, we do not know the 106 bit command suffix that the botmaster is currently using to issue the command. Without the command suffix, we cannot construct the lookup IDs and cannot deploy the monitors.

We find the command suffix by monitoring a large number of individual peers (i.e., use peer monitors), as described in Section 8.1. Since bots send commands to random peers, the larger the number of monitored peers, the higher the probability of intercepting malicious traffic. Research in [11] demonstrates that we can use one desktop machine to monitor 32,000 Kad peers with 90 % accuracy. Thus the

percentage of Tsunami's 2^{22} lookups that our monitors can receive is $\frac{Number of Monitors=32,000}{Total Population=2,000,000} = 0.016$, or 1.6 %. In other words, we can receive 67, 109 lookup messages on average—more than enough to find the command-carrying suffix (we only need one lookup for this purpose). The monitors decrypt all the intercepted traffic, using Tsunami's public key. By examining the decrypted traffic for command signature, the monitors finally discover the command suffix.

Upon finding command suffix, we construct 2^{22} lookup IDs by combining 2^{22} prefixes and one command suffix, and deploy 2^{22} ID monitors. It may seem resource-intensive to deploy such a large number of monitors. However, [11] demonstrates that we can deploy 2^{22} monitors using 64 desktop machines. Compared to the recent botnet defense proposals [6], which have advocated the use of a multi-million-node, non-malicious botnet to defend against a malicious botnet, 64 machines are negligible.

8.2.2 Redirecting malicious traffic

We redirect malicious traffic by sending it towards a *traffic sink*. The traffic sink consists of large number of modified Kad nodes, distributed across the world in order to prevent the identification of traffic sink nodes by IP prefix. These nodes pretend to be the closest peers to all the incoming traffic, preventing the traffic from going to any other peer.

The task of traffic redirection is best-suited for traffic monitors that capture malicious traffic. These monitors can mislead the traffic-sending bots and force them to send the traffic to the traffic sink. Specifically, they use false IDs for traffic sink nodes, such that those IDs appear to be closest to the lookup ID in the intercepted traffic. (The traffic monitors construct false IDs by using the same 30-bit prefix as the captured lookup ID, and choose the remaining 98 bits randomly. Usually a closest peer and a lookup ID share a 22-bit prefix [12]; using 30 bits is simply to be more safe.) As a result, the bots regard traffic sink nodes as the closest and stop sending further traffic.

8.3 Evaluation of tsunami defense

In this section we evaluate the effectiveness of our defense mechanism. Our evaluation metric is the percentage of requests that we successfully intercept and redirect. For evaluation, we emulate defense and C&C over the real Kad network, as described below.

8.3.1 Emulating defense

We emulate the defense by deploying ID monitors in an 8 bit zone of the Kad ID space. We deploy ID monitors by generating artificial command-carrying IDs. An 8-bit zone contains $\frac{1}{2^8} * 2^{22} = 2^{14}$ command-carrying lookup IDs. Thus, we generate 2^{14} ID prefixes. We assume that we know the 106-bit command-carrying suffix without monitoring a large number of Kad peers (Results in [12] shows that Montra can capture destination traffic with 90 % accuracy). By using 2^{14} prefixes and *one* constant 106-bit suffix, we generate 2^{14} lookup IDs and deploy 2^{14} ID monitors.

8.3.2 Emulating C&C

We emulate C&C over the real Kad network by issuing 2,000 command-carrying lookups towards the 8-bit monitored zone. We construct command-carrying lookup IDs as follows:

- We set the first 8 bits same as the prefix of the monitored zone, so that the lookups can enter that zone.
- We set the next 14 bits randomly, because the bots issue commands randomly.
- Finally, we use the same 106-bit command carrying suffix that we use for deploying ID monitors.

8.3.3 Results

The results of our evaluation are shown in Fig. 8.

Figure 8a shows the distribution of the number of common bits between requested ID and destination node under normal conditions. We borrowed this result from [12]. Figure 8a shows that for 20 % of requests, the closest possible node and the lookup ID has 19 bits in common; for the next 20 % of the requests, the closest possible node and the lookup ID has 20 bits in common; and the remaining 60 % of requests have 21 or more bits in common with the destination node.

Figure 8b shows the distribution of the number of common bits between requested ID and the last closest node that is contacted before the Montra monitors intercept the request and mislead the request-issuing bot. The figure shows that 70 % of the requests are captured at the 19^{th} bit. From this we can derive that 50 % of requests are captured before they reach their destination, since previously only 20 % of requests had destination at the 19^{th} bit. In addition only 10 % of the requests are captured at 21 or more bits. Under normal conditions, 60 % of the requests found their destination at 21 or more bits.

8.4 Limitations of proposed defense

Although our proposed defense intercepts and re-directs 50 % of traffic, it has the following weaknesses:

- We must know the command signature a priori. Tsunami may further use polymorphic commands to hide its traffic.

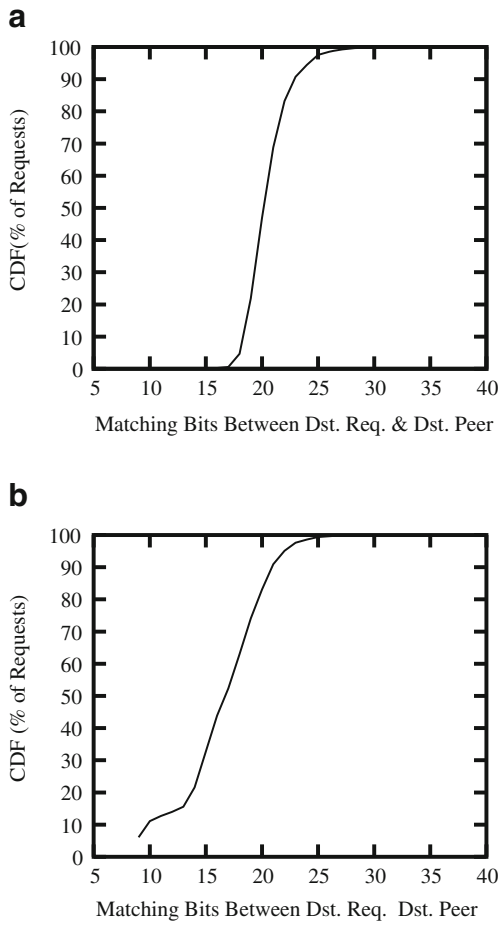


Fig. 8 Evaluations for defense against tsunami. **a** Number of bits matching between requested ID and destination node for regular traffic. **b** Number of bits matching between requested ID and destination node for traffic after adding montra monitors

- We capture traffic before it reaches destination, but bots at intermediate hops may still receive commands.
- The proposed defense only works for short commands that are carried by first-phase lookup messages. We cannot capture long commands that are carried by second-phase storage/retrieval messages.

9 Conclusion

This paper identifies and discusses a new type of botnet, called *Tsunami*. By building a parasitic relationship with a widely deployed system, Kad, Tsunami can successfully issue commands to its bots, launch various attacks, including distributed denial of service (DDoS) and spam, at ease, as well as receive responses from the bots, all without relying on any kind of central server or bootstrap nodes as current botnets do.

As we have demonstrated and evaluated using the Kad network, Tsunami is not only theoretically feasible, but also very real. In a Kad network with four million nodes and 6 % of them being embedded Tsunami bots, it can reach 75 % of its bots in just about 240 s. Meanwhile, defending a system such as Kad against Tsunami necessitates a systematic approach. As we discussed in the paper, a solid defense must be able to effectively intercept Tsunami traffic with a low cost, and do so without being intrusive to the system in question.

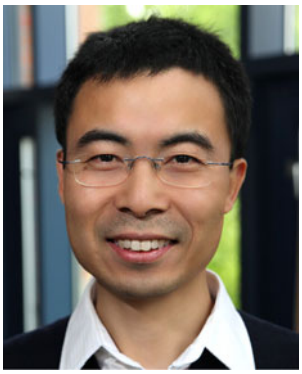
Acknowledgments We are extremely grateful to Sven Dietrich, Geoffrey Voelker, Peter Reiher and Jelena Mirkovic for their comments and suggestions on earlier drafts of this work. We are also thankful to anonymous reviewers of our earlier conference submissions, who helped in improving the clarity of this paper. Finally, we thank the members of Mirage and Netsec research groups at the University of Oregon for their continued feedback, comments and suggestions.

References

1. Alureon botnet. Website: http://arstechnica.com/security/news/2011/07/4-million-strong-alureon-botnet-practically-indestructable.ars
2. Analysis of phatbot. Website: http://www.secureworks.com/research/threats/phatbot/
3. emule. Website: http://www.emule-project.net
4. Machbot. Website: http://www.team-cymru.com/ReadingRoom/Whitepapers/2008/http-botnets.pdf
5. Chun B, Culler D, Roscoe T, Bavier A, Peterson L, Wawrzoniak M, Bowman M (2003) Planetlab: an overlay testbed for broad-coverage services. ACM SIGCOMM Comput Commun Rev 33(3):3–12
6. Dixon C, Anderson T, Krishnamurthy A (2008) Phalanx: withstanding multimillion-node botnets. In: NSDI'08: proceedings of the 5th USENIX symposium on networked systems design and implementation. USENIX Association, Berkeley, pp 45–58
7. Grizzard JB, Sharma V, Nunnery C, Kang BB, Dagon D (2007) Peer-to-peer botnets: overview and case study. In: Proceedings of the first conference on first workshop on hot topics in understanding botnets. USENIX Association, Berkeley, pp 1–1. http://dl.acm.org/citation.cfm?id=1323128.1323129
8. Holz T, Steiner M, Dahl F, Biersack E, Freiling F (2008) Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In: LEET'08: proceedings of the 1st usenix workshop on large-scale exploits and emergent threats. USENIX Association, Berkeley, pp 1–9
9. Kanich C, Kreibich C, Levchenko K, Enright B, Voelker GM, Paxson V, Savage S (2008) Spamalytics: an empirical analysis of spam marketing conversion. In: CCS '08: proceedings of the 15th ACM conference on computer and communications security. ACM, New York, pp 3–14. http://doi.acm.org/10.1145/1455770.1455774
10. Maymounkov P, Mazières D (2002) Kademlia: a peer-to-peer information system based on the xor metric. In: IPTPS '01: revised papers from the first international workshop on peer-to-peer systems. Springer-Verlag, London, pp 53–65
11. Memon G, Rejaie R, Guo Y, Stutzbach D (2009) Large-scale monitoring of dht traffic. In: IPTPS '09: proceedings of the 8th international workshop on peer-to-peer systems. http://www.usenix.org/events/iptps09/tech/full_papers/memon/memon.pdf
12. Memon G, Rejaie R, Guo Y, Stutzbach D (2011) Montra: a large-scale dht traffic monitor. Comput Netw 56(3):1080–1091

13. Rajab MA, Zarfoss J, Monrose F, Terzis A (2006) A multifaceted approach to understanding the botnet phenomenon. In: IMC '06: proceedings of the 6th ACM SIGCOMM conference on internet measurement. ACM, New York, pp 41–52. http://doi.acm.org/10.1145/1177080.1177086
14. Ripeanu M (2001) Peer-to-peer architecture case study: Gnutella network. In: First international conference on peer-to-peer computing. Proceedings, IEEE, pp 99–100
15. Rowstron AIT, Druschel P (2001) Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware '01: proceedings of the IFIP/ACM international conference on distributed systems platforms Heidelberg. Springer-Verlag, London, pp 329–350
16. Steiner M, Carra D, Biersack EW (2008) Faster content access in kad. In: P2P 2008, 8th IEEE international conference on peer-to-peer computing, Aachen. doi:10.1109/P2P.2008.28
17. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: SIGCOMM '01: proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications. ACM, New York, pp 149–160. http://doi.acm.org/10.1145/383059.383071
18. Stone-Gross B, Cova M, Cavallaro L, Gilbert B, Szydlowski M, Kemmerer R, Kruegel C, Vigna G (2009) Your botnet is my botnet: analysis of a botnet takeover. In: Proceedings of the 16th ACM conference on computer and communications security. ACM, pp 635–647
19. Stover S, Dittrich D, Hernandez J, Dietrich S (2007) Analysis of the Storm and Nugache Trojans: P2P Is Here. ;login: The USENIX Magazine 32(6):18–27. http://www.usenix.org/publications/login/2007-12/pdfs/stover.pdf
20. Stutzbach D, Rejaie R (2006) Understanding churn in peer-to-peer networks. In: IMC '06: Proceedings of the 6th ACM SIGCOMM conference on internet measurement. ACM, New York, pp 189–202. http://doi.acm.org/10.1145/1177080.1177105
21. Wang P, Sparks S, Zou CC (2007) An advanced hybrid peer-to-peer botnet. In: HotBots'07: proceedings of the first conference on first workshop on hot topics in understanding botnets. USENIX Association, Berkeley
22. Zhao BY, Kubiatowicz JD, Joseph AD (2001) Tapestry: an infrastructure for fault-tolerant wide-area location and Tech. rep., Berkeley

Ghulam Memon is a Ph.D. candidate at the University of Oregon. During his Ph.D., his focus has been the design, measurement and security of content-oriented networks (e.g., Kad, Rebus). From 2002 to 2004, Ghulam worked as a software developer for Synentia Karachi and Synentia Dubai. Ghulam also worked as a Research Programmer at the University of California San Diego from 2004 to 2006, where he designed and developed web-based analysis tools for geoscientists. Ghulam graduated with B.Sc. (Hons.) from the University of Huddersfield in 2002, and received a gold medal for maintaining the best GPA in his class. Ghulam has been a member of IEEE and ACM since 2006. Address: 120 Deschutes Hall, 1202 University of Oregon, Eugene, OR 97403-1202 Email: gmemon@cs.uoregon.edu



Dr. Jun Li is an associate professor in the Department of Computer and Information Science at the University of Oregon, and directs the Network Security Research Laboratory there. He received his Ph.D. from UCLA in 2002 (with honors), M.E. From Chinese Academy of Sciences in 1995 (with Presidential Scholarship), and B.S. from Peking University in 1992, all in computer science. In 2011 he is also a "Catedra de Excelencia" (Chair of Excellence) at the Carlos III University of Madrid, Spain, and a visiting researcher at the IMDEA Networks Institute in Spain.

Specialized in computer networks, distributed systems, and their security, Dr. Jun Li is currently researching Internet monitoring and forensics, social networking, future Internet architecture, and various network security topics. He studies both direct countermeasures against network security attacks (including Internet worms, phishing, and botnets) and fundamental security issues and solutions at the network architecture and protocol level (such as security for Internet routing, DNS, and peer-topeer networking). He has also done research on open architecture and programmable network as well as sensor networks.

He has published a book on disseminating security updates over the Internet and more than 30 peerreviewed papers. He has also served on several USA National Science Foundation research panels and more than 50 international technical program committees. He is a 2007 recipient of the prestigious NSF CAREER award, a senior member of ACM, and a senior member of IEEE.

Reza Rejaie received the M.S. and Ph.D. degrees from the University of Southern California, Los Angeles, in 1996 and 1999, respectively, and the B.S. degree from the Sharif University of Technology, Iran, in 1991. He is currently an Associate Professor at the University of Oregon, Eugene. From 1999 to 2002, he was a Senior Technical Staff Member at AT&T LabsResearch in Menlo Park, CA. His research interests include P2P networking, network measurement, multimedia networking and online social networks. Dr. Rejaie received an NSF CAREER Award for his work on P2P streaming in 2005. He has been a Senior Member of the Association for Computing Machinery (ACM) since 2006.