2016 IEEE 22nd International Conference on Parallel and Distributed Systems

Garlic Cast: Lightweight and Decentralized Anonymous Content Sharing

Chen Qian
UC Santa Cruz
cqian12@ucsc.edu

Junjie Shi
Nanjing Univ.
shijunjiechn@gmail.com

Zihao Yu
ICT, CAS
yuzihao@ict.ac.cn

Ye Yu
Univ. Kentucky
ye.yu@uky.edu

Sheng Zhong
Nanjing Univ.
zhongsheng@nju.edu.cn

***Abstract*—Anonymous content sharing in overlay networks protects user privacy and content confidentiality. Most overlay anonymous communication protocols employ some relay nodes as the proxies to forward content and require relays to perform cryptography or coding operations on messages. They have two major limitations. First, extra computation overhead may discourage overlay nodes from serving as relays. Second, long forwarding latency at relays makes an anonymous path easier to fail under network churn. In this paper, we present a lightweight and decentralized anonymous content sharing system named Garlic Cast, which requires near-zero computation cost on relays and does not rely on any centralized service. Garlic Cast uses random walks to find proxies in overlay networks and an security-enhanced Information Dispersal Algorithm to search and deliver content files. We have implemented a prototype of Garlic Cast and performed extensive simulation on real overlay topologies. Evaluation results show that the throughput of Garlic Cast is higher than that of RSA-based anonymous routing by over two orders of magnitude. Garlic Cast provides high level of anonymity and is robust to various attacks.**

I. Introduction

Overlay networks such as Akamai [1] serve as a major content delivery service for the Internet. The concept of overlay networks may also be extended to online social networks, which play a fundamental role in the diffusion of Internet information [2]. Anonymously searching and sharing contents are important for today's Internet users. For example, users having medical problems may wish to get helpful online contents from peer-to-peer (P2P) networks or online social networks (OSNs) without revealing any personal information, including their real and Internet IDs, IP addresses, and locations. An online survey collector needs to collect responses in the network that could be anonymous. In typical anonymous routing, the sender knows the destination of its messages. Anonymous content sharing is different in the sense that the communication initiator does not know the locations of content providers and both the initiator and provider may wish to be anonymous to the other side. Anonymous content sharing requires *mutual anonymity*.

The essential idea of anonymous communication is allowing the message to travel through multiple relays to hide the information of the source and destination. To preserve anonymity and message confidentiality, most, if not all of the existing protocols require that intermediate nodes perform cryptography or coding operations on message content. Chaum mixes [6], onion routing [7], and their variants [6] [7] [9] [8] [10] [11] require layered public key encryption and message decryption operations at relays. These operations introduce non-trivial forwarding delay and computation cost at each relay. In addition, public key dissemination either relies on a centralized and trusted public key infrastructure (PKI) or extensive message exchange between nodes. For overlay anonymous communication, a decentralized protocol is considered more practical. Cryptographic operations discourage overlay nodes from serving as relays. Moreover, extra operations on relays incur non-trivial latency and makes the message delivery easier to fail under network churn, i.e, nodes failure, leave, and join. It is desirable that an overlay-based anonymity system requires intermediate nodes to perform no more than simple message forwarding.

In this paper, we propose a novel decentralized anonymous content sharing system, Garlic Cast, which can utilize any overlay-based networks such as P2P systems and OSNs for Internet users. A user participated in Garlic Cast finds a group of "proxies" in one or more overlays. The initiator and provider will communicate via their proxies to achieve mutual anonymity. Like other practical solutions [14] [13], *Garlic Cast does not aim at perfect anonymity and confidentiality which may require tremendous resource to achieve. Instead, it is a lightweight and practical method that is possible to be deployed with current overlay networks.*

Garlic Cast has the following important properties.

- Garlic Cast is cost-efficient. It only requires symmetric cryptography on two endpoints of a communication session. Intermediate nodes perform message forwarding without any cryptography or encoding/decoding operation.
- Garlic Cast hides the information of two communication parties by creating random overlay paths between a node and its proxies. Each intermediate node on the path (including the proxies) only knows its predecessor and successor.
- Garlic Cast uses random walks to build anonymous paths. Unlike other random walk based anonymous systems [9] [11], a node running Garlic Cast does not need to know its k-hop neighbors ($k > 1$), and hence nodes do not reveal their neighbor list to others.
- To protect message confidentiality, a user uses a security-enhanced Information Dispersal Algorithm (IDA) to split its message into n segments. Only the message receiver can receive more than k out of n cloves and decode the original message.
- Garlic Cast is resilient to network churn because clove

1521-9097/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPADS.2016.35

delivery latency is short. The communication latency and throughput is close to simple overlay routing latency because relays only perform message forwarding. Cryptographic/coding operations on the two endpoints are not affected by network churn. The (k, n) IDA approach also introduces redundancy for robustness.

The balance of this paper is organized as follows. We discuss the related work in Section II. Section III introduces the system overview and model. We present the detailed design of Garlic Cast in Section IV. We analyze the security and anonymity of Garlic Cast in Section V. In Sections VI and VII, we use real implementation and simulation to evaluate the performance of Garlic Cast. We conclude this work in Section VIII.

II. Related Work

It is generally considered that Chaum mixes [6] and its variant onion routing [7] pioneered the modern anonymous communication systems. Tor [9] is a widely used implementation of onion routing. Many protocols utilize structured P2P systems for anonymity [8] [10] [11]. The resiliency to churn is a non-trivial concern for structured P2P systems. Information Slicing [14] employs linear network coding to achieve anonymous routing, which significantly improves the communication throughput. Rumor Riding (RR) [13] uses random walk to construct an anonymous channel for P2P file sharing.

Current protocols mainly have four limitations for overlay content sharing.

1) Symmetric or asymmetric key cryptography is used at every relay. Asymmetric key encryption/decryption need large cryptographic overhead. Symmetric key management, such as negotiation, distribution and group rekeying, is also expensive in large-scale overlay networks.
2) Single anonymous path is susceptible to relay failures under network churn. Using multiple paths can provide reliable message delivery. However, the maintenance cost is non-trivial, especially in dynamic environments.
3) Anonymous communications via static paths is vulnerable to timing and traffic analysis attacks [15] [16] [17].
4) Some network information, such as TTL and overlay neighborship, might be utilized by attackers to break the anonymity or message confidentiality.

To our knowledge, no method can resolve all of the above four problems.

III. Problem Overview and Model

Garlic Cast aims at providing anonymity for overlay-based content sharing. A user performs as a node in an application-based overlay. A pair of neighbors are two nodes connected based on established TCP links in P2P systems, friendship in OSNs [3], or user-to-user trust in an online community such as Advogato [4]. Two neighbors can exchange messages directly using various communication structures [12]. The node that is looking for a content is called the "initiator", and the node that is providing a content is called the "provider". An initiator first searches the network for an interested content and finds a content provider. Then the provider will deliver the content to the initiator. Such communication mode is essential for overlay networks such as P2P file sharing, video streaming, and information dissemination in OSNs. We also require Garlic Cast to be fully decentralized. There is no trusted third party or PKI.

Anonymous content sharing should satisfy the following security requirements: 1. *Initiator Anonymity*: the initiator cannot be identified by any other entities, *including* the provider. 2. *Provider Anonymity*: the provider cannot be identified by any other entities, *including* the initiator. 3. *Content Confidentiality*: the plaintext of the content is not exposed to any entities other than the initiator and provider.

Threat model: We consider an adversary that can collude a fraction of nodes (called malicious nodes) and launch collaborative attacks. Similar to other work [9] [14] [13], we do not assume the adversary can observe the traffic on all links. A malicious node can observe the network traffic forwarded by it. Malicious nodes can also collude among them by sharing observed traffic and other related information. The initiator or provider may also be a malicious node that wants to get the identity of the other communication party. However if the initiator or provider is malicious, the message confidentiality is not guaranteed. Similar to other work [13] [14], we assume the current methods to defend against the Sybil attack [29], such as Sybilguard [30]. We will discuss the robustness of Garlic Cast under various attacks in Section V-C.

IV. Garlic Cast

In this section we present the detailed design of Garlic Cast. We first introduce the security-enhanced IDA algorithm that protects message confidentiality. Then we describe the communication protocols for anonymous content sharing in Garlic Cast.

A. Security-enhanced IDA

The basic idea to achieve message confidentiality in Garlic Cast is splitting a message into multiple segments, each of which is in ciphertext. If the receiver gets enough segments, it can recover the plaintext message. In the literature there are two methods to achieve it, namely secret sharing [18] and Information Dispersal Algorithm (IDA) [19].

Garlic Cast uses Information Dispersal Algorithm (IDA), which is a k-threshold recovery method and each fragment is only l/k long plus a short piece of information with constant length. IDA is a well known example of Erasure Coding. It was first used for the distribution of some information among n processors, in such a way that any k of the processors can recover the information [19]. Unlike Secret sharing, IDA does not have the property "$k-1$ shares providing no information about the secret". It is possible that other peers holding i IDA fragments ($2 \leq i < k$) can recover partial original message.

We improve the security level of IDA by combining it with symmetric encryption, such as Advanced Encryption Standard (AES). To apply this security-enhanced IDA (S-IDA) [20] for a message M, the sender

1. encrypts M by an AES key K, getting $\{M\}_K$.

2. splits $\{M\}_K$ into n fragments, $M_1, M_2, ..., M_n$ by a k-threshold IDA.

3. splits K into n fragments, $K_1, K_2, ..., K_n$ by a k-threshold secret sharing.

4. constructs n messages called *cloves*, $C_1, C_2, ..., C_n$, where C_i includes two fields M_i and K_i.

5. sends the cloves to the receiver. We explain how the sending paths are constructed in the next subsection.

By receiving at least k cloves, the receiver:

1. recovers both the encrypted message $\{M\}_K$ using IDA and the key K by secret sharing.

2. decrypts $\{M\}_K$ and gets the original message M.

In this way, only when a peer collects k cloves, it can recover the key and decrypted $\{M\}_K$. Without the complete key, it can only see the ciphertext related to $\{M\}_K$. Note that the key is sent together with the message. Hence no centralized key management is needed.

B. Proxy Discovery

Before searching and sharing content, Garlic Cast requires that each node finds a number of proxies using the proxy discovery protocol presented as follows.

The initiator prepares a message M which only contains one sentence: "Are you willing to be a proxy?". It uses IDA to split M into n cloves, with the threshold $k = 2$. It also generates an one-time random clove sequence number and writes it on all cloves. The initiator then sends the cloves to n different neighbors that may belong to different overlays.

A node stores all cloves it receives in its local memory and records the last hop (predecessor) and the next hop (successor) information of each clove. Note that storing cloves and predecessor/successor information requires state maintenance on each node. Garlic Cast allows that the local stored clove is in soft state and will expire after a time period. If the state is not used after a period of time, it will be automatically deleted. Upon receiving a clove from a overlay neighbor, a peer can search its local memory by the sequence number, checking whether it has received multiple cloves of a series. If it does, the node can recover M, as the threshold is only two. If there is no hit, the newly received clove is then randomly forwarded to one of the neighbors in the same overlay. An example is provided in Fig. 1, Step 1.

If a peer recovers M and volunteers to be a proxy for the unknown initiator, it just replies a short agree message to each neighbor that passes a clove of the initiator to it. The agree messages will return to the initiator by the reverse paths. To allow the initiator to identify agree messages from a same proxy, the proxy will write a random sequence number on each agree message. An agree message includes a hop count field to record the number of hops between a proxy and the initiator. The purpose to put this record to the agree message rather than a discovery clove is to let the initiator know the path length while avoiding malicious nodes to track the initiator location. Fig. 1 Step 2 shows this process.

After collecting the agree messages, the initiator knows there are multiple paths to the proxy. It then generates a proxy sequence number and a symmetric session key K. The initiator uses 2-threshold IDA to generate cloves for the session key K, writes the proxy sequence number in plaintext on each clove, and sends the cloves to the proxy in different

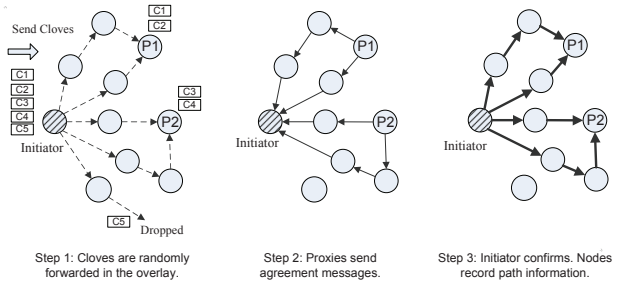


Fig. 1. An example of proxy discovery

paths, as shown in Fig. 1 Step 3. Every node on these paths uses the information of previous message route to deliver the confirmation, and records the proxy sequence number as well as the predecessor and successor.

Finally, we can expect that a number of nodes receive multiple cloves and become proxies. The initiator obtains a pool of proxies, each of which is identified by a proxy sequence number and has a unique session key. There are at least two anonymous paths between the initiator and a proxy. Every node on the paths only knows the predecessor and successor.

Note that a node only uses a neighbor for one anonymous path. If the number of proxies are not enough, the node will explore more proxies by sending cloves to more neighbors rather than contact those neighbors that have already provided anonymous paths. This mechanism ensures that no two neighbors can collaboratively tell the initiator a lot of fake proxies.

In order to prevent them walking in the overlay endlessly, all nodes agree on a probability p. When a node receives a clove and finds no hit, the clove has probability $1-p$ to be forwarded further, and probability p to be dropped. When a node receives a same clove twice, it will change the successor to be the new next-hop node but does not change the predecessor. Hence there is no loop on the path between the initiator and a proxy. The reason to use a probability rather than a fixed time-to-live (TTL) is to defend against trace-back attacks. For example, if a malicious node receives a clove with a very high TTL, it can guess that the predecessor is the original sender. Since cloves in proxy discovery are cheap, dropping them does not waste a lot network resource.

In the proxy discovery process, the initiator and its proxies also exchange their public keys. Note the public keys will only be used to encrypt sequence numbers. They will not be used for encrypt the content data. Due to security considerations or anonymous path failures caused by churn, each peer can timely delete (add) proxies from (to) its proxy pool.

C. Overlay search

To search a content in overlays, the initiator I generates a query message Q including the content name or ID, and the IP addresses and session keys of a number of its proxies. Using IDA, I splits Q into n cloves with 2-threshold.

Assuming n is even and $d = n/2$, the initiator randomly picks d proxies from the proxy pool. For each proxy, the initiator sends two cloves to it by different paths. Hence these

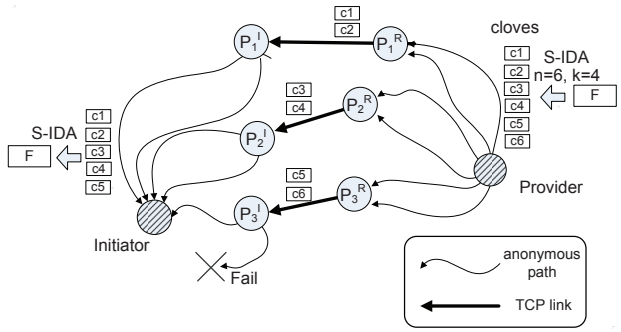


Fig. 2. Content delivery: The provider splits the content into 6 cloves, with a 4-threshold IDA. P_1^R, P_2^R, P_3^R are provider proxies, each of which can recover the IP address of an initiator proxy and sends the cloves to the initiator proxy directly. The initiator proxies P_1^I, P_2^I, P_3^I use anonymous paths to forward cloves. Finally the initiator recovers the content if it collects enough cloves.

proxies can recover the query message, and search the network on behalf of the unknown initiator by any searching methods such as central directories, distributed hash tables, or gossip-based search [22]. Garlic Cast does not limit the use of overlay routing protocols.

D. Content delivery

Suppose a content holder R receives one or more queries, and is willing to become a provider. Assume the proxies provided by the initiator in the query message are proxies $P_1^I, P_2^I, ..., P_{n'}^I$. The provider R generates cloves based on the requested content by the following steps.

1. encrypts the content F by an AES key K, getting $\{F\}_K$.

2. splits $\{F\}_K$ into n' fragments, $F_1, F_2, ..., F_{n'}$ by a k-threshold IDA.

3. splits the key K into n' fragments, $K_1, K_2, ..., K_{n'}$ by a k-threshold IDA.

4. for each initiator proxy P_i^I, the provider splits it IP address $IP(P_i^I)$ into two fragments, IP_{i1}, IP_{i2}, by a 2-threshold IDA.

5. generates a unique sequence number s for this content sharing. For each initiator proxy P_i^I, s is encrypted by the proxy session key, resulting $\{s\}_{K_{i,I}}$.

6. constructs n' garlic cloves $C_1, C_2, ..., C_{n'}$, each of which includes for fields: an IP fragment, an encrypted sequence number, a file fragment and a key fragment. A typical clove appears as $IP_{i1}||\{s\}_{K_{i,I}}||F_i||K_i$.

7. sends the cloves to d' proxies of the provider, denoted by $P_1^R, P_2^R, ..., P_{d'}^R$, where $d' = n'/2$. The provider sends two cloves to each proxy by two different paths, such that the proxy P_i^R will receive two cloves containing IP_{i1} and IP_{i2} respectively. For two cloves to a same proxy, the provider also attaches a same sequence number s' to notify that they belong to a same series.

After the provider proxy P_i^R receiving two cloves containing IP_{i1} and IP_{i2}, it can recover the IP address of a initiator proxy P_i^I. P_i^R directly sends the two cloves to P_i^I. When an initiator proxy P_i^I receives one or more cloves, it forwards the cloves to the initiator by the constructed anonymous paths. When the initiator receives a clove from a proxy P_i^I, it decrypts the sequence number s using the session key $K_{i,I}$. If the initiator receives no less than k cloves with the same sequence number s, it can recover the content file F.

Note that the sequence number s of this file delivery is always in ciphertext $\{s\}_{K_{i,I}}$. Such process is called *sequence number protection*. Even though the n' cloves have a same sequence number, it is encrypted by d' different session keys. Therefore only the provider, the initiator, and the initiator proxies can get the value of s. It guarantees that the initiator knows that the cloves belong to a same communication while intermediate nodes other than the initiator proxies do not know. Fig. 2 illustrates the file delivery of Garlic Cast. The content provider constructs six cloves for the requested content file F. Although one clove fails to reach the destination, five delivered cloves are enough for the initiator to get F.

V. Security Analysis

A. Confidentiality

To understand the level of confidentiality of Garlic Cast,, we briefly review the definition of packet independent security [23], [24]. Suppose x_1, x_2, ..., x_k are original messages, and y_1, y_2, ..., y_k are the corresponding cipher messages. Consider a function $y = f(X)$, where X is a combination of elements of $\{x_1, x_2, \ldots, x_k\}$. We say this function is packet independent secure, if the value of every single x_i is completely undetermined as long as one of the cipher messages is undetermined. This definition basically implies that to decrypt any message block x_i, an attacker needs to get at least k cipher messages. Similar to other k-or-nothing techniques [18], [23], [14], Garlic Cast is also packet-independent secure. Note that the number of cipher messages is more than k, but the k-or-nothing property still holds.

With sequence number protection, attackers can decode the messages only if they control no less than k initiator proxies. Even if attackers control nodes on k distinct paths, it still requires brute-force decoding of every possible combination of all communication messages forwarded by the attackers.

However, the confidentiality level of Garlic Cast is lower than public key cryptography based protocols such as onion routing. If the adversary colludes most of the nodes, it may be able to recover a Garlic Cast message. We do not believe such a case would become true in practice. According to our simulation results presented in Section VII-E, Garlic Cast can still achieve perfect confidentiality when the collaborative attackers are as many as 1% of the population.

B. Anonymity

We apply a commonly used entropy-based measurement [14] to evaluate the anonymity of Garlic Cast. The entropy of a system can be defined as follows:

Definition: Let S be the set of all nodes in the network and $|S| = N$. An attacker assigns each node x a probability p_x as being the source/destination of a message. The *entropy of a system* is defined as: $H(S) = -\sum_{x \in S} p_x \log_2(p_x)$.

The perfect anonymity happens when the attacker gains no information about the system. Hence it can only assign each

node equal probability $1/N$ to be the source. The entropy is then $H_{max} = \log_2(N)$. To evaluate the anonymity of a system, we normalize the entropy to be the measurement metric. The *anonymity of a system* is measure as: $\frac{H(S)}{H_{max}} = \frac{-\sum_{x \in S} p_x \log_2(p_x)}{\log_2(N)}$. Obviously the anonymity measure is in the range $[0, 1]$. Note that the anonymity value does not equal to the probability for the attacker to make a wrong guess. Suppose in a 10000-node system, the attacker can limit the source of every message in a subset of 100 nodes. Hence the probability that the attacker can make a correct guess is 1%. The anonymity measure of the system is

$$\frac{-(100 \times 0.01 \times \log_2 0.01 + 9900 \times 0)}{\log_2 10000} = \frac{\log_2 100}{\log_2 10000} = 0.5$$

Therefore the anonymity equal to 0.5 is not low, because the probability of the attacker to make a correct guess is only 1% in this system.

We use such metric to evaluate the anonymity of Garlic Cast. Suppose a source (e.g., the provider) is sending messages to a destination (e.g., the initiator). Let f be the fraction of collaborative malicious nodes. Some of them may sit on the k paths between the source/destination and the proxies. Suppose the number of all nodes on the k paths is L. There may exist multiple chains of consecutive attackers on the paths. The attackers may guess that the predecessor of every chain is the source and the successor of every chain is the destination. Note one of the two guesses must be wrong, because an endpoint of this path is a proxy. Hence the probability of a guess being correct is $\frac{k}{2(kL+k-s)}$, where s is the number of attackers on the k paths. Let Γ be the set of predecessors of all malicious chains. We then have,

$$\Pr(x = src) = \begin{cases} \frac{k}{2(kL+k-s)} & \text{if } x \in \Gamma \\ (1 - \frac{|\Gamma|k}{2(kL+k-s)}) \frac{1}{(1-f)N-|\Gamma|} & \text{otherwise} \end{cases}$$

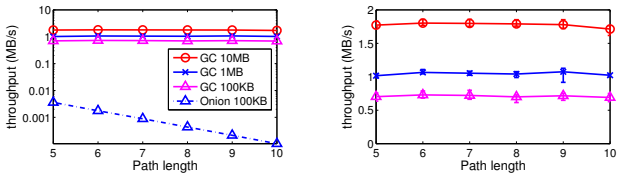
The entropy-based anonymity can be computed using the above probabilities. We will numerically evaluate the anonymity of Garlic Cast based on trace-driven simulation and compare it with Chaum mixes in the performance evaluation section.

C. Robustness to other attacks

In addition to the common collaborative attacks targeted on the confidentially and anonymity, we also analyze the robustness of Garlic Cast against a number of typical attacks.

Denial of service attacks: A malicious node can simply drop any message it receives. In such scenario, Garlic Cast decreases the vulnerability of an anonymous system to such an attack, compared with single-path approaches. It is because Garlic Cast has duplicate paths to deliver a message. To deny the delivery of a message, the attackers should sit on at least $n - k + 1$ distinct paths between the source and destination. The paths including no attacker can still be used for communication. For a typical single-path approach, one attacker in the middle can cause a message loss and force the sender to reconstruct a path.

In another denial of service attack, attackers keep sending faked messages to a node and make it busy forwarding



(a) Throughput comparison

(b) Throughput with 90 and 10 percentile bars

Fig. 3. Throughput of Garlic Cast and onion routing

cloves, denying other nodes to make use of it as relays. The duplicate paths (and relays) of Garlic Cast again decreases the vulnerability to such an attack.

An effective way to defend against denial of service attacks is to increase the network size [14]. Since Garlic Cast can bring many application layer networks together, it can easily increase the robustness of the system.

Traffic analysis attack: When collaborative attackers observe the traffic of the network, they can utilize traffic information such as traffic data, volume and patterns to build correlations between different nodes [26] [27] [28].

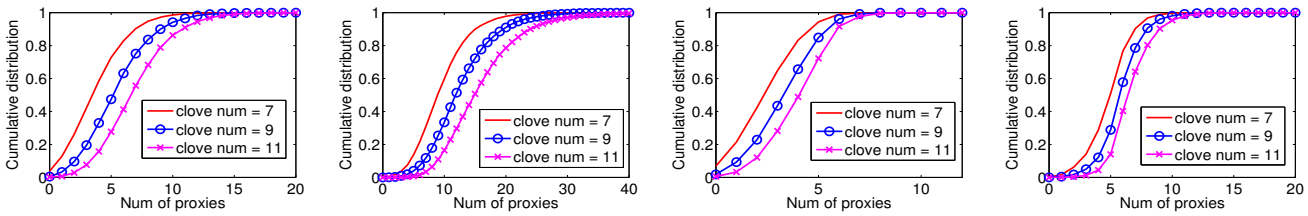
(I) Timing attack [26]: Attackers may add particular time delay between any two consecutive packets. When later other attackers also observe these delays among packets, they have the potential to location the transmission path. Garlic Cast is less vulnerable to the timing attack because there is no fixed transmission path between the initiator and the provider. For example, a proxy p_1 of the provider may send a clove to a random proxy p_2 of the initiator. Next time p_1 would communicate with a different proxy p_3 of the initiator to send. Hence only attackers controlling all possible paths between the initiator and provider can guarantee to observe the timing of cloves, which is hard to achieve in practise.

(II) Predecessor attack [28]: In this type of attack, attackers control a subset of nodes on a path and force to frequently rebuild the paths. They may be able to infer or identify the source and destination during such process. Compared with fixed-path approaches, Garlic Cast is less vulnerable to this attack because each source may select paths randomly from a relatively large proxy pool. Also a proxy pool may be reconstructed periodically.

Sybil attacks: Similar to most overlay anonymous communication protocols [13] [14] [11], Garlic Cast is vulnerable to Sybil attacks [29], because Sybil attackers may be able to attain the fraction of malicious nodes arbitrarily close to 1. The robustness of Garlic Cast can be improved by using Sybil-resistent social networks [30]. Like other work [13] [14] [11], we do not provide explicit solution for Sybil attacks and consider it as a separate open problem.

VI. Prototype Implementation

We have implemented a prototype system of Garlic Cast on six Dell PowerEdge R720 servers with Linux operation system to evaluate the throughput and cryptographic overhead of Garlic Cast. All servers are connected via a campus network. Each machine runs multiple threads to emulate multiple nodes



(a) Advogato $p = 0.1$ (b) Advogato $p = 0.05$ (c) Facebook $p = 0.1$ (d) Facebook $p = 0.05$

Fig. 4. Number of proxies for the Advogato and Facebook overlays

TABLE I
COMPARISON OF CRYPTOGRAPHIC OVERHEAD

Algorithms	Throughput (Mbytes/s)
AES Encrypt + IDA Dispersion	3.76
AES Decrypt + IDA Recovery	14.57
1024-bit RSA Encryption	1.16
1024-bit RSA Decryption	0.19

in an overlay. We configure the node deployment such that two neighbor nodes are in different machines. All cryptography algorithms were implemented with Crypto++ library 5.6.2.

We mainly measure the throughput of the content delivery phase, beginning with the provider's operations on the requested file and ending with the initiator's operations to successfully recovery the file.

A. Cryptographic overhead

We measure the overhead of IDA, AES and RSA on a file of size 100k using Crypto++ library 5.6.2. From the Table I, we can see that the cryptographic operations, IDA ($n = 6$, $k = 4$) and AES, in Garlic Cast are much efficient than RSA, the kernel operations in raw onion routing. More importantly, Garlic Cast requires no cryptographic overhead on intermediate nodes except the decryption/encryption of sequence numbers on proxies. It is known that the overall throughput is bounded by the throughput of a bottleneck. Hence the overall throughput of onion routing using RSA is less than 0.19 Mbytes/s while the overall throughput of Garlic is tens of times bigger.

B. File delivery throughput

We also measure the throughput to delivery files in different sizes using Garlic Cast and onion routing. We deployed some mini networks in our servers with different path length between the sender and receiver. Each server may serve different nodes by using multiple threads. We conduct each set of experiments for 10 times and take the average. In Garlic Cast experiments, we set the parameters as $n = 6$ and $k = 4$ and the proxy number for initiator and file holder both to be three. We also set the AES (ECB mode) key length to be 128 bits. For onion experiments, the RSA key length is 1024 bit and the padding scheme we adopted is Optimal Asymmetric Encryption Padding.

Figure 3(a) shows the throughput comparison of transmitting files of 100 KB, 1 MB, and 10 MB. We find that the throughput of onion routing decrease fast when the path length grows, which is a consequence of the layer-wise encryption and decryption. The throughput of Garlic Cast is over two orders of magnitude higher than that of raw onion routing. We also show the 10th and 90th percentile bars of the Garlic Cast throughput results in Figure 3(b). The throughput varies very little. According to Figure 3(b), increasing the path length does not result significant decrease of the Garlic Cast throughput. This is because the cryptographic operations contribute to most of the file delivery latency, which is mainly carried out on the two end points. Considering typical Internet host-to-host bandwidth and latency [32], the overall throughput of Garlic Cast to send a 1 MByte file on the Internet may still be close to 1 Mbytes/s.

VII. SIMULATION RESULTS

A. Methodology

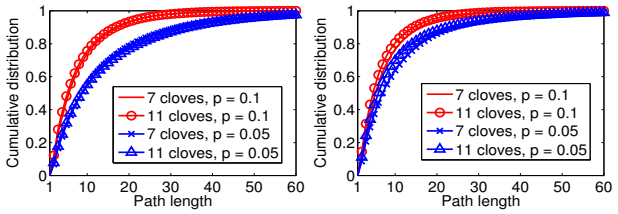
We conduct extensive simulation on real overlay network topologies. The first topology is the trust network of the Advogato online community [4] [5], where nodes are users and links represent trust relationships. The total number of users is 6,551 and the average degree value is 7.84. The second topology is a Facebook friendship network [31], where nodes are users and links represent a friendship between two users. The total number of users is 63,731 and the average degree value is 48.51.

Let the number of nodes in a network be N. We use f to denote the fraction of malicious nodes. The $f \cdot N$ malicious nodes can share information and perform collaborative attacks. Network churn is simulated by random selected node to fail. The node failure rate in a time duration follows Poisson distribution.

Performance criteria.

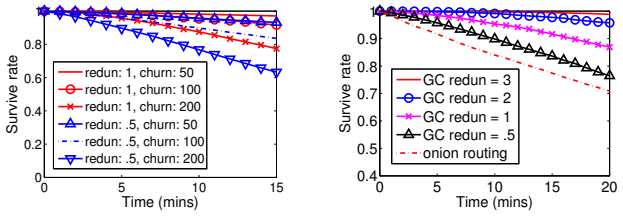
Number of proxies: the effectiveness of proxy discovery is measured by the number proxies a node can find in a overlay network. In the results we count the number of nodes that receive two or more cloves of an initiator. In practice, however, some nodes may refuse to serve as a proxy.

Path lengths: the path length is very important to an anonymous path. If a path is too short, any intermediate node has a very high probability to make a right guess about the sender or receiver. If a path is too long, network churn can easily break the path. Since an agreement message includes the hop count of the path, nodes can actively select those paths with proper lengths. However we need to know the availability of these paths using the proxy discovery protocol.



(a) Advogato (b) Facebook

Fig. 5. Path length for the Advogato and Facebook overlays



(a) Garlic Cast results (b) churn rate = 100 nodes/min

Fig. 6. Communication survive probability in Facebook under churn

Resiliency under churn: we evaluate the time that an anonymous path can exist in a network with different levels of churn. Node joins and leaves are all in Poisson distribution.

Anonymity: we use the entropy method presented in Section V.B to measure the system anonymity.

Confidentially: we measure the probability that the group of attackers can break the message confidentially. For Garlic Cast, only when the attackers compromise more than k initiator proxies, they can recover the message.

For results plotted in average or cumulative distribution, we conduct more than 10,000 experiments for each case.

B. Number of proxies

In this set of experiments, a random node will perform proxy discovery. We count the number of nodes that receive more than two cloves. Figure 4 shows the results in cumulative distribution. For the Advogato network, a node can find 3 to 10 proxies in most cases when the dropping rate $p = 0.1$, and it can find 6 to 20 proxies in most cases when $p = 0.05$. For the Facebook network, a node can find 2 to 5 proxies in most cases when $p = 0.1$, and it can find 4 to 10 proxies in most cases when $p = 0.05$. In all four sub-figures, using more cloves can get more proxies. Note that a node can use proxies from different overlays to send a same group of cloves. Hence users can always discover more proxies as long as they are participating multiple overlays. From Figures 4(a) and 4(c) we find a small fraction of nodes (around 5%) cannot find any proxy when $p = 0.1$ and the clove number is 7. We suspect those are inactive accounts which have very few neighbors.

C. Path Length

We show whether Garlic Cast can build paths in a variety of lengths for the initiator to choose. Figure 5 shows the cumulative distribution of path lengths. In Advogato most path lengths are in the range of $[2, 15]$ when $p = 0.1$, and in $[2, 30]$ when $p = 0.05$. When the value of p is smaller, path lengths are longer. The Facebook results are similar to those of Advogato. We also find that the number of cloves has very little impact to the path length. Note that we do not want to select very long paths, because using long paths consumes more network resources and they are easy to fail under churn. The initiator may intentionally not choose long paths since it can get the path length information. Combining the results of proxy numbers, we find that choosing a dropping rate between 0.1 and 0.05 may be proper. The proxy discovery can find enough number of proxies with appropriate path lengths.

D. Resiliency to churn

When there is network churn, an anonymous path connecting a node and its proxy may fail and the clove sent on the path will be lost. If less than the threshold k cloves are received by the receiver, a communication fails. Here we evaluate the resiliency of Garlic Cast in the Facebook overlay. When we say the churn rate is x nodes per minute, we mean there are 100 nodes fail and 100 nodes join the network within one minute. The fail and join times are in Poisson distribution. For cloves in (k, n) IDA where n is the number of cloves and k is the recovery threshold, we define the redundancy as $(n-k)/k$. For given churn rate, redundancy, and time period, we run experiments to send a group of cloves from a sender to a receiver, which repeats 1000 production runs. We define the survive rate as the probability of experimental cases in which there are still at least k paths alive.

Figure 6(a) shows the survive rate of Garlic Cast under different levels of churn (50, 100, and 200 nodes/min). When the redundancy is 1, Garlic Cast is quite resilient to churn. After 10 minutes, the survive rate is still above 90% when the churn rate is 200 nodes/min. Since Garlic Cast does not require cryptography operations on relays, we expect the end-to-end latency to send a clove is at most a few minutes. When the redundancy is 0.5, the survive rate decreases obviously. We compare the survive rates of Garlic Cast and onion routing in Figure 6(b). More redundancy offers stronger resiliency, but will potentially cause more traffic. Moreover onion routing requires public key decryption on each relay. It may need more time to finish the file delivery compared with Garlic Cast, and hence becomes more vulnerable to churn.

E. Anonymity and Confidentiality

In all experiments of this subsection, we generate a network of 10000 nodes. We vary the fraction of malicious nodes f from 0.001 to 0.9. For a node running Garlic Cast, n is the number of proxies to which it sends cloves, k is the threshold of S-IDA, and L is the average path length to the proxies. We assume nodes do not fail in these experiments. Similar to most other designs [14] [13], we compare Garlic Cast with onion routing [7], which can represent a large group of protocols. We choose the path length $L = 6, 8$ that are small values according to the results in Section VII-C.

We measure the anonymity by the entropy method presented in Section V.B. In Figure 7, we fix $n = 4$ for Garlic Cast and vary the path length for both Chaum and Garlic Cast.

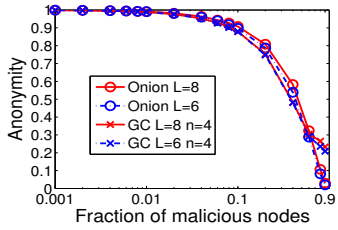


Fig. 7. Anonymity vs f, fixed n

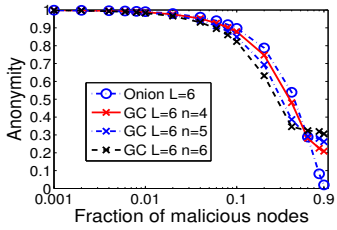


Fig. 8. Anonymity vs f, fixed L

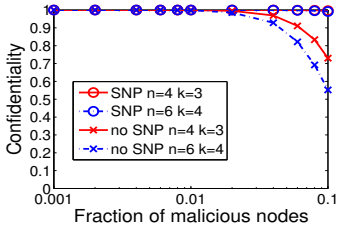


Fig. 9. Confidentiality with and w/o SNP

It shows that for $f < 0.7$, the anonymity of Garlic Cast is only slightly worse than that of Chaum. Increasing the path length can improve the anonymity values, but not significantly. Note that both protocols have very high level of anonymity: even if the anonymity is around 0.5, the malicious nodes still have a very low probability (could be less than 1%) to make a right guess. In Figure 8, we fix $L = 6$ and try different values of n. Similar to the results in Figure 7, Garlic Cast's anonymity is a little worse when $f < 0.7$. Interestingly, we find that smaller n provides better anonymity when $f < 0.7$, and worse anonymity when $f > 0.7$.

We measure the confidentiality by the value of $1-p$, where p is the probability that the malicious nodes can recover the original message. We evaluate Garlic Cast with and without sequence number protection, for $n = 4, k = 3$ and $n = 5, k = 3$. Figure 9 shows sequence number protection can evidently improve the confidentiality. When f is as large as 0.1, the confidentiality of Garlic Cast with SNP is still above 0.99.

VIII. Conclusion and Future Work

We propose a lightweight and decentralized anonymous content sharing system, Garlic Cast, based on overlay random walks and the security-enhanced IDA. Nodes can use any overlay networks such as P2P systems, content distribution networks, OSNs, and online organizations and communities. Garlic Cast provides mutual anonymity, confidentiality, and efficiency. The trade-off is that the anonymity level is a little less than that of onion routing. We have also implemented a prototype system and deployed it in a campus network. Extensive simulation results show that Garlic Cast provides high level of confidentiality and anonymity in various network environments.

Ongoing work on Garlic Cast includes deploying it on the Internet and improve its security levels. We will also investigate how Garlic Cast defends against more types of attacks such as the Sybil attack.

References

[1] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain, "Overlay Networks: An Akamai Perspective," in *Advanced Content Delivery, Streaming, and Cloud Services*, John Wiley & Sons, 2014.

[2] S. Myers, C. Zhu, and J. Leskovec, "Information Diffusion and External Influence in Networks," in *Proceedings of ACM KDD*, 2012.

[3] Facebook, https://www.facebook.com

[4] Advogato, http://www.advogato.org

[5] Advogato network dataset - KONECT, http://konect.uni-koblenz.de/networks/advogato, October 2013.

[6] D. Chaum, "Untraceable Electronic Mail Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, 1981

[7] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, 1999

[8] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: A Protocol for Scalable Anonymous Communication," in *Proc. of IEEE S&P*, 2002

[9] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," In *Proc. of USENIX Security Symp.*, 2004

[10] J. McLachlan, A. Tran, H. Hopper, and Y. Kim, "Scalable onion routing with Torsk," in *Proc. of ACM CCS*, 2009.

[11] P. Mittal and N. Borisov, "ShadowWalker: Peer-to-peer Anonymous Communication Using Redundant Structured Topologies," in *Proc. of ACM CCS*, 2009.

[12] P. Mittal, M. Caesar, and N. Borisov, "X-Vine: Secure and pseudonymous routing using social networks," in *Proc. of NDSS*, 2012.

[13] J. Han and Y. Liu, "Rumor Riding: Anonymizing Unstructured Peer-to-peer Systems," in *Proceedings of IEEE ICNP*, 2006

[14] S. Katti, J. Cohen, D. Katabi, "Information Slicing: Anonymous Using Unreliable Overlays," in *Proceedings of USENIX NSDI*, 2007

[15] S. J. Murdoch and G. Danezis, "Low-cost Traffic Analysis of Tor," in *Prof. of IEEE S&P*, Oakland, 2005

[16] L. Overlier, P. Syverson, "Locating Hidden Servers," in *Prof. of IEEE S&P*, Oakland, 2006

[17] S. J. Murdoch, "Hot or not: Revealing Hidden Services by Their Clock Skew," in *Prof. of ACM CCS*, 2006

[18] A. Shamir. "How to share a secret," *Communication of the ACM*, 1979

[19] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," in *Journal of ACM*, 1989.

[20] Hugo Krawczyk, "Secret sharing made short," in CRYPTO, 1993.

[21] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," *ACM SIGCOMM CCR*, 1997.

[22] X. Zhang, J. Liu, B. Li, and T. Yum. "CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming." in *Proc. of IEEE INFOCOM*, 2005.

[23] R.L. Rivest, "All-or-nothing encryption, in: Fast Software Encryption," in *Springer Lecture Notes in Computer Science*, 1997.

[24] D. Stinson, "Something about all-or-nothing (transforms)," in *Designs, Codes and Cryptography*, 2001.

[25] C.E. Shannon, "Communication Theory of Secrecy Systems," in *Bell Systems Technical Journal*, Vol. 28, pp. 656-715, October 1949.

[26] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing Attacks in Low-Latency Mix Systems,", in *Proc. Int'l Conf. Financial Cryptography*, 2004.

[27] Y. Zhu, et al., "Correlation-based Traffic Analysis Attacks on Anonymity Networks," in *IEEE Trans. Parallel and Distributed Systems*, 2009.

[28] M. K. Wright, M. Adler, B. N. Levine, and C. Shields, "The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems," in *ACM Trans. Information and System Security*, 2004.

[29] J. Douceur, "The Sybil Attack," In *Proceedings of IPTPS*, March 2002.

[30] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *Proc. of ACM SIGCOMM*, 2006.

[31] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," In *Proc. of WOSN*, 2009.

[32] S. Lee, P. Sharma, S. Banerjee, S. Basu, and R, Fonseca, "Measuring Bandwidth between PlanetLab Nodes," in *Proc. of PAM*, 2005.