Bitcoin Boomerang
Collaborative Anonymity for Transaction Broadcasting

Christopher A. Wood
Department of Computer Science
University of California Irvine
Email: woodc1@uci.edu

Chris H. Vu
Department of Computer Science
University of California Irvine
Email: ChrisHVu@gmail.com

***Abstract*—Sender anonymity is important in the design of several "cryptographically-enhanced" variations of Bitcoin, such as ZeroCoin [7]. Consequently, they often state that users install and use network-layer anonymity layers such as Tor [10] alongside their existing Bitcoin clients when introducing new transactions into the network. Such requirements make such variations less appealing from a usability perspective. Furthermore, since low-latency anonymity layers like Tor are susceptible to strategic timing attacks by adaptive global adversaries, and transactions do not have real-time constraints in which they must be broadcasted throughout the network, mixnets are a more appropriate solution for achieving sender anonymity. To this end, we introduce Bitcoin Boomerang, a distribtued, collaborative, application-level protocol for achieving mixnet-like behavior to increase sender anonymity. Boomerang is intended to run at the application-layer *within* Bitcoin software clients as an extension of the Bitcoin protocol. We analyze the anonymity properties of Boomerang, and assess the induced performance overhead using a custom simulator that emulates the behavior of Bitcoin nodes (software clients) adhering to the protocol. Our preliminary results indicate that with appropriate parameter tuning, increased sender anonymity with a single software client can be achieved for the small price of increased computational overhead and network bandwidth consumption for each participating Bitcoin user.**

I. Introduction

Bitcoin is a relatively young form of digital currency that has become increasingly popular in recent years. Its decentralized and peer-to-peer nature, in addition to its financial transaction security guarantees, makes it very appealing to alternative systems. However, providing adequate user anonymity when generating transactions remains an open problem that has motivated a significant body of work studying alternative constructions, protocol modifications, and general studies on Bitcoin anonymity. Of the most prominent deanonymization techniques are network- and protocol-level analysis. Kaminsky [5] pioneered network-level attacks on client anonymity by eavesdropping on network activity and associating the flow of transactions with the IP addresses from which they originated. This type of attack served to circumvent the now standard practice of generating fresh key pairs and shadow change (output collection) addresses for every transaction so as to ensure a degree of unlinkability among users and their transactions.

Protocol-level studies and attacks rely on the flow of transactions and other side-channel information in order to deanonymize clients. Even with the usage of anonymity layers such as Tor to hide their originating IP addresses, such flow-based techniques are highly effective at linking transactions to their original users. Using eWallets or mixing services to further break the link between users and their transactions is often recommended, but there are numerous problems associated with these partial solutions. Perhaps most importantly, such services need to be trusted to not disclose the identity of their clients or simply steal a users funds. Designing trustworthy and accountable mixing services has been studied several times in the literature, and the solutions to date either require modifications to the Bitcoin protocol [**?**] or have not yet been deployed to be adequately assessed in practice [14].

To circumvent the use of mixing services, others have proposed enhancements to the Bitcoin protocl that provide cryptographic guarantees of anonymity. Of these proposals, Zerocoin [7] appears to be the most feasible solution to implement in practice and see widespread adoption since it builds upon Bitcoin as a backing currency. Although there have been claims that its performance, which is based on RSA accumulators and a zero-knowledge proof system with expensive generation and verification of large proofs, will ultimately impeded its acceptance, enhancements such as PinocchioCoin [13] have been proposed to allieviate such issues. PinocchioCoin provides the same functionality as Zerocoin but uses more efficient pairing-based primitives and the Pinocchio verificable computation scheme to replace expensive zero-knowledge proofs. Also, while only briefly discussed in their respective publications, each of these "anonymous" coin schemes that build upon Bitcoin require a network-layer anonymizing layer such as Tor to prevent network-level attacks.

Based on these observations, it is clear that client anonymity at the network layer is fundamental for client anonymity at the level of the Bitcoin protocol. However, it would ideal if clients did not have to install or rely upon separate networking software to achieve this anonymity. To this end, we propose Bitcoin Boomerang, a protocol to emulate peer-to-peer mixnet behavior *built into* the Bitcoin protocol to provide similar anonymity guarantees as network-layer services such as Tor without the same vulnerabilities (e.g., timing side channel attacks). All Boomerang nodes, which are Bitcoin nodes adhering to the Boomerang protocol, must be connected to the Bitcoin P2P network as all nodes must be able to announce transactions. Due to this requirement, the design of the Boomerang protocol leverages the existing Bitcoin network to discover and connect to Boomerang nodes.

In this document we describe the design of Bitcoin Boomerang, henceforth referred to as just Boomerang for simplicity, and discuss the anonymity properties, expected performance, and preliminary implementation and simulation results that support our claims. The remainder of this document is outlined as follows. Section 2 provides a review of the Bitcoin network, Section 3 provides a detailed discussion of the Boomerang design, Section 4 presents an analysis of the security and anonymity that Boomerang enables, and finally, Section 5 discusses the expected performance of Bitcoin when using Boomerang for anonymity.

II. Review of the Bitcoin Peer-to-Peer Network

Bitcoin is built on top of a peer-to-peer network. To connect to the network, a local client uses four main methods of bootstrapping to locate an initial remote node. The first is the internal database of addresses the client saved during the previous session. If none of those addresses are active or the client is being run for the first time, the client uses a hard-coded DNS service to locate the address of seed nodes. The third method is via hard-coded addresses of seed nodes. The final method uses user inputted addresses from the command line or loaded from a text file. The Bitcoin network also previously used Internet Relay Chat (IRC) bootstrapping, but support for this method has been removed as of Bitcoin version 0.8.2.

To connect to a remote node, the client node first sends a `Version` message to the remote node. If the remote node accepts the `Version` message it replies with a `Verack` message and its own `Version` message. If the client node accepts the remote node's `Version` message, it replies with a `Verack` message. Finally, both nodes exchange `GetAddr` messages and `Addr` messages, which include the node's address information as well as no more than 2,500 addresses seen in the last 3 hours in the node's internal address database.

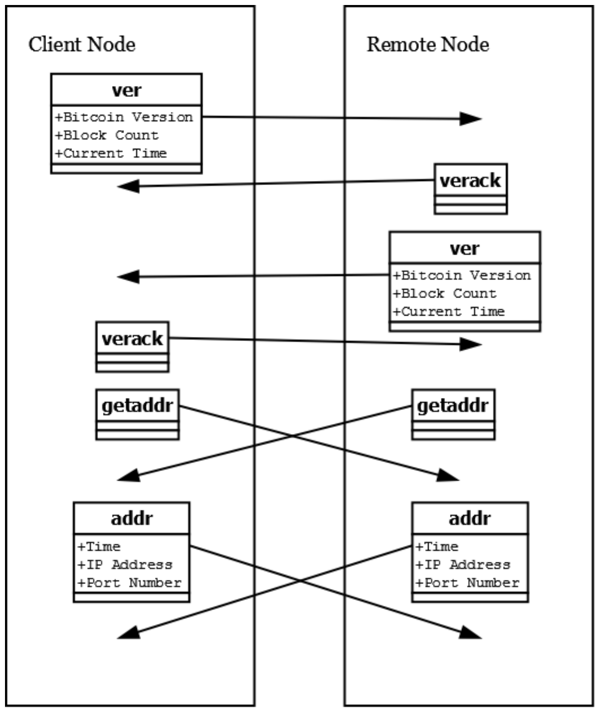

Fig. 1. Bitcoin connection setup.

After a node has joined the Bitcoin P2P network, peer discovery is propagated by callback addresses, `Addr` messages relays, and self broadcast. Callback addresses are addresses in the remote node's `Version` message that enable the local node to connect back. If the local node does connect back, then as described above, the local node will exchange `Addr` messages. Address relays occur after new addresses are added into the local node's internal database. The local node will pick two random nodes in its database and relay the new addresses in another `Addr` message. A node will also self broadcast, in which the node advertises its own address in an `Addr` message to all connected nodes.

III. Boomerang Overview

At its heart, Boomerang is a protocol to emulate a peer-to-peer mixnet that can be leveraged by any anonymizing coin, such as Zerocoin, to provide anonymity properties analogous to Tor without the same side channel vulnerabilities and need for an additional software component. The main idea is to hide the source or origin from which transactions are introduced into the network. To do this, we integrate traditional mixnet behavior into the peer-to-peer Bitcoin network so that new transactions which are to be broadcasted must first pass through a series of mixes (which are actually other participating Bitcoin users) in the network so as to obfuscate the originating network address. Functionally, this is not much different from Tor. However, Boomerang has several important distinctions that make it unique:

1) Transaction anonymity increases *for every participant* when more people use Boomerang messages to broadcast transactions - everyone is therefore incentivized to use Boomerang.
2) Involuntary mixing delays mitigate timing-based side channel attacks that can be leveraged to deanonymize clients using Tor.
3) Boomerang messages are enhancements to the Bitcoin protocol, rather than a means for anonymizing point-to-point TCP connections.

The main design goals of Boomerang are as follows:

Sender anonymity
A node achieves *sender anonymity* if a node cannot be (uniquely) identified, or linked, to a particular transaction that is broadcast at the end of a Boomerang circuit (see Section IV) [4].

Fault-tolerance via redundancy
Compromised or mobile nodes should not lead to significant delays in transaction broadcasts or induce traffic deadlocks.

Performance
Nodes in the network should incur minimal overhead from using Boomerang messages while maximizing their anonymity.

We now provide a brief overview of how Boomerang is used by clients for transaction broadcast anonymity; A complete description of the protocol is provided in Section 4. To broadcast a new transaction T using Boomerang, a client C first creates a set of W individual mixing circuits of length $D_1, D_2, \ldots, D_W$, where $W \geq 1$ and $D_i \geq 2$. Let $K = \sum_{i=1}^{W} D_i$ be the total number of nodes selected as mixing services for the circuits. With knowledge of the public keys for each of the K nodes, C then wraps T in D_i layers of encryption for each circuit $i = 1, \ldots, W$. Each layer of encryption also includes a forward pointer to the subsequent hop in the circuit so that decrypting nodes may forward the transaction to the appropriate location, or broadcast the plaintext transaction T if they are the last hop in the circuit.

As is standard with traditional mix networks, each nodes will only forward wrapped transactions after it has accumulated a certain number of transactions from other nodes [2]. For this reason, to avoid network deadlock, we require that "cover traffic" be circulated throughout the network to keep traffic moving smoothly, similar to the cover traffic used in the Tarzan mix network [3]. Furthermore, this cover traffic must be encoded such that it appears indistinguishable from legitimate encrypted transaction messages. To solve this problem, Boomerang cover traffic messages *are themselves legitimate transaction encryptions*, with the exception that the last hop of the circuit for these messages is the same client C that generated them in the first place. Thus, these dummy transaction messages will be circulated throughout the network and ultimately routed back to C, who can then easily check that they generated the transaction and discard it (or send out another dummy message). The cyclical flow of cover traffic, which is shown by the red trajectories in Figure 2, is the inspiration for Boomerang's name.

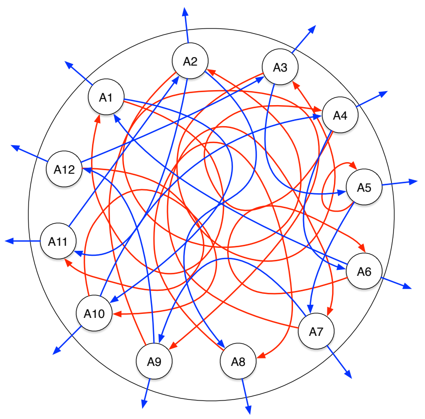

Fig. 2. Visual depiction of the traffic flow in a Boomerang network.

Given this short description, there are many important engineering problems to solve in order make Boomerang feasible in practice, such as public key distribution and usage, prevention of self-induced denial-of-service, and an overconsumption of network resources. We describe solutions to all such problems in Section 4, and include a preliminary performance analysis of the Boomerang technique (based on analytical modeling and simulations) in Section 5.

IV. Boomerang Design

As previously discussed, the Boomerang protocol enhancement for Bitcoin is motivated by the need to hide the source from which transactions are introduced into the network. Furthermore, this should be done in a transparent way so that any other form of anonymous coin extension on top of Bitcoin (e.g., Zerocoin) can leverage the service for transaction anonymity. Boomerang is *not* intended to support regular Bitcoin traffic; once a transaction is broadcasted and becomes public knowledge, Boomerang no longer plays a role in its distribution.

In the following sections we detail the core protocol and several important design and security tradeoffs that can be made in practice when using Boomerang. A formal analysis of the security and performance of Boomerang-enhanced Bitcoin is provided in Sections 4 and 5, respectively.

A. *Boomerang Peer-to-Peer Discovery*

Boomerang peer discovery is done in a similar manner to the Bitcoin peer discovery process described above with a few exceptions. Boomerang nodes can also query Bitcoin nodes after connecting to the Bitcoin network. Boomerang nodes can also learn the address of other nodes by routing Boomerang messages. The Boomerang peer discovery methods are:

1) User inputted addresses from command line or text file: If the user manually inputs address information, the client will attempt to connect to those Boomerang peers first.
2) Node reads stored Boomerang addresses from previous sessions: Boomerang clients will attempt to connect to peers stored in the internal address database from the previous session.
3) Nodes make a DNS request for Boomerang seed nodes: If there are no active nodes in the internal address database, then the client will perform a DNS lookup using hard-coded DNS servers for current seed nodes.
4) Nodes connect to hard-coded seed addresses: If DNS lookups fail, then the client attempts to connect to hard-coded seed addresses.
5) Nodes query Bitcoin network for Boomerang nodes: The client sends Boomerang connection requests to nodes on the Bitcoin network.
6) Nodes utilize callback addresses: After receiving a Boomerang connection request from a remote node, the client node can use the callback address to connect to the remote node and exchange address database information.
7) Nodes receive relayed addresses: After receiving new address information and verifying validity of the address, a node will randomly pick a few nodes in the internal database and relay the new address information.
8) Nodes will self broadcast periodically: Similarly to relayed addresses, periodically Boomerang clients will randomly pick a few nodes in the internal database and relay its own address information.
9) Boomerang messages: The destination of a Boomerang message must be a Boomerang node. The client can then request a connection with the destination node and exchange address database information.

B. *Transaction Encode and Broadcast Protocol*

At the heart of the Bitcoin protocol is the ability to encode new transactions as Boomerang messages and then ripple them throughout the network. We describe a high-levle overview of

TABLE I. BOOMERANG PROTOCOL NOTATION

Symbol	Description
W	Local number of parallel circuits constructed to emit a new transaction.
D	Global depth of transaction and dummy message circuits.
T	A boomerang transaction
$\bar{T}$	An encrypted boomerang transaction
M	A (dummy or encoded transaction) Boomerang message sent over the wire. A message is treated as an indexable array with fields shown in Figure **??**.
$\mathbf{C}$	Set of valid nodes in a node's address book from which to build a circuit.
AV	Address vector element of a Boomerang message.
addr	Network address of a Bitcoin node.
B	Minimum Boomerang buffer size.
Buffer	Boomerang message node buffer.
$E_{pk}(\cdot)$	ECC-based encryption of some plaintext under the public key pk.
$D_{sk}(\cdot)$	ECC-based decryption of some ciphertext under the private key sk.

the procedure for message encoding, EncodeTransaction, in Algorithm 1, where the notation contained therein is defined in Table I. The exact details of transaction encoding are given in Section IV-F, since they depend on the specific type of public-key encryption scheme. This procedure takes only two parameters - a transaction T to encode and a set of valid nodes $\mathbf{C}$ from which the circuit can be created, where nodes in the circuit are randomly sampled from this set (see Section IV-E for a more detailed description of how the address book of valid nodes is managed). An encoded Boomerang message has a very well-defined format, as shown in Table II, composed of seed element, encrypted address vector, encrypted message identifier (used internally during message validation), and the encrypted transaction. Details of these fields are given below:

1) A potentially re-encrypted seed. By the description of EncodeTransaction, it is required that the public-key encryption scheme used to mask these seeds has the same domain and range. This is needed because the decrypted seed for one hop is the encrypted seed for the next hop, very much like onion layers of encryption.
2) An encrypted address vector that is used by each hop to learn the next hop in the circuit without learning any other information about the nodes in the circuit. More specifically, a router can only learn about the immediate source and destination of a Boomerang message.
3) A potentially re-encrypted transaction message block. This block either stores the encrypted transcaction, where the encryption is done by XORing with a pseudorandom bit string generated by the decrypted seed value, or the plaintext transaction that is to be broadcast throughout the network.

This procedure is run again if the transaction T is not broadcasted within a certain time frame. We refer to this event as a transaction retransmission.

The asynchronous procedure to handle incoming Boomerang messages, BoomerangMessageHandler, is provided in Algorithm 3. This method will collect incoming Boomerang messages into the node's buffer, Buffer. When the number of messages in this buffer meets or exceeds B, the minimum buffer size, the BoomerangMessageForwarder asynchronous procedure will be invoked to shuffle the buffer, which is stored internally as a linked list, and then for each message peel off a layer of encryption on the transaction slot and transmit the result to the specified destination address acquired from the address vector AV.

Algorithm 1 EncodeTransaction($T, \mathbf{C}$)

```
1: for $i = 1$ to $W$ do
2:     $ID \leftarrow$ random unique ID
3:     Persist $ID$ to this node's internal state
4:     $(\bar{ID}||\bar{T}) := (ID||T)$
5:     $s \leftarrow$ random element from $\mathbb{F}_p$
6:     for $j = 1$ to $D$ do
7:         $p := \mathsf{PRG}(s)$
8:         $(\bar{ID}||\bar{T}) := (\bar{ID}||\bar{T}) \oplus p$
9:         $s \leftarrow E_{pk_{i,j}}(s)$
10:    end for
11:    $R \leftarrow$ random set of $D$ nodes from $\mathbf{C}$
12:    index := random even index from $[0, \ldots, 2D-1]$
13:    $\mathsf{AV} := [2D]$
14:    $\mathsf{addr}_0 := Nil$
15:    for $j = D$ downto $1$ do
16:        $\mathsf{AV}[\mathsf{index}] := E_{pk_{R[j]}}(\mathsf{addr}_{R[j]})$
17:        $\mathsf{index} := \mathsf{index} + 1$
18:        $\mathsf{AV}[\mathsf{index}] := E_{pk_{R[j]}}(\mathsf{addr}_{R[j-1]})$
19:        index := unpopulated index slot from $[0, \ldots, 2D-1]$
20:        $\mathsf{addr}_0 := \mathsf{addr}_{R[j]}$
21:    end for
22:    $M := \mathsf{Pack}(s, \mathsf{AV}, (\bar{ID}||\bar{T}))$
23:    $\mathsf{Transmit}(\mathsf{addr}_0, M)$
24: end for
```

Similar to the Tarzan P2P mixnet, a critical part of the Boomerang protocol is the inclusion of cover traffic that is indistinguishable from legitimate encoded Boomerang transaction messages [3]. This traffic is needed for two reasons: (1) to keep legitimate transactions moving through mixnet circuits (i.e., prevent deadlocks when no new transactions are being generated), and (2) to obfuscate the flow of legitimate transactions through the network. To support this cover traffic with minimal changes to the protocol, nodes in the network will *reuse* and *re-encode* old transactions to be sent throughout the network, with the exception that the destination node for the mixnet circuit (as specified in the EncodeTransaction procedure) will be the same as the sender. This is because the sender can easily discover when a transaction message has looped through the network and back to themselves, at which point they can then simply discard the transaction. Clearly, the rate at which this cover traffic is generated plays a critical role in the overall performance of the system when

TABLE II. BOOMERANG ENCODED MESSAGE FORMAT.

Field Size (bits)	Description	Data Type	Comments
256	Seed scratch	uint8_t[32]	This field is an element on the elliptic curve over the field $\mathbb{F}_p$.
$512D$	Encrypted address vector	uint8_t[D][64]	Encryption of the address vector used to define the circuit trajectory.
128	Message ID	uint8_t[16]	Unique identifier for a newly generated transaction.
$*$	Encoded transaction	uint8_t[]	Encoded transaction field with width equal to the maximum encoded transaction size.

Algorithm 2 BoomerangMessageHandler(j, M)

```
1:  $s := D_{sk_j}(M[0])$
2:  $(\bar{ID}||\bar{T}) := M[2] \oplus PRG(s)$
3:  if $\bar{T}$ is a well formed transaction then
4:      Broadcast($\bar{T}$) to the Bitcoin network
5:  else if $\bar{ID}$ is in this node's internal state then
6:      Drop the message
7:  else
8:      $s := M[0]$
9:      AV $:= M[1]$
10:     $i := 1$
11:     while $i <$ len(AV) do
12:         addr$_{src} := D_{sk_j}(AV[i])$
13:         if addr$_{src} =$ addr$_j$ then
14:             addr$_{dst} := D_{sk_j}(AV[i+1])$
15:             Buffer := append(Buffer, $(s, p, (\bar{ID}||\bar{T})$, addr$_{dst}$))
16:         else
17:             $i := i + 2$
18:         end if
19:     end while
20: end if
```

Algorithm 3 BoomerangMessageForwarder(j)

```
1: Lock Buffer (block new additions)
2: Shuffle Buffer
3: for $i = 1$ to |Buffer| do
4:     $(s, p, (\bar{ID}||\bar{T})$, addr$_{dst}$) := head(Buffer)
5:     $(ID'||T') := (\bar{ID}||\bar{T}) \oplus p$
6:     Transmit(addr$_{dst}$, $T'$)
7:     head(Buffer) := next(head(Buffer))
8: end for
9: Unlock Buffer (allow additions)
```

using Boomerang. We discuss the selection of parameters that achieve optimal performance without sacrificing security in Section 5.

C. Peer Discovery Messages

There are two types of Boomerang messages: peer discovery and maintenance messages, and encoded transaction and dummy (cover traffic) messages. The encoded transaction and dummy messages were already described in Section IV-B. Here, we describe the format of the following peer discovery and maintenance messages: `Version`, `VerAck`, `GetAddr`, and `Addr`. Every one of such messages is prepended with a standard header whose format is shown in Table III. The `Version` message is used to establish a connection with a remote node, relay address information about a remote node, and verify Bitcoin protocol compatibility. `Version` messages are composed of the standard message header and the fields listed in Table IV. In response to a `Version` message, `VerAck` messages are sent from the recipient to the original sender. These messages include the standard message header and the fields listed in Table V. The `GetAddr` messages are composed of just the standard header - the body of the message is empty. Intuitively, one may think of these as short messages broadcasted throughout the group. Finally, the `Addr` message contains information about active nodes on the network. An active node is a node that has been validated within the last 3 hours (this time parameter may differ on a per-client basis). The `Addr` message contains the standard message header and the fields listed in Table VI.

D. Node Connections

To connect to a remote node, the client sends a `Version` message, which contains Bitcoin network information, Boomerang version, and a hash of the client's public key. If the remote node accepts the information in the `Version` message, it will reply with a `Verack` message that contains the remote node's public key as well as its own `Version` message. The client will reply with a `Verack` message, which contains the client public key, if it accepts both messages from the remote node. Both nodes then add the address, public key, and time in their respective internal databases.

E. Node Validation

Boomerang clients draw from a pool of validated addresses for sending Boomerang transactions. If this pool is less than 1,000, Boomerang will begin to use dummy Boomerang messages to validate new addresses. The pool of addresses to be validated is equal to the number of validated addresses needed to reach the 1,000 address cap and is comprised of the most recently active non-validated addresses. If the pool of addresses is too small, Boomerang will begin sending `GetAddr` queries to increase the pool size. An address is considered non-validated if it either has an invalid validation timestamp or a timestamp older than 3 hours. When a remote node's address is added to the pool of addresses to be validated, the client must check if it has the node's public key. If the address was added from a relayed `Addr` message, the client must connect to the remote node in order to obtain the public key. If the connection could not be made, then the address is removed from the internal database and pool.

In order to validate addresses, dummy Boomerang messages will be routed through the addresses chosen from the pool of addresses to be validated as well as validated addresses. If the Boomerang message returns to the client and decrypts correctly, the nodes along the message's route are marked as verified with a second timestamp in the internal database. Both the first timestamp, which keeps track of when the node was last seen as active, and the second validation timestamp are updated with the time the message arrived plus a small random number (from 0 to an hour). Boomerang will also update the timestamps if a transaction message sent by the client is heard over the Bitcoin network. Boomerang clients will only relay

TABLE III. MESSAGE HEADER FORMAT.

Field Size	Description	Data Type	Comments
4	magic	uint32_t	Magic value indicating message origin network, and used to seek to next message when stream state is unknown
12	command	uint8_t[12]	
4	length	uint32_t	Length of payload in number of bytes
4	checksum	uint32_t	First 4 bytes of SHA256(SHA256(payload))

TABLE IV. `Version` MESSAGE FORMAT.

Field Size	Description	Data Type	Comments
4	`Version`	int32_t	Identifies protocol `Version` being used by the node
8	services	uint64_t	bitfield of features to be enabled for this connection
8	timestamp	int64_t	standard UNIX timestamp in seconds
26	addr_recv	net_addr	The network address of the node receiving this message
26	addr_from	net_addr	The network address of the node emitting this message
8	nonce	uint64_t	Node random nonce, randomly generated every time a `Version` packet is sent. This nonce is used to detect connections to self.
?	user_agent	var_str	User Agent(0x00 if string is 0 bytes long)
4	coin_version	int32_t	Identifies Bitcoin protocol `Version` being used
4	start_height	int32_t	The last block received by the emitting node
32	hash public key	uint8_t[32]	Hash of public key used by the node

TABLE V. `VerAck` MESSAGE FORMAT.

Field Size	Description	Data Type	Comments
256	public key	uint8_t[32]	Public key used by the recipient node

TABLE VI. `Addr` MESSAGE FORMAT.

Field Size	Description	Data Type	Comments
1+	count	var_int	Number of address entries (max: 1000)
30x?	addr_list	(uint32_t + uint8_t[32] + net_addr)[]	Address of other nodes on the network. Each address also includes timestamp of the last time the node was seen active as well as the hash of the public key the node uses.

address information from addresses verified in the last 3 hours and never reveal the validation timestamp.

F. Cryptographic Primitives

Based on the EncodeTransaction and BoomerangMessageHandler procedures, we require the following cryptographic primitives to support Boomerang messages:

1) Efficient chosen plaintext secure (CPA-secure) public-key encryption [1] with regards to computational complexity and bandwidth requirements.
2) Deterministic PRG whose input is an element in the range specified by the public-key encryption scheme.

To satisfy the first primitive for public-key encryption and decryption, Boomerang leverages the standard elliptic curve public-key encryption cryptosystem over the NIST-recommended field $\mathbb{F}_p$, where $|p| = 256$ (the size of prime p in bits) [**?**]. The domain parameters for the Boomerang encryption scheme are specified below:

- p - the prime number defining the finite field over which all elliptic curve operations are performed.
- a, b - the coefficients that define the elliptic curve - $y^2 = x^3 + ax + b \mod p$.
- G - the generator base point for the field.
- n - the order of the curve generator point G (i.e., the number of points on the field).
- h - the cofactor of the field (unused for encryption and decryption).

For completeness, we provide a brief (and modified) description of the setup, key generation, encryption, and decryption procedures used by the cryptosystem, denoted Setup, KeyGen, Enc, and Dec, respectively, as used in Boomerang.

- Setup: Generate and output domain parameters p, a, b, G, n and h.
- KeyGen: Select a random integer sk such that $0 < sk < n$ and compute $pk = sk \cdot G$. Output the public and private key pair (pk, sk).
- Enc(pk, m): Select a random integer r such that $0 < r < n$, compute $R = r \cdot G$, $S = r \cdot pk$ (S is the "secret" mask), $T = m \oplus \mathsf{DPRG}(S)$. Output the ciphertext tuple (R, T).
- Dec$(sk, (R, T))$: Compute $S = sk \cdot R = sk \cdot r \cdot G = r \cdot (sk \cdot G) = r \cdot pk$, $m = T \oplus \mathsf{DPRG}(S)$.

Traditionally, the element S is used to generate a symmetric key used to encrypt some message m. In our modified scheme, the element S *is used to generate a one-time pad* of the message m using a deterministic pseudorandom bit generator

[1]The observant reader may see that CCA-security would be better suited for this design since it more accurately models a real-world adversary who can maliciously send malformed Boomerang messages with chosen ciphertexts to an honest node and observe the decrypted seed value that is output. However, since nodes will accumulate messages prior to mixing, an adversary cannot be sure *which* output message corresponded to their input message, and therefore the likelihood of an adversary discovering the plaintext associated with their chosen ciphertext is small.

(DPRG), and R is used to recover the seed to remove this pad. Since r is chosen uniformly at random, the seed to the DPRG is uniformly random, and by definition the encryption $m \oplus DPRG(S)$ is CPA-secure. Using this encryption scheme, the seed and transaction slots are generated and modified as follows:

1) $(R_i, S_i) \leftarrow$ random elements in $\mathbb{F}_p$.
2) $(R_{i+1}, S_{i+1}) \leftarrow E_{pk}(R_i)$.
3) Save R_{i+1} in the seed slot.
4) $(\bar{ID}_{i+1}||\bar{T}_{i+1}) := (\bar{ID}_i||\bar{T}_i) \oplus \mathsf{DPRG}(\mathsf{S}_{i+1})$.
5) Save $(\bar{ID}_{i+1}||\bar{T}_{i+1})$ in the transaction slot.

Observe that, upon receipt of a Boomerang message, a node at hop i will compute $S_i = sk_i \cdot R_i$, and then using S_i it will compute $\mathsf{DPRG}(S_i)$ to generate the pad to unravel a layer of decryption on the transaction slot.

To satisfy the second primitive for the PRG, Boomerang leverages the counter mode deterministic pseudorandom bit generator (CTR-DRBG) algorithm as specified in [9] with AES-128 security strength (there is no security reason why this is chosen over Hash-DRBG or HMAC-DRBG). The CTR-DRBG takes as input a 256-bit seed to achieve AES-128 security (a seed derivation function could be used, but we omit this from the design). To use CTR-DRBG, a node must instantiate the state of the algorithm by specifying an entropy input, personalization string (fixed), and security strength parameter (optional, but assumed to be AES-128). To pump out pseudorandom bits, the update procedure is invoked using the state of the algorithm and the specified seed to produce the required number of output bits.

V. Boomerang Properties

In this section we elaborate on the anonymity and performance goals of Boomerang. We first discuss the anonymity properties that are provided by Boomerang, and then discuss system-level properties such as fault tolerance and performance.

A. Anonymity Properties

Inspired by Tarzan [3], we present a simplified analysis of the anonymity properties of Boomerang with respect to static and adaptive adversaries. In particular, we strive to show that senders achieve anonymity against a minority of colluding nodes. Before proving any claims, we present the main sources of information exposure in Table VII; a positive entry indicates that an attacker will be able to uncover the source of information, whereas a negative entry indicates that such exposure is not feasible given the Boomerang design.

One of the defining properties of Boomerang messages is that both encoded transactions and dummy messages are computationally indistinguishable. Therefore, an eavesdropping adversary cannot deterministically determine the type of messages based soley on passive observation, thus ensuring that the contents of each packet are not leaked in the network. Furthermore, even if an adversary successfully differentiates cover traffic from encoded (wrapped) transactions, an adversary cannot determine whether or not a compromised node is forwarding a transaction or is the original source of the transaction.

More formally, the size of the sender anonymity set for any particular message is exponential in the path length, i.e., in a network of N nodes comprised of N_{bad} nodes, there are $((N - N_{bad})/N)^i$ possible sending nodes (the size of the anonymity set) of a message at hop i that could have generated the original message, assuming uniformly random and unbiased circuit creation. As a result, the probability that a node n is the originator for a particular message m if intercepted at hop i in the circuit is $((N - N_{bad})/N)^{-(i)}$. Building upon the anonymity analysis completed for Tarzan, we may precisely quantify the confidence that a specific node in the anonymity set is the real sender as follows:

$$C_i = \frac{\Pr[H_i]}{E(|AS_i|) \times \Pr[H_{i+}]},$$

where $|AS_i| = ((N - N_{bad})/N)^i$ and $(\Pr[H_i]/\Pr[H_{i+}])$ is the probability that *some* node preceding the node at hop i is the sender. In this context, we use H_i to denote the event that first compromised node occurs at the i-th hop and H_{i+} to be the event that the first compromised node occurs somewhere after the i-th hop. Stated differently, $\Pr[H_i]$ is the probability that a message travels through $(i-1)$ uncompromised nodes prior to reaching the first malicious node. Analogously, $\Pr[H_{i+}]$ is the probability that a message traverses *at least* i honest nodes before reaching the first adversary. If the length of a circuit is D, this means that $\Pr[H_i]$ is equivalent to the product of $((N - N_{bad})/N)^{i-1}$ and (N_{bad}/N) (i.e., the i-th hop is compromised); similar computations hold for $\Pr[H_{i+}]$. Stated formally, these probabilities can be computed as follows:

$$\Pr[H_i] = \left(\frac{N - N_{bad}}{N}\right)^{i-1} \left(\frac{N_{bad}}{N}\right)$$
$$\Pr[H_{i+}] = \sum_{k=i-1}^{D} \left(\frac{N - N_{bad}}{N}\right)^{k} \left(\frac{N_{bad}}{N}\right)$$

A simple verification of this confidence equation can be seen by observing that as $N_{bad}/N \rightarrow 1$, $C_1 \rightarrow 1$ as well, which implies that the adversary's confidence in successfully deanonymizing the original sender is 1.0 if they compromise the entire network. However, by the assumptions of the Bitcoin network, the number of honest nodes will always be a majority of the total nodes in the network. Making this substitution, we see that the expected size of the anonymity set $E(|AS_i|)$ is bounded above by $(1/2)^{i-1}$, in the worst case, which means that C_i also reduces to the following:

$$C_i = \frac{1}{\sum_{k=i-1}^{D} \left(\frac{N - N_{bad}}{N}\right)^k}$$

If an active adversary is strategic in the selection of their nodes to compromise, one may naively try to compromise the start and end nodes of a circuit to identify the sender. However, one of Boomerang's defining characteristics is that a circuit is use *once* to broadcast a new transaction. Therefore, standard attacks such as packet relay, tagging, and reordering are ineffective since they all rely on persistent circuits present in anonymizing networks such as Tor [10].

B. System-Level Properties

From a systems perspective, Boomerang was designed with fault-tolerance and performance in mind. In particular,

TABLE VII. BOOMERANG INFORMATION EXPOSURE.

Information Exposed	*Bad Entrance Node*	*Bad Intermediate Node*	*Bad Exit Node*	*Bad Entrance/Exit Nodes*
Sender activity	Maybe	Maybe	No	Maybe
Sender content	No	No	No	Maybe

we (easily) claim that the Boomerang scheme is resistant to any adversarial attempt to leverage a successful denial of service (DoS) attack on the network. By the assumptions of the Bitcoin network, a majority of the participating nodes will always be honest (i.e., effectively uncompromised). Now, assume that there are N total nodes in the Bitcoin network, at least $N/2$ such nodes are honest, and $N_{bad} \leq \lceil N/2 \rceil - 1$ nodes are compromised. By this fact, during node selection and circuit formation, D nodes will be drawn at random without replacement, meaning that the probability of forming a circuit with at least one corrupt node $\Pr[\mathsf{BadCircuit}]$, in the worst case by the union bound, is at most

$$\Pr[\mathsf{BadCircuit}] = \sum_{i=0}^{D-1} \left[\frac{\lceil (N-1)/2 \rceil}{N-1-i} \left(\prod_{j=0}^{i-1} 1 - \frac{\lceil (N-1)/2 \rceil}{N-j-1} \right) \right]$$

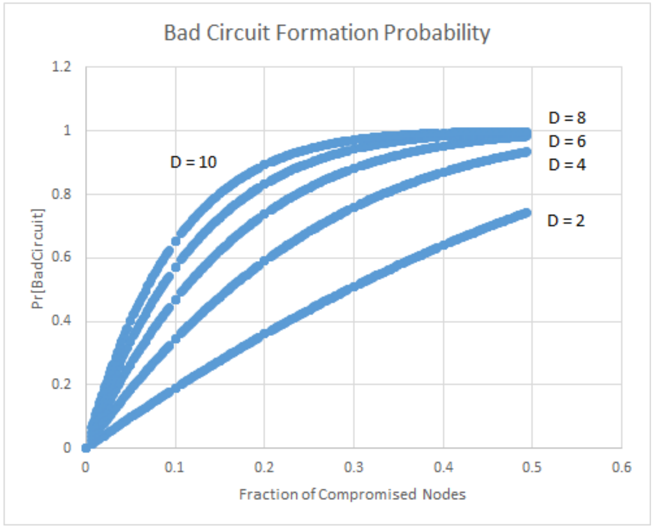

Fig. 3. $\Pr[\mathsf{BadCircuit}]$ behavior as a function of N and D (assuming $W = 1$). As expected, the probability of forming a bad circuit, i.e., one with at least one compromised node, grows logarithmically with the number of compromised nodes in the network.

Figure 3 illustrates the growth of $\Pr[\mathsf{BadCircuit}]$ as a function of the ratio of compromised nodes for various values for D. Observe that this probability quickly grows as $N_{bad} \rightarrow N/2$.

For reasonable measures of fault-tolerance, we require that this probability is kept as small as possible. However, for anonymity purposes, we require that D is maximized to achieve optimal mixing cascades (and thus, anonymity) throughout the network. To make this selection in an actual deployment of Boomerang, we would require some apriori knowledge about the expected number of compromised nodes at any given point in time. We are currently not aware of any means of acquiring or estimating this number. Therefore, assuming a reasonable majority of honest nodes, we recommend the selection of $D \in \{4, 5, 6, 7, 8\}$ in order to increase sender anonymity.

Note that it is possible for the adversary to introduce many invalid addresses into the network. In this case, to keep $\Pr[\mathsf{BadCircuit}]$ small, an honest node can use Boomerang messages to validate addresses so as to aggressively prune invalid addresses from their internal databases. In a similar fashion, Boomerang messages will prune nodes that improperly modify data that is routed through them. Due to the open nature of the Boomerang network, no solution to denial-of-service attacks are currently possible.

VI. Simulations and Performance Analysis

In this section we describe the implementation of our simulator and discuss some performance measurements acquired using this tool. We also describe features that should be added to the simulator to support more realistic experiments. We conclude with a discussion of good parameter selection based on our observations using the simulator.

A. *Simulation Design*

To assess the expected overhead introduced by Boomerang we implemented a custom discrete-time simulator that emulates the behavior of Bitcoin nodes (software clients) running Boomerang. Time in the simulation is measured in epochs; at every epoch a series of events occurs that advances the state of the system in some (usually) deterministic way. Our simulator supports the following behavior to closely resemble Boomerang:

1) Nodes enter and exit the network at random times.
2) Nodes make new transactions using configuration-specificed parameters W and D at a random rate π.
3) Nodes generate cover traffic at a random rate σ.
4) Nodes manage their internal address address books according the protocol described in section IV.

In addition, the simulation dynamically computes the following performance metrics:

1) Average number of "computations" done per node (i.e., the number of public-key encryption operations to encode a transaction).
2) Total and average message latency from the start to end of a circuit for single and every message, respectively.
3) Node forwarding throughput (messages/s).
4) Number of completed messages (transactions and cover messages) vs the number of in-progress messages.
5) Average number of transaction broadcast retries per node.

The parameters for a particular simulation are specified via a YAML configuration file which is parsed using the Java-based JYaml library [11]. An example configuration file which creates a simulation with $N = 100$ nodes, $D = 6$, $W = 2$, and cover and transaction generation rates uniformly distributed between $[1, 5000]$ and $[1, 7500]$ epochs (i.e., the most granular unit of time).

```
simTime: 2500
numNodes: 100
enterRate: 750
exitRate: 750
gridHeight: 10000
gridWidth: 10000
chaffGenRate: 5000
txGenRate: 7500
circuitWidth: 2
circuitDepth: 6
retryLimit: 7500
buffSize: 10
mixDelay: 50
pktSize: 1024
initialAddressSize: 250
validNodeTransmitReq: 50
addressBookSize: 1000
seed: 256
outfileprefix: "config-out"
path: "."
genMatrices: false
keepInMemory: false
```

The `genMatrices` and `keepInMemory` flags are used to ensure that the Java heap space is not exhausted from memory leaks by storing all of the events generated by the simulation at each time epoch. To run the simulation with 8GB of heap space on the example configuration listed above, which is stored in a local file `config.yaml`, one would run the following command:

```
java -cp ./jyaml-1.3.jar:. -Xmx8g Boomerang
                config.yaml
```

B. Performance Metrics and System Parameters

Using our simulation, we performed a series of experiments; the properties and simulation results for a small number of such experiments are summarized in Tables VIII and IX, respectively[2]. Contrary to what one might originally anticipate, there is no clear relationship between the number of messages (cover traffic and encoded transactions) generated and average number of transaction retries. As more messages flood the network, one would expect that the message flow to be expedited since mix node buffers fill up quicker, leading to a higher average number of forwarded messages. As a result, one might also naively expect that the likelihood of transaction retries would decrease because transactions would propogate through their respective circuits much quicker. However, the flow of traffic in the network is dependent on many variables, including the rate of cover and transaction generation, the distribution of node selection during circuit formation, the mix buffer size, and the random mix delay. Formulating a statistical model for the flow of traffic throughout the network is beyond the scope of this work, so we just note that more experiments should be conducted to gain a better understanding of how these system parameters interact with one another.

We can, however, state with relative certainty that the performance overhead of the Boomerang scheme is tightly coupled to W, D, and the rate at which cover traffic and new encoded transactions generation (σ and π, respectively). Furthermore, given our anonymity analysis and these performance results, it is clear that we wish to maximize D (circuit depth) and minimize W (circuit width - or the number of independent circuits) so as to reduce the overall work performed by a node while also improving anonymity. However, observe that with few independent circuits of larger depth, the likelihood that a transaction needs to be re-transmitted is increased. This result appeals to intuition since a larger number of hops will ultimately increase the overall message latency.

An illustration of cover and transaction messages flowing through the network during the entire duration of Experiment #1 is shown in Figure 4, and a series of smaller windows during which this information is captured is shown in Figure 5. As expected, the flow of converges to a random walks across all nodes in the network, even when analyzed in small time windows. To summarize, we recommend that D is maximized, W is minimized, and σ is maximized subject to node computational limitations and the expected congestion of the network. Choosing an appropriate value for σ should be tied to the expected transaction generation rate π, which is ultimately controlled by the users, i.e., it is not a system parameter. Unfortunately, we do not have the means to estimate this rate, and thus we leave the selection of the system parameters as future work dependent on such an analysis.

VII. Related Work

Tarzan's P2P network is much more secure than Boomerang, but may not function well in the Bitcoin environment due to scalability. Tarzan's peer discovery protocol involves gossip and validation. Nodes gossip network information by exchanging all known addresses and corresponding public key hashes. A node validates an address by directly connecting to it with a gossip message that includes a nonce.

The benefit of Tarzan's gossip procedure is that malicious nodes cannot respond with a different public key for every node in the network. Nodes learn public keys from gossip and not from the owner of the keys. Tarzan's validation method prevents the propagation of invalid address information throughout the network. Assuming the first node a client connects to in the Tarzan network is honest, the client will eventually learn of every honest node in the network. The Tarzan network is not as scalable due to the amount of communication required to set up and maintain the network.

Peer-to-peer distributed hash table (DHT) networks, such as Chord and Kademlia, use a distributed system to keep track of key-value pairs. DHT networks use a 160-bit key space. The Chord network has the keys arranged in a circle. Each node in the Chord network maintains a segment of the circle adjacent to it. A successor to a Chord node is the node that maintains the next set of keys. By searching linearly through

[2]Due to physical memory limitations and the initial single-threaded nature design of our simulator, we could not conduct experiments beyond $N \approx 25000$. We will address this shortcoming in our simulation design for future work.

TABLE VIII. SUBSET OF EXPERIMENTAL PARAMETERS EXPLORED WITH THE BOOMERANG SIMULATOR.

Experiment #	N	D	W	σ_{max}	π_{max}
1	50	6	2	5000	7500
2	100	6	2	5000	7500
3	150	6	2	5000	7500
4	200	6	2	5000	7500
5	250	6	2	5000	7500
6	1000	6	2	5000	7500
7	10000	6	2	5000	7500
8	50	8	1	5000	15000
9	100	8	1	5000	15000
10	150	8	1	5000	15000
11	200	8	1	5000	15000
12	250	8	1	5000	15000
13	1000	8	1	5000	15000
14	10000	8	1	5000	15000

TABLE IX. SIMULATION RESULTS GATHERED FROM THE EXPERIMENT CONFIGURATIONS LISTED IN TABLE VIII. SINCE MESSAGE LATENCY IS NOT A GOAL OF BOOMERANG (I.E., THE TIMELINESS OF TRANSACTION BROADCASTS IS NOT OF CRITICAL IMPORTANCE), THIS MEASUREMENT IS OMITTED FOR BREVITY. ALL SIMULATIONS WERE RUN FOR A

Experiment #	Avg. Chaff Generated	Avg. Transactions Encoded	Avg. Forwarded Messages	Avg. Retries
1	18.47	8.5	1.11	0.15
2	46.63	19.01	20.84	1.08
3	58.23	24.36	29.78	1.41
4	30.09	13.46	3.32	0.27
5	72.45	30.48	39.67	1.80
6	35.83	15.49	5.21	0.32
7	36.57	16.0	4.88	0.33
8	7.27	3.54	0.0	0.09
9	20.36	10.33	2.10	0.83
10	26.56	13.47	4.27	1.11
11	31.69	16.54	6.99	1.4
12	35.91	18.18	7.37	1.55
13	15.77	6.48	0.0	0.16
14	16.22	6.65	0.05	0.17

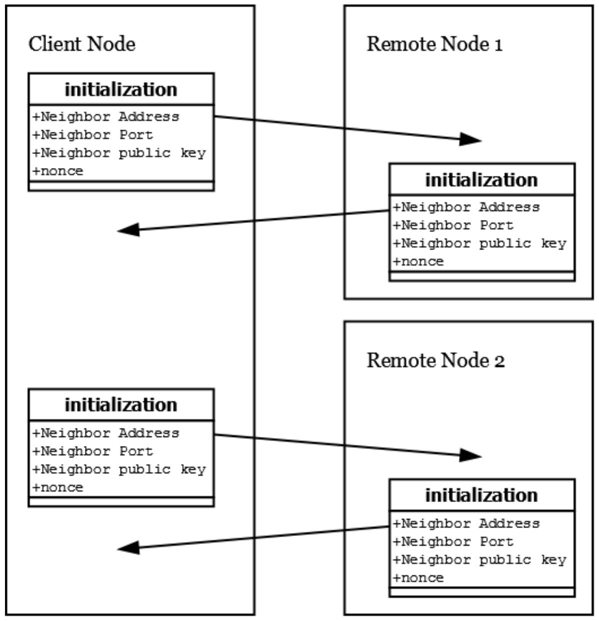

Fig. 6. Tarzan gossip and validation protocol.

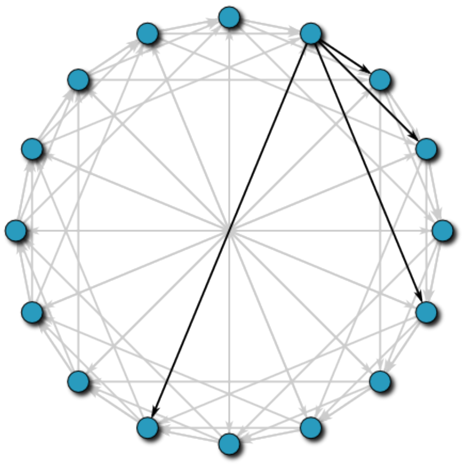

Fig. 7. A Chord network with “finger” table entries highlighted.

nodes, the node responsible for a key-pair can be found. To speed up searches, each node also keeps a finger table that contains successor information as shown below.

Kademlia DHT is similar to Chord except that it is organized as a tree and distance is calculated using the exclusive or operation. The exclusive or operation allows the nodes in the Kademlia network to have symmetric distances between two nodes. Kademlia's simpler distance calculation and iterative search, which is made possible by the tree-shaped network, allows for much faster searches than the Chord network. Both networks are decentralized, autonomous network that has fault tolerance and scalability. Unlike the Tarzan network, DHT networks do not prevent attacks on key searches or index poisoning and therefore allow a few malicious nodes to prevent nodes from finding other honest nodes.

References

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Consulted* 1 (2008).

[2] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM* 24(2) (1981), 84-90.

[3] Michael J. Freedman and Robert Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM conference on Computer and Communications Security*, ACM (2002).

[4] A. Pfitzmann AND M. Køohntopp. Anonymity, Unobservability and Pseudonymity A Proposal for Terminology. *In Federrath* 12, 1-9.

[5] Dan Kaminsky. Black Ops of TCP/IP Presentation. *Black Hat, Chaos Communication Camp* (2011).

[6] David L. Chaum. Blind Signatures for Untraceable Payments. *Crypto* 82 (1982).

[7] Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. *IEEE Symposium on Security and Privacy* (2013).

[8] Tor. Bitcoin Wiki. Available online at https://en.bitcoin.it/wiki/Tor. Last accessed: 1/25/14.

[9] Elaine Barker and John Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. 2012. Available online at: http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf. Last accessed: 2/10/14.

[10] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. *Naval Research Lab Washington DC* (2004).

[11] JYaml - Yaml library for the Java language. Available online at http://jyaml.sourceforge.net/. Last accessed: 2/10/14.

[12] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to Better – How to Make Bitcoin a Better Currency. *Financial Cryptography and Data Security*, Springer Berlin Heidelberg (2012), 399-414.

[13] George Danezis, Cédric Fournet, Markulf Kohlweiss, Bryan Parno. Pinocchio Coin: Building Zerocoin from a Succinct Pairing-Based Proof System. In *Proceedings of the First ACM Workshop on Language Support for Privacy-Enhancing Technologies*, ACM (2013).

[14] Joseph Bonneau, Arvind Narayan, Andrew Miller, Jeremy Clark, and Joshua A. Kroll. Mixcoin - Anonymity for Bitcoin with accountable mixes. Preprint available online at: http://cs.umd.edu/~amiller/mix.pdf.

[15] Darrel Hankerson and Alfred Menezes. NIST Elliptic Curves. *Encyclopedia of Cryptography and Security* (2011), 843-844.

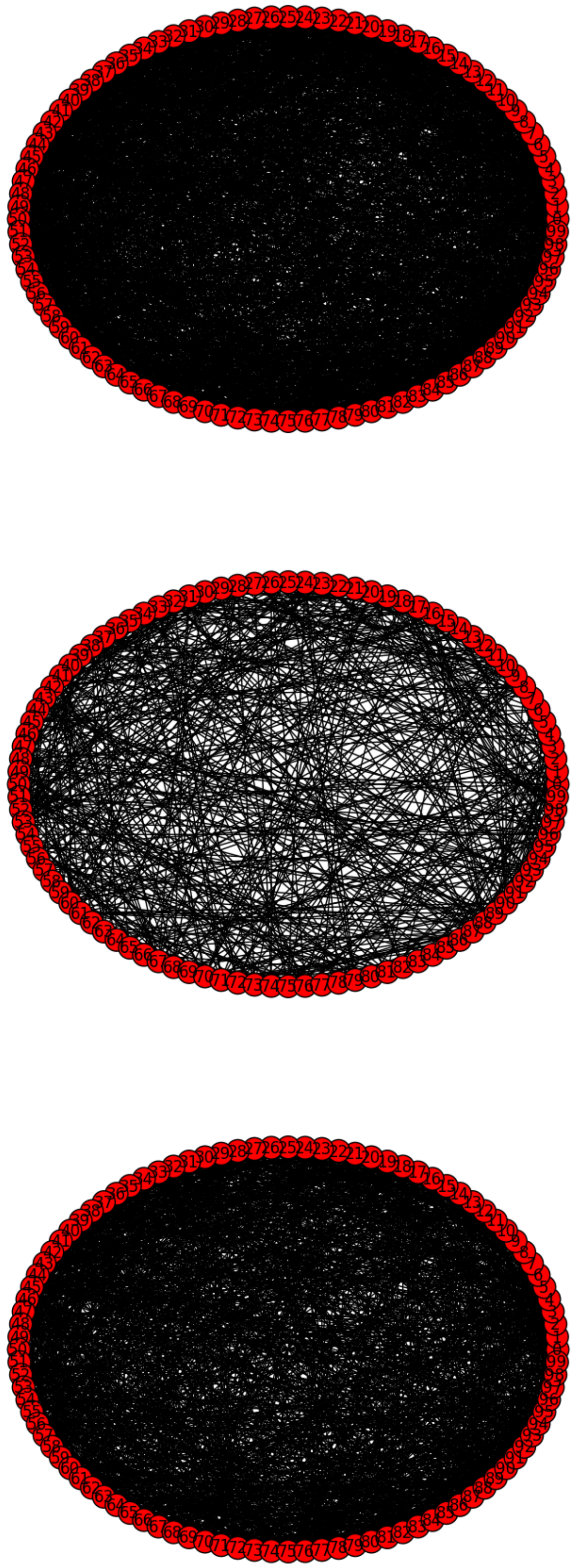

Fig. 4. The top figure shows the flow of both dummy messages and forwarded transactions from Experiment #1, the middle figure shows only the transaction messages, and the bottom figure shows the cover traffic. Nodes have directed edges between them if some message was sent between them during the lifetime of the simulation. The coverage of nodes is clearly uniformly distributed, as desired.

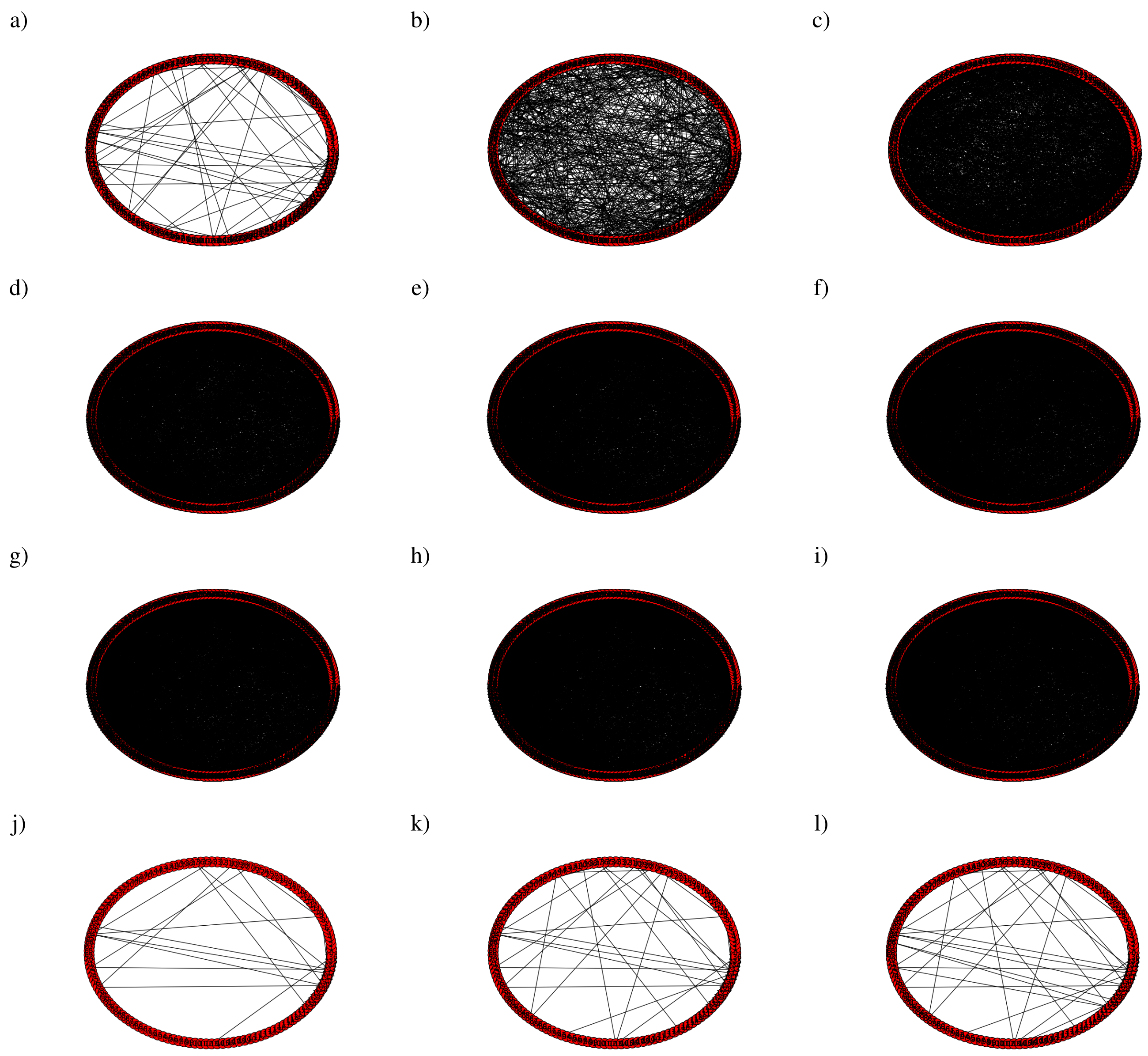

Fig. 5. Time series evolution of message flow in the Boomerang network. The flow of messages becomes quite uniform as the number of generated messages increases, which can be seen by the increased density of message flow and uniform message trajectories. The reduction in messages near the end is an artifact of the simulator in that messages stop being generated when the simulation time approaches the specified time limit.