A TorPath to TorCoin:
Proof-of-Bandwidth Altcoins for Compensating Relays

Mainak Ghosh, Miles Richardson, Bryan Ford[1], and Rob Jansen[2]

[1] Yale University, New Haven, CT
{`bryan.ford, mainak.ghosh, miles.richardson`}`@yale.edu`
[2] U.S. Naval Research Laboratory, Washington, DC
`rob.g.jansen@nrl.navy.mil`

Abstract. The Tor network relies on volunteer relay operators for relay bandwidth, which may limit its growth and scaling potential. We propose an incentive scheme for Tor relying on two novel concepts. We introduce *TorCoin*, an "altcoin" that uses the Bitcoin protocol to reward relays for contributing bandwidth. Relays "mine" TorCoins, then sell them for cash on any existing altcoin exchange. To verify that a given TorCoin represents actual bandwidth transferred, we introduce *TorPath*, a decentralized protocol for forming Tor circuits such that each circuit is privately-addressable but publicly verifiable. Each circuit's participants may then collectively mine a limited number of TorCoins, in proportion to the end-to-end transmission goodput they measure on that circuit.

1 Introduction

The Tor network suffers from slow speeds, due to a shortage of relay nodes from volunteers. This is a well studied problem, but despite many attempts, there is not yet a widely-adopted mechanism for compensating relay operators while retaining anonymity of clients [1–7]. This paper outlines one possible solution, embodying two complementary novel ideas:

1. **TorCoin** is an alternative cryptocurrency, or altcoin, based on the Bitcoin protocol [8]. Unlike Bitcoin, its proof-of-work scheme is based on bandwidth rather than computation. To "mine" a TorCoin, a relay transfers bandwidth over the Tor network. Since relays can sell TorCoin on any existing altcoin exchange, TorCoin effectively compensates them for contributing bandwidth to the network, and does not require clients to pay for access to it.

2. **TorPath** is a secure bandwidth measurement mechanism that utilizes decentralized groups of "Assignment Servers," extending Tor's existing "Directory Servers," to assign each client a Tor circuit that is publicly verifiable, but privately addressable. This mechanism allows TorPath to "sign" newly-minted TorCoins, so that anyone can verify a TorCoin by checking the blockchain.

2 Motivation and Related Work

Solving the problem of compensating Tor relays is attractive in that it might immediately improve the scalability of the Tor network. Prior research has not arrived at a fully satisfactory design for such an incentive scheme, however. We now outline what we believe to be the key requirements for such a scheme, while noting that more extensive discussion of these requirements and the architectural tradeoffs they entail is available elsewhere [9].

2.1 Requirements of an Incentive System

An incentive system must retain anonymity but verifiably measure bandwidth and reliably distribute payment to the nodes that provide it. The system must be resilient to adversaries attempting to identify clients or fake bandwidth transfer.

Preserving Anonymity: Among Tor's existing properties that TorCoin must preserve, no Tor client can recognize another, and no relay can identify the source and destination of any packet flow. Proposed incentive schemes like Tortoise [5] and Gold Star [3] may compromise clients' anonymity by allowing their traffic to be identified [6]. In a proportionally differentiated service [10, 11] like LIRA [6], a speed-monitoring adversary can potentially partition the anonymity set into clients that are paying for higher speeds, thus reducing anonymity. TorCoin should at least preserve the anonymity of the current Tor protocol, and ideally improve on it by attracting more clients and relays.

Verifiable Bandwidth Accounting: TorCoin needs to measure bandwidth in such a way that anyone can verify its measurements. Optimally, it will not require self-reporting or centralized servers, unlike EigenSpeed [12] or Opportunistic Bandwidth Monitoring [13]. The system should be robust to attackers or groups of attackers colluding to misreport bandwidth measurements, and the entire network should agree on all measurements. Rather than relying on reported network speeds, TorCoin uses an onion-hashing scheme to push bandwidth probe packets through Tor circuits to measure their end-to-end throughput.

Anonymous Payment Distribution: Once TorCoin measures the bandwidth a given relay has contributed to the Tor network, TorCoin must distribute payment to that relay in a way that preserves anonymity. Specifically, no one should be able to associate a bandwidth payment or measurement with a specific relay. Since TorCoin also requires verifiable accounting, the problem becomes how to verify bandwidth without identifying its provider.

Reliable Transaction Storage: TorCoin must store sufficient records of previous payments to avoid rewarding a relay twice for the same bandwidth transfer. Prior incentive schemes like LIRA [6] use a trusted central bank to assign coins and track spending, making the incentive scheme dependent on the central authority. TorCoin avoids relying on a central authority by using the Bitcoin protocol's distributed ledger to track transactions and avoid double spending [14].

Incremental Deployment: To simplify deployability, TorCoin should require minimal changes to the Tor codebase, and should not significantly increase latency of requests. A good incentive scheme should also be scalable to accommo-

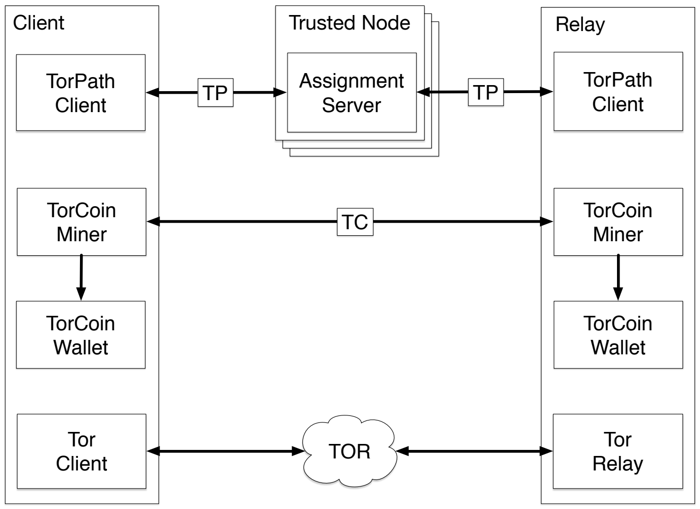

Fig. 1. High level TorCoin system architecture for clients and relays. A *TorPath Client* assigns Tor circuits to clients via the TorPath protocol, described in the next section. A *TorCoin Miner* mines TorCoins and stores them in a *TorCoin Wallet*. Each *Tor Client* and *Tor Relay* operates as usual, but on circuits assigned via the TorPath protocol.

date a large number of users and relays operating concurrently. TorCoin pursues deployability and scalability through its decentralized structure and small transaction overheads supported by existing technologies such as Bitcoin.

2.2 Key Technical Challenges

To illustrate the key challenge this paper addresses, we envision a naïve bandwidth-measurement scheme using blinded cryptographic tokens to signify bandwidth transfer. Suppose in this scheme, a client is able to give each relay a token for a given amount of bandwidth the relay provides. Relays are then able to convert these client-signed tokens into some form of currency. Such a scheme would be vulnerable to colluding groups of clients and relays, however, who can simply sign each other's transfer tokens without actually transferring any bandwidth.

We address this challenge via the TorPath scheme described below. TorPath restricts clients' ability to choose their own path, ensuring that *most* paths include at least one non-colluding participant (the client or at least one of the three relays). Assignment servers bundle large groups of clients and relays into *groups* that collectively choose paths. Even in the relatively rare event that a path constructed this way consists entirely of colluding clients and relays, an upper bound on the number of coins each path can mint rate-limits potential loss to these few, probabilistically rare, entirely-colluding paths.

3 TorCoin Architecture

The TorCoin architecture consists of two protocols, TorCoin and TorPath. In brief, the TorCoin protocol is a Bitcoin variant that mines coins, while TorPath protocol assigns a circuit (entry, middle, and exit servers) to each client, thereby "authorizing" the minting of TorCoins through verifiable proof-of-bandwidth.

TorCoin runs as a standalone service, requiring little modification to the Tor or Bitcoin codebase. Tor clients and relays operate as usual, except clients receive circuit assignments from *assignment servers*, instead of choosing relays arbitrarily from the Tor directory. Separately, a *TorCoin Miner* on each machine mines TorCoins by monitoring the throughput of the local Tor TLS tunnel, and communicating with its circuit neighbors via the TorCoin algorithm.

Figure 1 shows a basic overview of this architecture.

3.1 Adversary Model

We are primarily concerned here with an adversary who wishes to obtain TorCoins without contributing useful bandwidth to the Tor network. We assume the adversary is able to control a number of clients and relays. We assume that malicious clients and relays know about each other and are able to collude. We also assume that the adversary is able to control a minority of assignment servers on the network, and that other servers are honest-but-curious.

3.2 The TorPath Protocol

The TorPath protocol assigns Tor circuits to clients, replacing the usual Tor directory servers with *assignment servers*, which form decentralized *consensus groups*. The protocol guarantees that no participant on a circuit can identify all other participants, and that each circuit includes a publicly verifiable signature. We use TorPath to "sign" each TorCoin, so that anyone can verify a TorCoin's validity by comparing its signature to a global history of consensus groups.

Requirements: The TorPath protocol adheres to the following constraints:

- No client can generate its own circuit.
- Every circuit has a unique, publicly-verifiable signature.
- No client can know the circuit of another client.

Protocol Outline: The TorPath protocol consists of three sequential steps:

1. **Group Initialization.** Assignment servers form a *consensus group*. Clients and available relays provide public keys to join the group.
2. **Verifiable Shuffle.** The consensus group performs a decentralized, verifiable shuffle of all the public keys, resulting in a circuit assignment for each client.
3. **Path Lookup.** The assignment servers publish the result of the shuffle, such that each client can identify only its entry relay, and each relay can identify only its immediate neighbor(s) in the circuit.

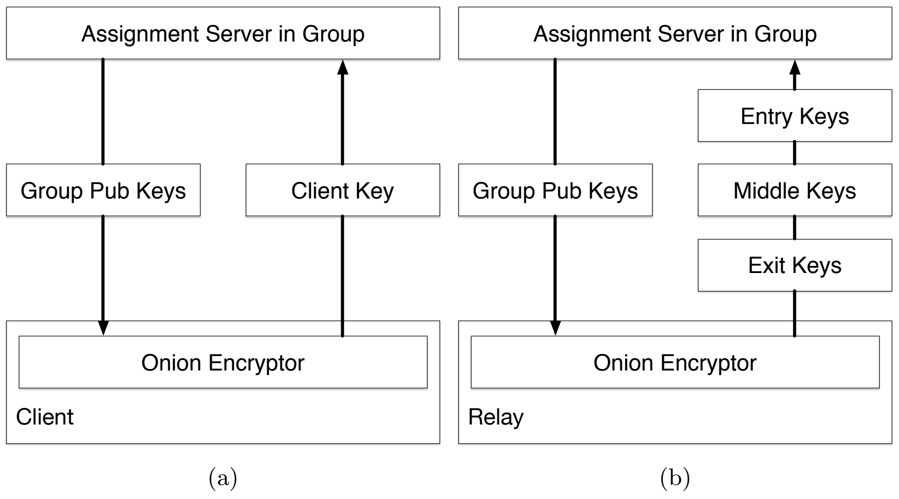

Fig. 2. Both relays and clients use onion-encryption to encrypt their own temporary public keys with the public keys of all the assignment servers in the group. (a) Each client generates one keypair, and sends its public key onion-encrypted to the server. (b) Each relay generates multiple keypairs, to support multiple clients and/or circuit positions within a consensus group, as instructed by the assignment servers.

Stage 1 – Group Initialization: A consensus group is formed when a suitable quorum of assignment servers come together to assign circuits to recently registered clients. For example, if there are 10 assignment servers in the network, we might require at least 6 of them to participate in forming each consensus group. The assignment servers can modulate the size – and hence anonymity – represented by each consensus group, by waiting until there are some configurable number n of clients registered before proceeding to the next stage. Different categories of consensus groups might use different values of n, allowing clients to trade anonymity for wait time. Groups with larger values of n would provide a larger anonymity set, at the expense of longer circuit setup times.

A consensus group forms in three steps:

1. **Each assignment server** shares its public key with its group members, and broadcasts these public keys to all clients and relays connected to it.
2. **Each client** connects to one assignment server in the group. The client then generates a temporary private and public key pair. The client onion-encrypts its temporary public key with the public keys of all assignment servers in the group, resulting in a ciphertext that each server can only partially decrypt. The client submits this ciphertext to its assignment server.
3. **Each relay** can act as an entry, middle, and/or exit relay, and it chooses which position(s) to service. The number of available relays available for a given position (especially exit relays) may often be less than the number of clients in the group needing circuits. To ensure parity between clients

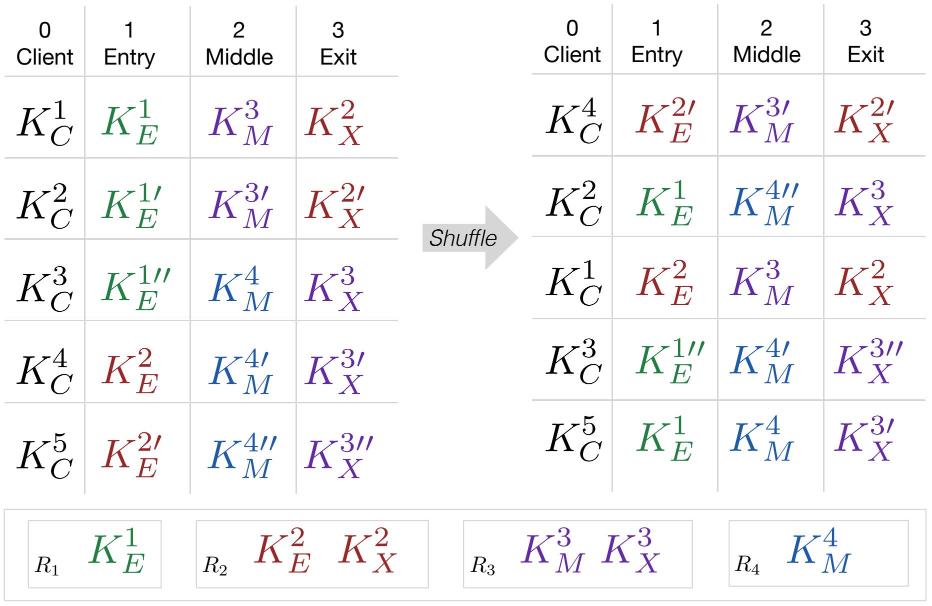
0 Client
1 Entry
2 Middle
3 Exit
K_C^1 K_E^1 K_M^3 K_X^2
K_C^2 $K_E^{1\prime}$ $K_M^{3\prime}$ $K_X^{2\prime}$
K_C^3 $K_E^{1\prime\prime}$ K_M^4 K_X^3
K_C^4 K_E^2 $K_M^{4\prime}$ $K_X^{3\prime}$
K_C^5 $K_E^{2\prime}$ $K_M^{4\prime\prime}$ $K_X^{3\prime\prime}$
Shuffle
0 Client
1 Entry
2 Middle
3 Exit
K_C^4 $K_E^{2\prime}$ $K_M^{3\prime}$ $K_X^{2\prime}$
K_C^2 K_E^1 $K_M^{4\prime\prime}$ K_X^3
K_C^1 K_E^2 K_M^3 K_X^2
K_C^3 $K_E^{1\prime\prime}$ $K_M^{4\prime}$ $K_X^{3\prime\prime}$
K_C^5 K_E^1 K_M^4 $K_X^{3\prime}$
R_1 K_E^1
R_2 K_E^2 K_X^2
R_3 K_M^3 K_X^3
R_4 K_M^4

Fig. 3. Example matrix shuffle with 5 clients (C_1, C_2, C_3, C_4 and C_5) and 4 relays (purple, blue, green, red). Each relay R_i generates a number of public keys K_p^i for each circuit position $p \in E, M, X$, as instructed by its assignment server. Here, each client C_j is assigned the circuit represented by the j^{th} row of the shuffled matrix.

and relays for each position, each assignment server instructs its relays to generate a sufficient number of temporary keys for each position. The relay server uses onion-encryption to generate n ciphertexts from n temporary public keys. The relay packages these ciphertexts by position and sends them to its upstream assignment server.

Figure 2 illustrates Steps 2 and 3.

We can represent the temporary keys as an $n \times 4$ matrix M for n clients, where each row corresponds to a client and its three relays (see Figure 3).

Stage 2 – Verifiable Shuffle: The consensus group now shuffles each column of the temporary key matrix independently, using a verifiable shuffling algorithm such as the Neff Shuffle [15]), and jointly decrypts the shuffled keys. Each row in the public result matrix now contains a random 4-tuple of temporary public keys representing one circuit: namely the client and the three relays serving that client. The assignment servers collectively sign and publish the resulting matrix to a public log, accessible by all clients and relays. Although everyone learns which four temporary public keys represent the participants of each circuit, the verifiable shuffle prevents anyone except a given key's owner from learning *which* participating client or relay owns each of these temporary keys.

Sender	**Message Tuple**
Client	$(K_C^i, \{IP_C^i\}_{K_E^i})$
Entry relay	$(K_E^i, \{IP_E^i\}_{K_C^i}, \{IP_E^i\}_{K_M^i})$
Middle relay	$(K_M^i, \{IP_M^i\}_{K_E^i}, \{IP_M^i\}_{K_X^i})$
Exit relay	$(K_X^i, \{IP_X^i\}_{K_M^i})$

Table 1. Each participant in a circuit sends its message tuple onion-encrypted to the server. $\{X\}_Y$ denotes X encrypted with Y.

Stage 3 – Path Lookup: In the final step of the algorithm, each client obtains the IP address of its entry relay, and each relay obtains the IP address of its neighbor(s) on the circuit. The path lookup algorithm ensures that each client and relay can obtain *only* the IP addresses of its neighbors in the circuit.

Each client encrypts its own IP using the public key of its neighbor. The client forms a tuple (public key, encrypted IP) as shown in table 1. The client onion-encrypts this tuple and sends it to the assignment servers. Each relay follows the same procedure, for every key in the matrix belonging to it.

The assignment servers shuffle this new list of tuples. Each client and relay now finds its neighbors in the matrix by locating the tuples containing the public keys it needs. Finally, each participant decrypts the relevant cells using its private key, revealing the IP address of its circuit neighbor(s).

Now all clients in the consensus group have a usable Tor circuit.

Each circuit formed by a consensus group obtains a unique Circuit Identifier, which is an ordered pair consisting of the hash of the matrix M and the row number i of the circuit within M: $CS_i = (\text{Hash}(M), i)$. This identifier will be used in the TorCoin algorithm, together with signatures using the four anonymous temporary public keys comprising that row of M, to prove that TorCoins were minted by circuits assigned in consensus groups.

Security Considerations

Anonymity: The TorPath protocol guarantees that no single relay knows any client's entire circuit. If malicious clients or relays collude, they may be able to shrink the anonymity set to the set of honest relays and clients in the consensus group. Groups can have varying sizes, however, allowing clients to choose a desired balance between anonymity threshold and circuit assignment delay.

Group Formation: The TorPath protocol's random circuit selection mechanism prevents colluding clients and relays from deterministically placing themselves in the same circuit, provided not too many participants in each group collude. Even if half of the temporary keys in matrix M are held by colluding participants, for example, only $1/2^4 = 1/16$ of the assigned circuits will be compromised and able to mint (a limited number of) TorCoins without performing useful work. We could add further protection against "flash mobs" of colluding participants by randomizing group assignment across longer time periods, instead of using temporal locality as the only grouping criterion.

Circuit Diversity: TorCoin's Neff shuffle could assign the same relay to one circuit in multiple positions: e.g., choose the same physical relay as both entry and middle relays. With a reasonable number of participating relays, however, it should be extremely unlikely that one relay gets assigned to *all three* positions on the same circuit. In any case, the risk of accidental relay duplication on one path should not be substantially greater than the risk Tor users already face of randomly placing multiple relays owned by the *same operator* on a circuit. We anticipate that privacy-preserving independence testing techniques [16] could be adapted to detect and reject circuits in which the same relay (or operator) appears multiple times, but we leave this challenge to future work.

Persistent Guards: The assignment process above picks three relays afresh for each circuit, contrary to Tor's practice of keeping entry relays persistent for longer periods. The circuit assignment mechanism could be adapted for persistent entry relays by combining the first two columns of matrix M into one column representing each client together with its choice of entry relay, at the cost of slightly increasing the chance of forming all-compromised circuits.

3.3 TorCoin Mining

In contrast with Bitcoin's reliance on proof of computation, mining TorCoin requires proof of Tor bandwidth transfer. In TorCoin, all participants on a circuit assigned by TorPath may collectively mine a limited number of TorCoins, incrementally, based on the end-to-end goodput they observe on the circuit.

Proof of Bandwidth

1. Each client and relay creates a temporary key R and its hash $R'_* = \text{Hash}(R_*)$.
2. Every m Tor packets, the client sends a tuple $(\text{coin}\#, R'_C)$, where coin# is the number of TorCoin packets previously sent in this circuit.
3. Each relay similarly adds its own temporary hash to the tuple – R'_E, R'_M, and R'_X – and forwards the tuple to the next relay in the circuit.
4. The exit relay forms the coin commitment blob $B = (\text{coin}\#, R'_C, R'_E, R'_M, R'_X)$.
5. The exit relay then signs the blob B with its temporary public key for this circuit to create signature S^B_X, then opens its commitment to reveal R_X, and sends the tuple (B, S^B_X, R_X) to the middle relay.
6. Each prior participant in the circuit i, in turn, similarly signs the blob B to create S^B_i, adds its signature and opened commitment to the tuple, then forwards the tuple to the previous participant in the circuit.
7. The client forms bandwidth proof $P = (B, S^B_X, S^B_M, S^B_E, S^B_C, R_X, R_M, R_E, R_C)$, which anyone may verify against the four temporary public keys in the appropriate row of matrix M for this circuit.

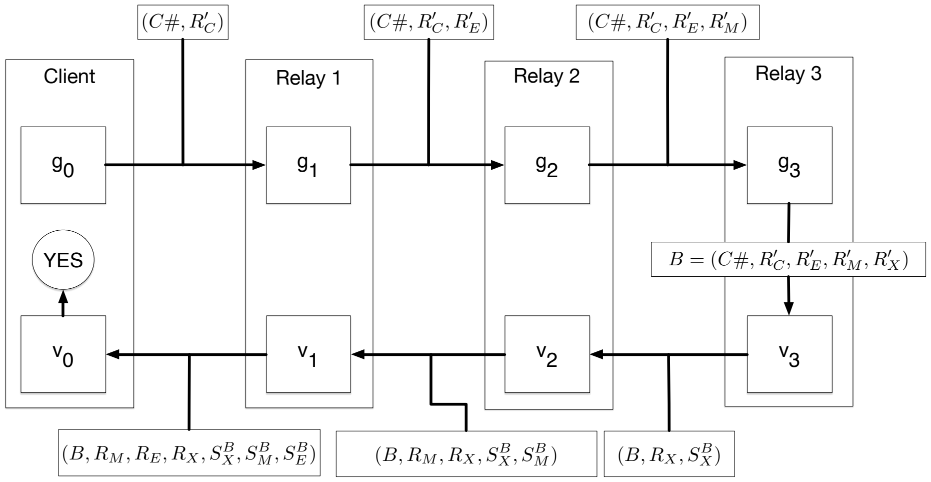
$(C\#, R'_C)$
$(C\#, R'_C, R'_E)$
$(C\#, R'_C, R'_E, R'_M)$
Client
Relay 1
Relay 2
Relay 3
g_0
g_1
g_2
g_3
YES
$B = (C\#, R'_C, R'_E, R'_M, R'_X)$
v_0
v_1
v_2
v_3
$(B, R_M, R_E, R_X, S^B_X, S^B_M, S^B_E)$
$(B, R_M, R_X, S^B_X, S^B_M)$
(B, R_X, S^B_X)

Fig. 4. TorCoin Proof of Bandwidth algorithm. In the upper part of the cycle, the relays add their hashes to the tuple. In the lower part, they add their temporary keys and sign the tuples.

8. To check if a TorCoin has been mined, the client checks if the lower-order bits of Hash$(CS_i, B, R_X, R_M, R_E, R_C) == 0$. If so, the client adds the full proof-of-bandwidth tuple P to the blockchain to validate the new TorCoin. To be valid, the coin# within P must be one greater than that of the last TorCoin in the blockchain mined from circuit i, and must be less than the well-known limit on the number of TorCoins per assigned circuit.
9. Finally, the client uses the standard Bitcoin transfer protocol to pay each relay in the circuit one third of the mined TorCoin.

This protocol leaves all information necessary for verifying proof-of-bandwidth in the blockchain. Any interested party can verify that the circuit identifier in B refers to a valid consensus group by referring to the public log. They can also verify that the blob B was signed by the correct participants by verifying the signatures against the temporary public in the consensus matrix M, and verify that the openings R_i correspond to the corresponding commitments R'_i. Because the low-order-bit test in step 8 depends only on values secretly committed on the "forward path" in steps 2–4, then revealed only on the "return path" in steps 5–6, no proper subset of the circuit's participants can unilaterally recompute B in order to mine TorCoins out of proportion to measured circuit goodput.

Security Considerations

Enforcing packet rate: All honest relays and clients enforce the standard TorCoin packet rate m. Any relays or clients that deviate from this are reported to the assignment servers and the circuit is terminated.

Enforcing circuits: Relays know their neighbours' IP addresses and will refuse connections from any other IP address. Even if malicious relays connect to each other, they will not be able to sign TorCoins unless they own a complete circuit.

Compromised circuits: Colluding clients and attackers needs to control all four components of a circuit to mine TorCoins fraudulently. Even if an adversary controls up to half the network, only $1/2^4 = 1/16$ of assigned circuits will be fully colluding. In practice, we hope and expect that gaining control of even half of all Tor clients and relays would be difficult. To limit the impact of occasional colluding circuits, TorCoin also limits the number of coins each circuit can mine. This coin number is included in the blockchain, so it is easily verified. The impact of compromised circuits can be further reduced by ensuring that consensus groups expire at regular intervals, requiring clients either to form new circuits or cease obtaining new TorCoins from old circuits.

Bitcoin anonymity: This paper focuses on adapting Bitcoin's mining scheme to measure Tor bandwidth instead of computation, and not on Bitcoin's transaction mechanism. Since TorCoin also needs to protect the anonymity of clients as they make transactions, we must also account for well-known concerns about the limited anonymity the basic Bitcoin transaction protocol offers [17]. Any of the recently proposed approaches such as Zerocoin [18], Mixcoin [19], or CoinShuffle [20] should offer a suitable solution to this orthogonal challenge.

Deployment: The TorPath network is not backward compatible with the existing Tor network, due to the fundamental differences of route assignment and access control, which are missing in Tor but necessary for the TorPath and TorCoin schemes to work. However, any given physical server could run both services at the same time. TorCoins are, of course, generated only from TorCoin traffic.

4 Preliminary Results

The TorCoin protocol adds a small amount of overhead to Tor traffic. To evaluate this overhead, we set up a series of servers using the Python Twisted framework [21] to simulate the passing of TorCoin generation and verification messages through a set of relays.

Assuming that the keys, hashes and signatures are all 32 bytes in length, the total overhead from one round of successful TorCoin mining (i.e., one entire round trip from client through all the relays and back again) results in a total TorCoin packet overhead of 1752 bytes. This can be broken down into:

- The first packet from client to entry relay: 34 bytes.
- Packet forwarded from entry to middle relay: 66 bytes.
- Packet forwarded from middle to exit relay: 98 bytes.
- Packet from from exit to middle relay: 324 bytes.
- Packet from from middle to entry relay: 518 bytes.
- Packet from from entry relay to client: 712 bytes
- Total: 1752 bytes

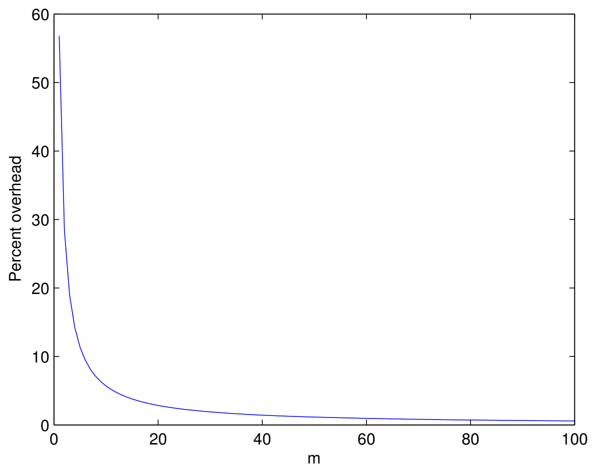

Fig. 5. TorCoin packet overhead

Each round of TorCoin generation and verification happens only after m Tor packets have been sent. Each standard Tor cell is 514 bytes long, so each round trip on the network requires transmission of 514 * 6 = 3084 bytes. Thus, if $m \geq 10$, the TorCoin protocol overhead is around 5%. The value of m can be calibrated in further experimentation and as needed in order to achieve the sweet-spot of transmission efficiency and incentive maximization for relay providers.

The system might decrease the value of m when load is high, incentivizing relay operators to provision more relay bandwidth at such times.

While the Neff shuffle is complex and requires several communications between the servers, we expect the assignment servers will be few (less than 10) and well-provisioned, and do not expect the shuffles to be a major bottleneck. Since this is a one-time cost of connecting to the network, we hope users will accept this setup time if it gives them access to higher-capacity relays. For impatient users, the TorCoin client could use conventional Tor circuits immediately on startup, then transition to TorCoin circuits as they become available.

5 Conclusions

We have introduced TorPath, a novel scheme to assign paths to Tor clients securely and anonymously. TorPath motivated by the need to verifiably mine TorCoins, a a Bitcoin variant based on measured bandwidth over the Tor network. The TorCoin protocol is robust to malicious relays and clients colluding to mint TorCoins without transferring bandwidth.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001-11-C-4018. This work was supported by the Office of Naval Research.

References

1. Androulaki, E., Raykova, M., Srivatsan, S., Stavrou, A., Bellovin, S.M.: PAR: Payment for anonymous routing. In Borisov, N., Goldberg, I., eds.: Privacy Enhancing Technologies: 8th International Symposium, PETS 2008, Leuven, Belgium, Springer-Verlag, LNCS 5134 (July 2008) 219–236
2. Chen, Y., Sion, R., Carbunar, B.: XPay: Practical anonymous payments for Tor routing and other networked services. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2009), ACM (November 2009)
3. Ngan, T.W.J., Dingledine, R., Wallach, D.S.: Building incentives into Tor. In Sion, R., ed.: Proceedings of Financial Cryptography (FC '10). (January 2010)
4. Jansen, R., Hopper, N., Kim, Y.: Recruiting new Tor relays with BRAIDS. In Keromytis, A.D., Shmatikov, V., eds.: Proceedings of the 2010 ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010, ACM (2010)
5. Moore, W.B., Wacek, C., Sherr, M.: Exploring the potential benefits of expanded rate limiting in Tor: Slow and steady wins the race with Tortoise. In: Proceedings of 2011 Annual Computer Security Applications Conference (ACSAC'11), Orlando, FL, USA. (December 2011)
6. Jansen, R., Johnson, A., Syverson, P.: LIRA: Lightweight Incentivized Routing for Anonymity. In: Proceedings of the Network and Distributed System Security Symposium - NDSS'13, Internet Society (February 2013)
7. Johnson, A., Jansen, R., Syverson, P.: Onions for sale: Putting privacy on the market. In: Financial Cryptography and Data Security. Springer (2013) 399–400
8. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Consulted **1** (2008) 2012
9. Jansen, R., Miller, A., Syverson, P., Ford, B.: From onions to shallots: Rewarding Tor relays with TEARS. In: HotPETS. (July 2014)
10. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services (1998)
11. Dovrolis, C., Ramanathan, P.: A case for relative differentiated services and the proportional differentiation model. Network, IEEE **13**(5) (1999) 26–34
12. Snader, R., Borisov, N.: EigenSpeed: secure peer-to-peer bandwidth evaluation. In: IPTPS. (2009) 9
13. Snader, R., Borisov, N.: A tune-up for Tor: Improving security and performance in the Tor network. In: NDSS. Volume 8. (2008) 127
14. Karame, G., Androulaki, E., Capkun, S.: Two Bitcoins at the price of one? double-spending attacks on fast payments in Bitcoin. IACR Cryptology ePrint Archive **2012** (2012) 248
15. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: Proceedings of the 8th ACM conference on Computer and Communications Security, ACM (2001) 116–125
16. Zhai, E., Chen, R., Wolinsky, D.I., Ford, B.: An untold story of redundant clouds: Making your service deployment truly reliable. In: 9th Workshop on Hot Topics in Dependable Systems (HotDep). (November 2013)

17. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of Bitcoins: Characterizing payments among men with no names. In: Internet Measurement Conference. (October 2013)
18. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from Bitcoin. In: IEEE Symposium on Security and Privacy. (May 2013)
19. Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A., Felten, E.W.: Mixcoin: Anonymity for Bitcoin with accountable mixes. In: Financial Cryptography and Data Security. (March 2014)
20. Ruffing, T., Moreno-Sanchez, P., Kate, A.: CoinShuffle: Practical decentralized coin mixing for Bitcoin. In: HotPETS. (July 2014)
21. Twisted Matrix Labs: Twisted Framework. https://twistedmatrix.com/trac/