Rome: Performance and Anonymity using Route Meshes

Krishna P. N. Puttaswamy, Alessandra Sala, Omer Egecioglu, and Ben Y. Zhao
Computer Science Department, University of California at Santa Barbara
{*krishnap, alessandra, omer, ravenben*}@*cs.ucsb.edu*

***Abstract*—Deployed anonymous networks such as Tor focus on delivering messages through end-to-end paths with high anonymity. Selection of routers in the anonymous path construction is either performed randomly, or relies on self-described resource availability at routers, making systems vulnerable to low-resource attacks. In this paper, we investigate an alternative router and path selection mechanism for constructing efficient end-to-end paths with low loss of path anonymity. We propose a novel construct called a "route mesh," and a dynamic programming algorithm that determines optimal-latency paths from many random samples using only a small number of end-to-end measurements. We prove analytically that our path search algorithm finds the optimal path, and requires exponentially lower number of measurements compared to a standard measurement approach. In addition, our analysis shows that route meshes incur only a small loss in anonymity for its users.**

I. Introduction

Privacy of online communications is more important today than ever before. Use of anonymous communication tools such as the Tor network [2] can protect users by preventing third parties from monitoring personal web traffic and associating specific IP addresses with private URLs or webpages. Tor provides anonymity by routing user traffic through a random sequence of encrypted tunnels, each linking two nodes in the public Tor network of more than 1000 nodes. Although popular, the deployed systems provide poor performance even for low-overhead traditional applications like email and web browsing [5], [9], [6]. Paths are built by connecting a set of randomly chosen Tor nodes with highly varying resource capacity and load values. A recent TOR measurement study [5] suggests that even the top quartile of Tor paths have round-trip times around 2 seconds! These round trip times provide unacceptable performance for general web browsing, and completely rule out the use of latency-sensitive applications.

Unlike traditional overlay networks, where paths are easily optimized for end-to-end (E2E) latency, optimizing for low latency paths on Tor poses a significant challenge. Any optimization scheme must preserve anonymity of the E2E path. The key challenge is gathering information about router latencies and capacities without information leakage. One approach is to use a directory service (as in Tor) that advertises node capacities. However, malicious nodes can attract large volume of flows and lower system anonymity by falsely advertising highly desirable qualities, as demonstrated by a recent study using low-resource attacker nodes [1]. A more reliable alternative would be to perform active measurements on E2E paths. However, this requires the source node to contact a large number of first hop nodes, thus increasing its exposure to attackers performing passive timing attacks such as the predecessor attack [10].

The goal of our work is to design a path construction algorithm for anonymous routing networks that provides a user-tunable tradeoff between performance and anonymity that improves upon E2E path measurements. We propose the use of structured anonymous "route meshes," an overlay construction that embeds a large number of random paths. We then describe a dynamic programming algorithm that systematically detects the optimal path for different hop lengths through the mesh, finally selecting an efficient anonymous path. Our dynamic programming algorithm is optimal, and supports simultaneous discovery of multiple node-exclusive backup paths, all while minimizing the exposure of the source node to potential attackers in the network. Our solution, Rome, is general and can be adopted by all path-based anonymous systems, *e.g.* [2]. By performing selective measurements, our approach obtains trustworthy results while minimizing impact on anonymity.

II. Preliminaries

We first introduce the terminology we use in the rest of the paper. All participants in the anonymous system are called *nodes*. A node that initiates an anonymous communication session is called the *source* and the destination of the connection is referred as the *receiver*. A specific communication flow between a source and a receiver routes through several nodes that we call *relay nodes*, and we refer to the combination of the source, receiver and relay nodes as the *path*. The length of time a source remains connected to the same receiver is a *session*. If nodes fail and a path needs to be rebuilt, we refer to the time between each path rebuild process as a *round*.

In this section, we first discuss the performance versus anonymity tradeoff in the context of the Tor anonymous network. We then set the groundwork for our proposed system by defining our assumptions and threat model.

A. Anonymity versus Performance

Relay based anonymous protocols such as Tor [2], and others all share a common tradeoff between anonymity and performance. Practical requirements for performance require that the path construction algorithm take into account the load or performance of heterogeneous nodes in the overlay. The key question is how information about potential routers is gathered and accessed, without exposing the source or making it vulnerable to false advertisements.

Two Approaches for Quantifying Performance. There are two general approaches for gathering performance data about overlay nodes. First, a source node can repeatedly performs end-to-end measurements on p paths. These repeated performance measurements expose the source node to new nodes in the system, increasing vulnerability to passive logging attacks [10], and is akin to building p different paths. The alternative is to ask nodes directly for their performance information. For example, Tor maintains node statistics on uptime and bandwidth in a central directory. While this is scalable and more robust, it makes the system vulnerable to malicious nodes who claim plentiful resources to bias flows to choose them as routers [1]. This increases the probability of success of multiple attackers colluding together to break the anonymity of a flow. Despite systems to verify resource availability, attackers can always obtain resource-rich machines to bias path formation. Such an attack cannot be prevented as long as path selection is biased for resource availability.

Our insights. Two insights ultimately lead us to our proposed system, Rome. First, we believe that performance should be quantified by measurements initiated by the source node. This protects against malicious attackers that misreport resource availability and alternatives that are vulnerable to the Sybil attack [3], including cooperative measurements and reputation systems. Second, biasing path formation for performance must be done in a controlled fashion that improves performance while avoiding dependence on the "optimal" path. Performing limited tuning will enable flows to avoid both performance bottlenecks and resource-based attacks while limiting the source's exposure to passive logging attacks.

B. Assumptions and Threat Model

We consider a passive attacker model in this paper, similar to the model considered in prior work [10]. Attackers can passively log traffic, monitor links and perform passive attacks including timing attacks and the predecessor attack. But they cannot perform active attacks like inverting the encryption, modify message contents, etc. We assume that a fraction of the network is malicious, and hence can monitor the traffic in a fraction of the network. Attackers can collude and share their logs with no delay to enhance the impact of their attack.

In addition, we also make an assumption that each source node using Rome has access to one or more other nodes, called *aliases* (S_1, S_2, S_3 in Figure 1). As we describe later, these aliases help the source perform the initial testdrive measurements anonymously. The source trusts these aliases, and we conservatively assume that compromising the anonymity of an alias compromises the anonymity of the source. These aliases can be additional instances the source user is running on different machines, as in [4]; or they can be trusted friends linked to the source via a social network. As shown in other recent work, trusted friends from social networks can effectively protect nodes from traffic logging attacks [8].

III. Route Meshes and Testdrive

We propose *Rome*, a user-controlled system for optimizing performance of anonymous routes. At the core of our approach is a new construct we call a "route mesh." Instead of connecting an anonymous path between the source and destination through a set of random nodes, Rome selects k times as many random nodes, and randomly places them into a route mesh arranged in the form of a regular matrix, where there are k potential routers to choose from at each hop as shown in Figure 1. We also propose an accompanying end-host driven measurement algorithm called testdrive that uses end-to-end (E2E) light-weight probes to determine the "best" hop path, out of all possible paths through the route mesh. For a L-hop anonymous path, Rome builds a random (k, L) mesh for each flow based on user specified values of k, uses testdrive to determine the best path through the mesh, and uses that path to carry anonymous traffic for the flow. We show in Figure 1 an example of a route mesh for $k = 3, L = 4$.

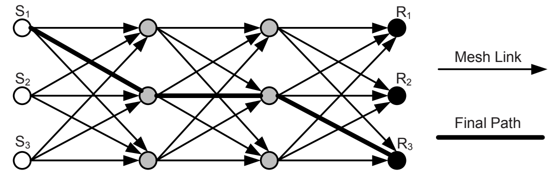

Fig. 1. A simple $k = 3$ route mesh for selecting a 3-hop ($L = 4$) route. The dark line denotes the optimal path found by testdrive.

In each route mesh with k rows and L columns, there are a total of k^L possible $(L-1)$-hop paths. To determine the optimal path out of this sample set, Rome must address two main challenges. First, Rome must measure the performance of different paths *anonymously* without revealing the source or any node's position in the mesh. Second, testing many (k^L) paths is infeasible in practice, and also exposes the source to malicious traffic loggers. Therefore, the source needs to identify the best path with minimal number of measurements ($<< k^L$). We present our solutions to these challenges next.

A. Mesh Construction Details

As shown in Figure 1, Rome organizes kL nodes into k rows and L columns. The source (S_1) and its $k-1$ aliases reside in the first column, and the receiver (R_1) and its $k-1$ aliases reside in the L^{th} column. In the other $L-2$ columns, Rome places $k(L-2)$ randomly chosen nodes. In the rest of the paper, we will use column and level interchangeably.

Building a Symmetric Mesh. Using multiple sources and receivers makes the mesh completely symmetric, such that each router node has k predecessors in the previous level and k successors in the following level. The reason for source and destination aliases is simple. Without them, the first and last columns of the mesh only has one node (instead of k), the source and receiver respectively. Therefore, all nodes in the second level receive messages only from one node (the source), and can easily identify their predecessor as the real source. Similarly, nodes at the $(L-1)^{th}$ column can identify the real receiver as the only node they route probes to. A symmetric mesh prevents these attacks.

Interconnections in the Mesh. Each relay node has k incoming edges and k outgoing edges connecting it to all nodes in the adjoining levels of the route mesh. All edges are unidirectional, and always flow from the source towards the receiver (*left* to *right* in Figure 1). Formally, we adopt matrix notation for naming vertices: a vertex $v_{i,j}$ identifies the j^{th} relay node on the i^{th} row of the mesh. There is an edge between two nodes in the mesh *if and only if* the two nodes are in consecutive levels. We represent an edge in the mesh as: $(v_{i,j},\ v_{m,j+1})$.

Tradeoffs. Since our mesh of kL nodes can form a total of k^L different paths, we can compare the merits of our mesh to an alternative of exploring k^L disjoint random paths. Both cases have a total of k^L paths, but the mesh is constrained to exploring paths on a fixed kL nodes, while disjoint paths can cover a larger portion of the node space. This disadvantage is more than offset by two key benefits of the mesh: lower exposure of the source to first hop routers (k in the mesh, and k^L in disjoint paths), and fewer measurements (k^2L using testdrive, and k^L for disjoint paths).

B. Anonymous Performance Probes in Testdrive

To maintain anonymity during measurement, we cannot use existing measurement techniques such as per-hop latency measurements. We also cannot measure the latency from the source to individual mesh nodes, since that would expose the source to all nodes in the mesh. Finally, to limit initial path setup overhead, we cannot use bandwidth capacity measurement tools such as PathChar to perform per-link measurements.

We use E2E latency measurements to the receiver, in testdrive, to estimate path performance. The source constructs an encrypted message (onion) for each path it wants to *test*, and *drives* it along the path. The message payload contains a request for the receiver to construct and send back an anonymous reply in the reverse direction to help measure the latency. The malicious relays on a path cannot distinguish these probe messages from regular messages, and therefore cannot manipulate routing to shorten the E2E latency.

This simple measurement mechanism accomplishes the goal of measuring the cumulative round-trip-time (RTT) of a path. However, given a path, it cannot identify (thus avoid) its latency bottleneck, the node that is most heavily-loaded (and therefore contributing the most delay). Unfortunately, traditional techniques that localize performance bottlenecks either reveal too much information or incur very heavy measurement overheads. We show below a recursive measurement technique that finds the optimal path using dynamic programming.

C. Minimizing Testdrive Probes

Our goal is to locate the minimum latency path out of k^L paths in the mesh using minimal testdrive probes. Using only E2E measurements to the receiver, our algorithm isolates and determines optimal subpaths between the source and receiver. The idea is to incrementally determine the best path to each node in the mesh by comparing latencies across alternate paths to that node that share common subpaths.

Algorithm 1 Finds Optimal End-to-End Path in Mesh

Path=Source.Optimal_Path(Mesh mesh)

Random_Test();
$< P_1, P_2, ..., P_k >= Best\_Path(L)$;
for $i = 1$ to $i \leq k$ **do**
  $m_i = Measure(P_i)$;
end for
b = index i s.t. $m_i = \min\{m_1, m_2, ..., m_k\}$;
Return P_b

Algorithm 2 Produces Random Sequence of Dummy Probes

Random_Test()

x= Random_Number();
for $i = 1$ to $i \leq k$ **do**
  P_i =Horizontal_Path$(i, 1, L)$;
end for
for $i = 1$ to $i \leq x$ **do**
  for $j = 1$ to $i \leq k$ **do**
    $Measure(P_j)$
  end for
end for

The algorithm begins as follows. For each first hop relay, we compare all E2E paths that differ only in their first hop. Because they share all links except the first hop, comparing their E2E latencies reveals the shortest first hop link to this relay. We use this to build a dictionary of shortest paths to all relays in the second column. Then for each relay r in column 3, we construct k E2E paths by extending the k shortest paths for column 2 to r in column 3, and add a common suffix path from r to a receiver. Comparing E2E latencies of these paths reveals the shortest paths to r. This process recurses for all relays in column 3, and across columns. Thus after step i, we have computed the shortest paths from the source to all k nodes in column i. Since the shortest path to any node on the $(i+1)^{th}$ column must contain a shortest path to column i, we only need to compare the relative latencies of k possible candidates for each node. We provide formal proof for optimality of this algorithm in Section IV.

We show an example of testdrive in action in Figure 2. Here, testdrive is computing the optimal paths to node $v_{1,4}$ (the 4^{th} relay node on the 1^{st} row), having already computed shortest paths for each of the k nodes in the previous column (chosen paths marked in thick arrows). Computing the shortest path to $v_{1,4}$ comes down to comparing E2E latencies of k possible paths generated from the concatenation of an optimal path to a predecessor p of $v_{1,4}$, the link between p and $v_{1,4}$, and a common suffix path from $v_{1,4}$ to a receiver (R).

The source builds a mesh and calls *Optimal_Path*, described in Algorithm 1, which locates the optimal path in the mesh. This in turn calls *Best_Path*, Algorithm 4, to compute an optimal path to each of the k receivers in the last column. Algorithm 1 also calls *Random_Test*, that is necessary to avoid attacks described later in § IV.

Algorithm 3 generates fixed suffix paths, *e.g.* $(v_{1,4}, R)$ in Figure 2, which are concatenated to a pre-computed optimal path and a link being evaluated. The result is k E2E paths

Algorithm 3 Generates Common Suffix Subpaths

Path=Horizontal_Path(row i, first_column j, last_column L)

Path=$\not\vdash$; */* Start with an empty path */*
for $m = j$ to $m \leq L$ **do**
  Path=Path$\circ(v_{i,m}, v_{i,m+1})$;
end for
Return Path;

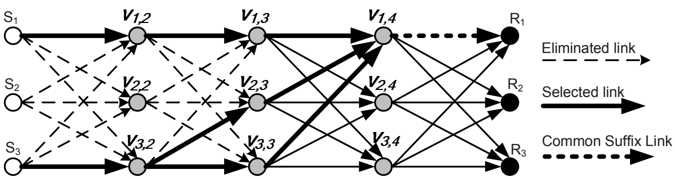

Fig. 2. A snapshot of testdrive in action. Optimal paths have been computed for nodes through column 3. Computing the optimal path to $v_{1,4}$ means comparing E2E latencies for 3 paths, each containing an optimal subpath and sharing the same suffix from $v_{1,4}$ to R.

Algorithm 4 Recursive Search Function for Optimal Paths

$< P_1, P_2, ..., P_k >$=Best_Path(level g)

if g==1 **then**
  for $i = 1$ to $i \leq k$ **do**
    $M[i, j] =\not\vdash$;
    $P_i =\not\vdash$
  end for
  Return $< P_1, P_2, ..., P_k >$;
end if
$< P_1, P_2, ..., P_k >=$ Best_Path($g - 1$);
for $i = 1$ to $i \leq k$ **do**
  for $j = 1$ to $j \leq k$ **do**
    m_j= Measure($P_j \circ (v_{j,g-1}, v_{i,g}) \circ$ Horizontal_Path(i, g, L));
  end for
  b = index i s.t. $m_j = min_j\{m_1, m_2, ..., m_k\}$;
  $M[i, g] = P_b \circ (v_{b,g-1}, v_{i,g})$;
end for
for $i = 1$ to $i \leq k$ **do**
  $P_i = M[i, g]$
end for
Return $< P_1, P_2, ..., P_k >$

that, when compared, reveal the shortest path to the node in question. Algorithm 4 implements the recursive function to compute the best path starting from all sources. This algorithm computes the best path involving all nodes in each level using information computed from the previous recursive call. When it terminates, it returns an optimal path for each of the k receivers. *Measure_Path* returns the round-trip-latency of a given path, which is a cumulative measure of both link delays and processing delays at intermediate routers.

IV. Analytical Results

First, we present optimality analysis of testdrive, and quantify the path construction cost. We then bound the loss in anonymity in Rome compared to Tor-like relay paths.

A. Optimality of the Testdrive Output

Testdrive constructs the optimal paths from the source-aliases to each node on the mesh incrementally level by level. The paths are constructed at a level using the information computed in the previous level. This approach allows us to formalize our problem as a dynamic programming (DP) algorithm. The optimal path, for each node i at level L, is constructed using the following recursive formulation:

$$P_i(L) = \begin{cases} 0, & \text{if L=1 and } 1 \leq i \leq k; \\ min \begin{cases} P_1(L-1) \circ (v_{1,L-1}, v_{i,L}), \\ P_2(L-1) \circ (v_{2,L-1}, v_{i,L}), \\ ..., \\ P_k(L-1) \circ (v_{k,L-1}, v_{i,L}) \end{cases}, & \text{otherwise.} \end{cases} \tag{1}$$

Using this formulation, we argue that the DP algorithm *Best_Path* is optimal, has the optimal substructure and overlapping subproblems properties. We omit the proofs of each of these properties for space constraints. Please refer the Technical Report for detailed proofs [7].

Proof of Optimality. To prove optimality, we need to show that in a mesh with kL total nodes, testdrive produces the path with minimum delay among k^L possible paths. To simplify notation, v_i indicates one of the nodes in the i^{th} level. Note that a path from the source to the receiver must go through exactly one node in each level.

Theorem 1. *Let $P =< v_1, v_2, v_3, .., v_L >$ be a resultant path between v_1 and v_L constructed by our DP formulation, then P is optimal.*

Theorem 2. *Let $P =< v_1, v_2, v_3, .., v_L >$ be the path between v_1 and v_L returned from our DP algorithm, then each prefix of P is an optimal path $\forall v_i$ with $2 \leq i \leq L - 1$.*

Quantifying the Costs of Testdrive. We quantify the measurement overhead of path construction in testdrive next.

Theorem 3. *The number of messages sent during the testdrive path construction phase is $O(k^2L^2)$.*

We believe this overhead is quite reasonable, because k and L are small in real systems ($L = 3$ in Tor [2]).

B. Anonymity of the Mesh

Anonymous paths use relay to protect endpoint identities from attackers. Previous work has shown that rebuilding paths between the same end-points makes the flow increasingly vulnerable to passive anonymity attacks [10].

Attacks to Identify the Position of Relay Nodes. Attackers are interested in knowing their position in the path. This knowledge enables attackers to launch attacks such as timing attacks, and simplify the execution of predecessor attacks. Hence, we need to guarantee that malicious nodes cannot recognize their position on the mesh by counting messages.

During testdrive phase, the source sends measurement packets along different paths. However, the attackers can exploit an asymmetry in the measurement phase to identify the source. During measurements, a node in the second level (the level after the source), sees packets from all its incoming links in one phase, then in the next phase forwards traffic to all of its outgoing links. However, the nodes in the other level see packets only from the horizontal link first, and see packets from other links after a few measurements. To avoid this asymmetry, we introduced *Random_Test* in testdrive, as described in Algorithm 2 that sends m dummy packets before the real measurements.

Lemma 1. *After the testdrive phase, a malicious relay node in the second column can infer that it is in the second position in the mesh only with probability:* $\frac{1}{\lfloor\frac{m}{k}\rfloor+1}$.

Thus, the more the random messages sent, the lower the probability with which a malicious node can guess its position. It is possible that two or more attackers are in the mesh, collude by counting the number of messages they receive and derive their relative position in the mesh based on this count. This helps the attackers perform predecessor attack faster. However, this attack is valid only when the path length is fixed. The source node can easily build meshes of different lengths and avoid this attack.

Anonymity Lost Under the Predecessor Attack. Now we quantify the anonymity of the mesh when c colluding attackers (out of N total nodes) perform the predecessor attack [10]. We compare the anonymity of the mesh both with Tor-like paths, and the case where a source explores k^L disjoint paths (the same number of paths supported by Rome). The attackers on the mesh immediately after the source's aliases and immediately before the receiver's aliases can perform the predecessor attack. We analyze the probability with which c colluding attackers can compromise a mesh-based path next.

Lemma 2. *In each round, c colluding attackers can log the right source and receiver with probability:* $1-(1-(\frac{c}{N})^2)^{k+1}$.

Please refer [7] for the proof. Using this probability that the attackers can log the right source and the receiver in a given round, we can calculate the number of rounds the attackers need to guess the right source and receiver.

Theorem 4. *The end-points of the mesh-based anonymous communication path can be discovered by c colluding malicious nodes, performing the predecessor attack, in* $O(\frac{N^{2k+2}}{N^{2k+2}-(N^2-c^2)^{k+1}}\log N)$ *rounds, with high probability.*

As discussed in Section III-A, our mesh provides k^L different paths, but only uses kL different nodes. This limited number of nodes limits the search space for possible paths, but helps maintain anonymity against passive attackers. The larger the number of paths used, the lower the anonymity.

Theorem 5. *The end-points of an anonymous communication system using* k^L *disjoint paths between them in each round, can be discovered by c colluding malicious nodes performing the predecessor attack in* $O(\frac{N^{2k^L}}{N^{2k^L}-(N^2-c^2)^{k^L}}\log N)$ *rounds.*

This proof follows the same scheme as Theorem 4. We summarize the key results in Table I, and the compare the vulnerability of the three different approaches, we consider in this paper, to the predecessor attack.

TABLE I
ASYMPTOTICAL BOUNDS TO ATTACK DIFFERENT ANONYMOUS SYSTEMS.

Strategies	Rounds to attack W.H.P
Tor-like paths	$O((\frac{N}{c})^2\log N)$
Mesh-based paths	$O(\frac{N^{2k+2}}{N^{2k+2}-(N^2-c^2)^{k+1}}\log N)$
k^L disjoint paths	$O(\frac{N^{2k^L}}{N^{2k^L}-(N^2-c^2)^{k^L}}\log N)$

V. IMPLEMENTATION AND EVALUATION

For comparison, we implemented Rome and a simplified version of the Onion Router in Python, and deployed both on the PlanetLab internet testbed. We choose a number of random pairs of communication endpoints. Each source node selects a "best path" according to Rome, Onion Routing and an Oracle path formation scheme. For space reasons, we refer the reader to our tech report [7] for detailed graph results.

As expected, we observed in our measurements that the latencies of the Onion routing for larger lengths degrade significantly. For example, in our measurements, for $L = 3$ nearly 70% of the paths stay under a latency of 1 second, while the latency of only around 40% of the paths stay under 1 second when the path $L = 6$. When we compare latency of mesh-based paths against Onion routing latencies ($L = 3$), nearly 90% of the mesh paths stay under 1 second delay, while only around 70% of the onion paths stay under 1 second for $L = 3$. Mesh performance improves with larger k values.

Finally, when we compare mesh paths against the best latencies from k^L disjoint paths, we find that the performance of a single mesh is very close to that of the disjoint paths for identical k and L values. This shows why route meshes can provide exceptional path latency without generating the k^L traffic and associated loss in anonymity.

These results show that Rome, our user-managed approach to tuning tradeoffs between performance and anonymity, provides efficient searching for optimal paths using a limited number of end-to-end performance measurements with low cost in anonymity.

REFERENCES

[1] BAUER, K., ET AL. Low-resource routing attacks against tor. In *Proc. of Workshop on Privacy in Electronic Society* (Alexandria, VA, 2007).

[2] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *USENIX Security* (2004).

[3] DOUCEUR, J. R. The Sybil attack. In *Proc. of IPTPS* (March 2002).

[4] KATTI, S., COHEN, J., AND KATABI, D. Information slicing: Anonymity using unreliable overlays. In *nsdi* (2007).

[5] MCCOY, D., ET AL. Shining light in dark places: A study of anonymous network usage. Tech. Rep. CU-CS-1032-07, Univ. of CO, 2007.

[6] PRIES, R., ET AL. On performance bottleneck of anonymous communication networks. In *Proc. of IPDPS* (Miami, FL, 2008).

[7] PUTTASWAMY, K. P. N., SALA, A., EGECIOGLU, O., AND ZHAO, B. Y. Rome: Performance and anonymity using route meshes. Tech. Rep. 2008-11, UCSB, 2008. http://www.cs.ucsb.edu/research/tech_reports.

[8] PUTTASWAMY, K. P. N., SALA, A., AND ZHAO, B. Y. Improving anonymity using social links. In *Proc. of NPSec* (October 2008).

[9] SNADER, R., AND BORISOV, N. A tune-up for tor: Improving security and performance in the tor network. In *ndss* (San Diego, CA, 2008).

[10] WRIGHT, M. K., ET AL. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM TISS 7*, 4 (2004).