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 from each router, which makes the system 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. Meanwhile, experimental deployment of our anonymous routers on Planet-lab shows dramatic improvements in path selection quality using very small route meshes that incur low measurement overheads.**

I. Introduction

Privacy of online communications is more important today than ever before. With different aspects of our lives being digitized and moving online, each of us is accumulating a large volume of personal information in the form of online records and logs. Sufficiently motivated, a malicious entity can use social networks, blogs and online logs to obtain information about our shopping and reading habits, travel plans, personal opinions and friends and family. As shown in the recent Viacom vs. Youtube ruling [6], online privacy for the average Internet user may be sacrificed to protect content owners against the misbehavior of the few. Similar shifts in U.S. policies may also signal the advent of Internet wiretapping as a common intelligence tool [16].

Use of anonymous communication tools such as the Tor network [3] 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 [10], [17], [13]. 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 [10] 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 such as Voice-over-IP.

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 used 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. Recent studies have demonstrated the effectiveness of this attack on a large fraction of the users in the network even with 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 malicious nodes performing passive timing attacks such as the predecessor attack [22], [21].

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 proven to be optimal, and supports the 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.* [5], [3]. By performing selective measurements, our approach achieves accurate and trustworthy results while minimizing impact on anonymity.

This paper makes three key contributions. First, we describe in Section III a general route mesh design for anonymous path construction, and a *testdrive* algorithm for scalable route selection. Second, we use detailed analysis in Section IV to prove the optimality of our algorithm, and bound the tradeoff between anonymity and number of random paths searched. Third, we present extensive simulation and measurement results that quantify the performance improvement of Rome over existing path construction approaches.

II. Preliminaries

Anonymous routing networks such as Tor construct E2E anonymous tunnels from randomly chosen nodes in the overlay. While this random selection supports strong anonymity [3], [12], it ignores heterogeneity of node capacities in the overlay, easily overloading low-resource nodes and creating performance bottlenecks. Our goal is to improve performance by allowing users to tradeoff performance and anonymity by performing informed path selection following a small number of E2E measurements.

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

Chaum-mix based anonymous protocols include Tor [3], Salsa [12], Tarzan [5] and others [24], [14], all of which 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 node or making it vulnerable to false advertisements.

Two Approaches for Quantifying Performance. There are two general approaches for gathering performance data about overlay nodes, performing active measurements to estimate node capacity, or asking the nodes for information.

One can imagine a system where a source node repeatedly performs E2E measurements on p potential paths to select one that provides the desired performance properties. This is akin to building paths and tearing them down repeatedly p times. These repeated performance measurements expose the source node to p new nodes in the system, increasing its vulnerability to passive logging attacks such as the Predecessor attack [21], [22]. In addition, it is likely that p remains a small value, and the source can only sample the performance of a small number of potential paths.

The alternative is to ask nodes directly for their performance information, and is the simple approach adopted by the Tor anonymous network. Tor uses a central directory to maintain node statistics on uptime and bandwidth. While this is scalable and preserves anonymity, it relies on truthful information from individual nodes. Malicious nodes can claim high resources in order to bias flows to choose them as routers [1]. This increases the probability of multiple attackers colluding together to break the anonymity of a single flow. Researchers have proposed verification techniques such as bandwidth measurements and distributed reputation systems. However, attackers can actually obtain high-powered machines and truthfully advertise high resources to bias path formation. Such an attack cannot be prevented as long as the path formation algorithm included bias for resource availability.

Our insights. From studying the two approaches discussed above, we arrive at two independent insights that ultimately lead us to our proposed system, Rome. First, we believe that performance should be quantified by measurements initiated by the source node. Clearly, malicious attackers can report arbitrarily high performance values. In addition, indirect alternatives based on distributed reputations or collaborative measurements are all vulnerable to the Sybil attack [4], where a single attacker can instantiate multiple online identities and use them to manipulate collaborative measurement or reputation systems.

Second, existing attacks have shown that biasing path formation for performance leads to increased vulnerability to resource-based attacks [1]. Therefore, tuning 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. Finally, in the case of measurement-based approaches, limiting performance tuning will also protect the source node from passive logging attacks.

B. Assumptions and Threat Model

We consider a passive (non-active) attacker model in this paper, similar to the model considered in prior work [21], [22], [14]. 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. Formally, c nodes are malicious out of N total nodes in 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 (S) using Rome has access to one or more other nodes, called *aliases* ($S_1, S_2 ... S_k$ in Figure 1). As we describe later, these aliases help the source S 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 S. These aliases can be additional instances the source user is running on different machines, as in [7]; 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 [15].

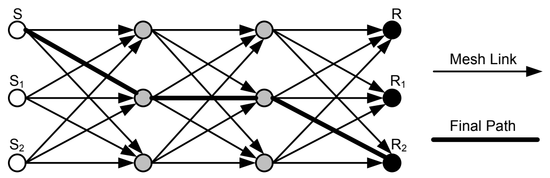

Fig. 1. A simple $k = 3$ route mesh for selecting a 3-hop ($L = 4$) route. The dark line denotes the optimal path found by testdrive.

III. Route Meshes and Testdrive

Based on our observations in Section II, 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. 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 $k = 3, L = 4$ (we explain the symmetric design of the mesh below).

Rome's route meshes allow users to customize their level of anonymity-performance tradeoff. While testdrive is proven to determine the optimal path in the mesh in polynomial time, users control the size of the mesh, and consequently the number of random paths sampled in the path selection process. Increasing the value of parameter k adds more random paths to the search space, increasing the likelihood of finding a better path, and along with it an increased vulnerability to a passive logging by resource-rich attacker nodes.

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 in detail in the remainder of this section.

A. Mesh Construction Details

As shown in Figure 1, Rome organizes kL nodes into k rows and L columns. The source and its $k-1$ aliases reside in the first column, and the receiver 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). We use analysis and measurements to quantify these tradeoffs in Sections IV and V.

B. Anonymous Performance Probes in Testdrive

Rome faces the unique challenge of maintaining anonymity of the source while performing path measurements. As a result, we cannot use existing performance 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.

Testdrive uses E2E latency measurements to the receiver 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. Note that malicious relays on a path cannot distinguish these probe messages from regular messages, and therefore cannot manipulate routing to shorten the E2E latency. In addition, a source node can add extra padding and dummy data to vary packet sizes and avoid recognition by intermediate relay nodes. Since packet padding and anonymous route reply techniques are well-studied in literature [24], [3], we do not describe them here.

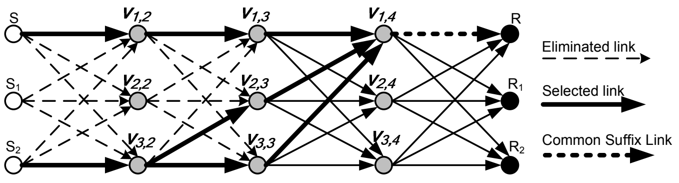

Fig. 2. A snapshot of Testdrive in action. Optimal paths have been computed for nodes through column 3. Computing the optimal path to $v_{1,4}$ means comparing E2E latencies for 3 paths, each containing an optimal subpath and sharing the same suffix from $v_{1,4}$ to R.

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

```
Path=Source.Optimal_Path(Mesh mesh)
1: Random_Test();
2: $< P_1, P_2, ..., P_k >= Best\_Path(L)$;
3: for $i = 1$ to $i \leq k$ do
4:     $m_i = Measure(P_i)$;
5: end for
6: b = index i s.t. $m_i = min_i\{m_1, m_2, ..., m_k\}$;
7: Return $P_b$
```

Algorithm 2 Produces Random Sequence of Dummy Probes

```
Random_Test()
1: x= Random_Number();
2: for $i = 1$ to $i \leq k$ do
3:     $P_i$ =Horizontal_Path(i,1,L);
4: end for
5: for $i = 1$ to $i \leq x$ do
6:     for $j = 1$ to $i \leq k$ do
7:         $Measure(P_j)$
8:     end for
9: end for
```

Algorithm 3 Generates Common Suffix Subpaths for Concatenation

```
Path= Horizontal_Path(row i, first_column j, last_column L)
1: Path=$\not\vdash$;
2: for $m = j$ to $m \leq L$ do
3:     Path=Path$\circ(v_{i,m}, v_{i,m+1})$;
4: end for
5: Return Path;
```

This simple measurement mechanism accomplishes the goal of measuring the cumulative round-trip-time (RTT) of a path. For a given path, however, it cannot identify (and 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. Luckily, this is not essential to our goal. 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 the minimal number of testdrive probes. Using only E2E measurements to the receiver, we propose a novel algorithm that incrementally isolates and determines optimal subpaths between the source and receiver. The idea is to incrementally determine the shortest path to each node in the mesh by comparing latencies across alternate paths to that node that share common subpaths.

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).

We next describe our algorithm in detail with pseudo-code. 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*(Algorithm 2), which introduces a random number of dummy probe messages. These messages prevent nodes from determining their location in the mesh by monitoring message flows. Without them, nodes in each level can monitor messages and could distinguish "waves" of measurement traffic, and use the index of the wave to determine which column it resides in the mesh. Introducing random number of dummy messages artificially inflates any such estimate. We analyze this mechanism in Section IV.

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

Our algorithm assumes that the link latency and node processing delays are stable during our mesh measurement

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 $< P_1, P_2, ..., P_k >$
```

phase. This assumption is not really restrictive, since our algorithm runs in time polynomial to the mesh size (shown in Section IV), and mesh sizes are quite small.

IV. Analytical Results

First, we will present formal analysis of the optimality of testdrive. We then bound the loss in anonymity using Rome compared to Tor-like relay paths. Finally, we quantify our performance improvement over single relay paths.

A. Optimality of the Testdrive Output

We will first prove the optimality of paths produced by testdrive, then quantify the cost of our approach in terms of total messages generated to test paths in the mesh. The algorithm 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 recursive structure allows us to formalize our problem as a dynamic programming algorithm. The optimal path, for each node i at level L, is constructed using the following recursive formulation:

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

Using this recursive relation, we later prove that our dynamic programming algorithm *Best_Path* has the optimal substructure and overlapping subproblems properties. These properties are necessary to prove the optimality of the path resulting from the *Best_Path* algorithm.

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 from our dynamic programming algorithm, then P is optimal.*

Proof: A path P is the optimal path between v_1 and v_L if the cumulating delay incurred in going through nodes $v_2, v_3, ..., v_{L-1}$ is the minimum compared to all the possible paths in the mesh (i.e. k^L in total). Suppose there exists a path $B =< u_1, u_2, ..., u_L >\neq P$ such that $Delay(B) < Delay(P)$, then there are three cases to analyze.

First, P and B are completely disjoint, which means $\forall i$ $v_i \neq u_i$. Note that we compare only pairs at the same level because, by construction, a path is defined as the concatenation of exactly one relay from each level in increasing order, i.e. for $i = 1$ to $i = L$. Because of the fact that these two paths are completely disjoint they will be compared before the function *Source.Optimal_Path()* returns. This means that P won the test of the minimal path on the last-but-one line in the *Source.Optimal_Path()* function. Therefore, the hypothesis that $Delay(B) < Delay(P)$ results an absurd.

Second, it is similar to the previous case, when P and B share an prefix on their paths, which means that on the prefix they have the same delay. Also in this case, the paths P and B will be compared on the last-but-one line of the *Source.Optimal_Path()* function with the same previous result, which means we find the same contradiction about the *Delay* of P with respect to B.

Third, P and B share a common suffix. Again, suppose that P is the optimal path returned from our algorithm, but there exists B such that $Delay(B) < Delay(P)$. Formally, when two paths share a common suffix, it means that $\exists\ 2 \leq i \leq L-1$ such that $v_j = u_j\ \forall j$ from i to L. Obviously, the *Delay* on this suffix is exactly the same, and so $Delay(prefix\ of\ B)$ has to be less than $Delay(prefix\ of\ P)$ in order for $Delay(B) < Delay(P)$. Since the recursive *Best_Path(i)* function (for the level i) compares the incoming paths, on $v_i = u_i$ the test between $Delay(prefix\ of\ P)$ and $Delay(prefix\ of\ B)$ is won from $Delay(prefix\ of\ P)$ because it is the optimal and so $Delay(prefix\ of\ B) < Delay(prefix\ of\ P)$ is impossible. This concludes the proof. ∎

Optimal Substructure. Here we show that the path P generated by our dynamic programming algorithm has the optimal substructure property.

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

Proof: Let $P =< v_1, v_2, v_3, .., v_L >$ be the optimal path. Assume that $\exists\ 2 \leq i \leq L-1$ and $\exists$ a path $A =< u_1, u_2, ..., u_i >$ such that the *Delay* of A is the minimum for the node u_i and $v_i = u_i$. In this setting, A is the optimal path for the node u_i by assumption, and because $u_i = v_i$ we can construct a new path $B =< u_1, u_2, ..., u_i, v_{i+1}, .., v_L >$ which has $Delay(B) = Delay(A) + Delay(suffix\ of\ P)$ that is less or equal to $Delay(P)$ because by assumption $Delay(A)$ is the minimum for the node u_i. Because P is the optimal path, the assumption about the optimality of A is absurd – which means that $Delay(A) \geq Delay(< v_1, v_2, v_3, .., v_i >)$. This result confirms the existence of the optimal substructure property. ■

Overlapping Subproblems. An indispensable characteristic of an optimization problem solved using dynamic programming that the optimal solutions to subproblems have to be reused over and over in order to generate the optimal solution for bigger problems.

The optimal path is constructed level by level and so when the algorithm computes the optimal paths for the nodes at level i it reuses the optimal paths from each node at level $i-1$ that have been computed using the optimal paths from each node at level $i-2$ and so on. Formally, $\forall\ level\ i$ and for each ones of the k nodes in the level i, the optimal paths are computed using all the k optimal paths at level $i-1$.

Because our algorithm reuses the previously computed solutions for subproblems to find the solution to bigger problems, we are able to show the overlapping subproblem property. In addition, by using the recursive subproblem solution, we will show later (see Theorem 3) that the final cost of this algorithm is polynomial in the input size (of the mesh).

Quantifying the Costs of Testdrive. In order to understand the measurement overhead due to testdrive mechanism, we quantify the amount of traffic introduced, in terms of the number of messages sent, during the testdrive phase to find the optimal path.

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

Proof: The algorithm tests each level once. To test a level, the algorithm has to test k^2 edges. The paths going through these edges produce a total of $k^2(L-1)$ messages. This must be repeated for each level, producing a total number of messages equal to $(L-1)k^2(L-1)$, which is asymptotically $O(k^2L^2)$. ■

In real systems, k and L are very small ($L = 3$ in Tor [3]). Considering that this is a one-time up-front cost that improves the performance for an entire round, this overhead seems quite reasonable. Next, we will quantify the loss in anonymity from mesh-based probing.

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 [21], [22].

Attacks to Identify the Position of Relay Nodes. Attackers are interesting 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. Careful analysis of *Best_Path* shows that there is an asymmetry in the measurement phase that attackers can exploit 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. This procedure sends an initial random number r of dummy packets from the source along each horizontal path before starting real measurements. We have the following:

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

Proof: The testdrive mechanism tests the levels one by one, and in each test level k^2 messages are involved. A malicious node in the second level receives r random messages (indistinguishable from the test-path messages) and then it is involved in the test phase. Because of these initial messages, a node in the second level can only guess that it can be in any position between the second and$(\lfloor \frac{r}{k} \rfloor + 1)^{th}$ level in the mesh. As a result, the probability with which it can guess to be a member of the second level is only: $\frac{1}{\lfloor \frac{r}{k} \rfloor + 1}$. ■

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 Predecessor Attack. Next, we quantify the anonymity of the mesh when c colluding attackers perform the predecessor attack. 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).

In Tor-like paths, the probability to compromise just the two nodes directly in contact with the source and the receiver, the source's successor and the receiver's predecessor, has been shown to be $(\frac{c}{N})^2$ [21], [22]. To analyze the amount of anonymity that c colluding attackers can gain in an mesh-based path, we need to figure out which are the positions that the attackers should compromise in order to log the right information about the endpoints. We do this in the next lemma.

Lemma 2. *In each round, c colluding malicious nodes can compromise position in which they are able to log the right source and receiver with probability:* $1 - (1 - (\frac{c}{N})^2)^{k+1}$.

Proof: Each node in the column immediately after the sources see all the sources. In a standard anonymous system, in order to perform a successful predecessor attack the attackers have to control the first position after the source and the last before the receiver. In these systems in each round there is only one path between source and receiver, and so the probability that the attackers are in the right positions is $(\frac{c}{N})^2$, as proven in [21], [22]. The mesh involves more nodes and gives more opportunity to the attackers to log the source and the receiver. The mesh structure allows malicious nodes to attack the mesh using $k+1$ different configurations – k cases when the successor of a source in i^{th} row colludes with a predecessor of a receiver in the same i^{th} row, and one case when the successor of the source and the predecessor of the receiver on the optimal path collude. Formally, to perform a successful predecessor attack, the malicious nodes must compromise at least one of the following pairs of relay nodes: $\forall\ 1 <= i <= k\ v_{i,2}\ and\ v_{i,L-1}$ or the successor of the source and the predecessor of the receiver on the optimal path. The total probability of this happening can be expressed as: $\sum_{i=1}^{k+1} \binom{k+1}{i} ((\frac{c}{N})^2)^i (1-(\frac{c}{N})^2)^{k+1-i} = 1 - \binom{k+1}{0}((\frac{c}{N})^2)^0(1-(\frac{c}{N})^2)^{k+1} = 1-(1-(\frac{c}{N})^2)^{k+1}$. ■

Now that we know the probability that the attackers can log the right source and the receiver in a given round, we need to next prove the number of rounds the attackers have to log information in order to guess the right source and receiver with high probability.

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 $O(\frac{N^{2k+2}}{N^{2k+2}-(N^2-c^2)^{k+1}} \log_e N)$ rounds, with high probability.*

Proof: The predecessor attack logs information in each round. The probability that the attackers are in the right position has been presented in the Lemma 2. Let A be the event that the malicious nodes control the relay nodes that logs the right source-receiver information.

Let $X_1, X_2, ..., X_T$ be T random variables such that:
$X_i = \begin{cases} 1, & \text{if the event A is true during the i-th round} \\ 0, & \text{otherwise.} \end{cases}$
Let p_i be the probability that $X_i = 1$, in our case $p_i = P[A\ is\ true\ during\ i-th\ rounf]$ and let $\mu = E[X] = \sum_{i=0}^{T-1} p_i$. By Chernoff bound [11] we have $P(X < (1-\delta)\mu) < e^{\frac{-\mu(\delta)^2}{2}}$. In particular, $p_i = 1 - (\frac{N^2-c^2}{N^2})^{k+1}$ and $\delta = 1/2$ we have:

$$\mu = \sum_{i=0}^{T-1} 1 - (\frac{N^2-c^2}{N^2})^{k+1} = (1-(\frac{N^2-c^2}{N^2})^{k+1})T$$

and so, $P(X < (1-\delta)\mu) = P\left(X < 1/2(1-(\frac{N^2-c^2}{N^2})^{k+1})T\right) < e^{-1/8((1-(\frac{N^2-c^2}{N^2})^{k+1})T)}$. This probability is $< \frac{1}{N}$ iff: $T > 8(\frac{N^{2k+2}}{N^{2k+2}-(N^2-c^2)^{k+1}}) \log_e N$. We can see that with probability $\frac{N-1}{N}$ the number of rounds used from the attackers is $T = O\left(\frac{N^{2k+2}}{N^{2k+2}-(N^2-c^2)^{k+1}} \log_e N\right)$. ■

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, as we now show.

TABLE I
ASYMPTOTICAL BOUNDS TO ATTACK DIFFERENT ANONYMOUS SYSTEMS.

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

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 $O(\frac{N^{2k^L}}{N^{2k^L}-(N^2-c^2)^{k^L}} \log_e N)$ rounds.*

This proof follows the same scheme as Theorem 4, and so we omit it. We summarize in the Table I results comparing the vulnerability of three different approaches to the predecessor attack: the standard Tor-like single path, Rome, and the k^L disjoint paths.

C. Quantifying the Probability of Finding a Fast Path

We wish now to analytically quantify the performance impact gained by using route meshes in Rome. Specifically, we will model a heterogeneous network and compare analytically the probabilistic performance of Rome paths against both randomly chosen Tor-like paths and selecting an optimal path from k^L disjoint random paths. We assign to each node in the network a unique number that represents the delay that a node brings into a path when it is added to an anonymous path. This delay is a cumulative measure that represents multiple factors such as CPU processing delay for crypto operations and local network I/O processing delay. This delay measure is continuous, and we can use it to order all network nodes by their "quality." We also assume that the network contains a number of links, each characterized by a link latency. For simplicity, we assume each node is associated with a link that represents its access network.

We model a path as a combination of nodes and their links. Because of the random selection of nodes, every couple of nodes can be chosen as consecutive in the anonymous path and so a link between each possible couple of nodes can be selected. Finally, we can see a path as a chain of elements "node-link-node". In this setting there are $N(N-1)$[1] "node-link-node" segments in the network that can be used to form the anonymous path. Each of these has a *weight* that is the sum of the link latency and the delay on the second node. We do not consider the delay of the first node to avoid duplicate delays when we combine elements to form a path. We can sort these $N(N-1)$ segments by their weight, *i.e.* from the fastest segment to the slowest.

We now examine our three approaches to path formation, and for each, present results that compute the probability of building a path that is composed of segments belonging to the top x segments sorted by lowest segment weight. We quickly present the results in sequence, then compare them.

[1]We consider $N(N-1)$ items instead of N^2 because we do not admit self-loop, which means, in our case, twice the same node in the same path in consecutive position.

Tor-like paths. Let Ω be the set of tuples of segments of size $L-1$ from a population of $N(N-1)$ segments. We will select $L-1$ disjoint segments from $N(N-1)$. Therefore: $\Omega = \{(n_1, n_2, ..., n_{L-1}) : \ 1 \leq n_i \leq N(N-1) \ and \ n_1 \neq n_2 \neq ... \neq n_{L-1}\}$

Let x be a portion of segments from the $N(N-1)$ with the lowest "weight."

Claim 1. *Let A be the event that a Tor-like system selects a path within the best x elements of the space, with $L-1 \leq x \leq N(N-1)$, then the probability of A is:*

$$P[A] = \frac{\binom{x}{L-1}}{\binom{N(N-1)}{L-1}}$$

The key result is that as paths grow longer (higher L), $P[A]$ decreases, since the chance of choosing a poor (high weight) segment increases.

Disjoint Paths. In the case that we select the best out of k^L random disjoint paths, we can use the hypergeometric distribution to model our problem and compute the desired probability. The selection of k^L paths with no repetitions means choosing a set of tuples which size is $(L-1)k^L$ from a population of $N(N-1)$ segments: $\Omega = \{(n_1, n_2, ..., n_{(L-1)k^L}) : \ 1 \leq n_i \leq N(N-1) \ and \ n_1 \neq n_2 \neq ... \neq n_{(L-1)k^L}\}$

Let x again be the group of elements with lowest weight among the $N(N-1)$ total segments.

Claim 2. *Let B be the event that at least one of the k^L paths performs within the best x elements. Let consider for simplicity $P[C] = 1 - P[\overline{B}]$ with $(L-2) \leq x \leq N(N-1) - (L-1)k^L$ then the probability of C is:*

$$P[C] = 1 - \sum_{i=(L-1)k^L-(L-2)}^{(L-1)k^L} \frac{\binom{x}{(L-1)k^L-i}\binom{N(N-1)-x}{i}}{\binom{N(N-1)}{(L-1)k^L}}$$

Obviously it has to hold:

$$\sum_{i=0 \vee k^L(L-1)+x-N(N-1)}^{x \wedge k^L(L-1)} \frac{\binom{x}{i}\binom{N(N-1)-x}{k^L(L-1)-i}}{\binom{N(N-1)}{k^L(L-1)}} = 1$$

The key result here is that increasing k increases the search space and produces higher probability of a high-quality path.

Mesh Paths. In route meshes, we are building k^L paths using only $k^2(L-1)$ segments from the same population of $N(N-1)$ total elements. The selection of $k^2(L-1)$ items with no repetitions produces: $\Omega = \{(n_1, n_2, ..., n_{(L-1)k^2}) : \ 1 \leq n_i \leq N(N-1) \ and \ n_1 \neq n_2 \neq ... \neq n_{(L-1)k^2}\}$

Claim 3. *Let D be the event that at least one of the k^L paths performs within the best x elements. Let consider for simplicity $P[E] = 1 - P[\overline{D}]$ with $(L-2) \leq x \leq N(N-1) - (L-1)k^2$ then the probability of E is:*

$$P[D] = 1 - \sum_{i=(L-1)k^2-(L-2)}^{(L-1)k^2} \frac{\binom{x}{(L-1)k^2-i}\binom{N(N-1)-x}{i}}{\binom{N(N-1)}{(L-1)k^2}}$$

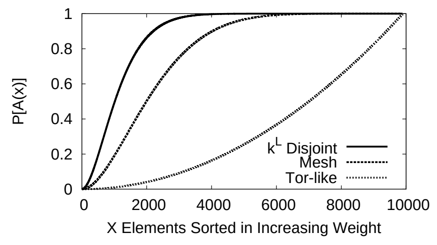

Fig. 3. Comparison of the different strategies with $L = 3$, $k = 2$.

Obviously it has to hold:

$$\sum_{i=0 \vee k^2(L-1)+x-N(N-1)}^{x \wedge k^2(L-1)} \frac{\binom{x}{i}\binom{N(N-1)-x}{k^2(L-1)-i}}{\binom{N(N-1)}{k^2(L-1)}} = 1$$

In Figure 3 we compare the three strategies using typical values for L and k (these results are representative of those achieved using different parameter values). Clearly Rome approximates the ideal gain from searching k^L disjoint paths, and significantly outperforms the single path approach in Tor.

V. Evaluation

We now present measurement results from a prototype of Rome running on the PlanetLab Internet testbed.

Setup. We installed our software on 241 different PlanetLab machines across the world, and ran our experiments for over five days. This distributed network of shared machines closely matches conditions on the public Tor network reported by a recent measurement study [10]. Using this testbed, we ran experiments for the Rome algorithm, a Tor-like path construction algorithm, and for optimal path selection from k^L random disjoint paths. latency.

Implementation. We implemented the Rome algorithm, and a simplified version of the Onion Router in Python. For each datapoint, a source node selected a "best path" according to each path formation approach, then measured its end-to-end latency via repeated probes (averaging results of 5 probes). Each plot of the mesh and the Onion routing paths in the following graphs is from 1000 such constructions, for every combination of parameters. Plots of k^L disjoint paths are from a subset of 15K disjoint onion path construction measurements.

Metrics. In the graphs that follow, we plot the cumulative distribution of the delays measured in our experiments. The x-axis shows the actual delay values, while the y-axis shows the percentage of paths that have latency less than or equal to the corresponding latency on the x-axis.

Onion Routing Performance. 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 $L = 6$.

Mesh Performance. Figure 4 shows the latencies from mesh-based paths for $L = 3$, as k increases from 2 to 4. We

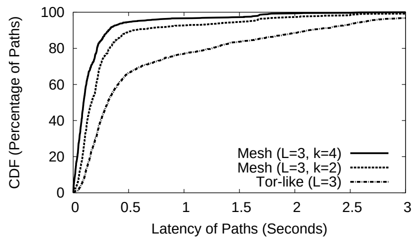

Fig. 4. Comparison of the Mesh-based and Tor-like path latencies for $L = 3$ and different values of k. All legends sorted in the order of plots.

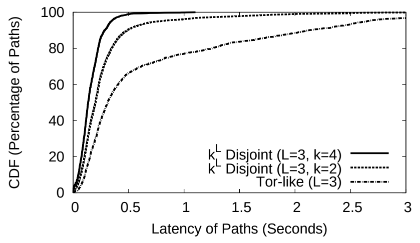

Fig. 5. Comparison of Tor-like and Min(k^L disjoint) path latencies for $L = 3$ and different values of k. All legends sorted in the order of plots.

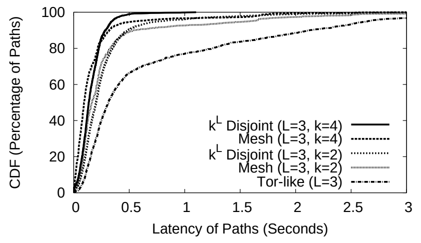

Fig. 6. Comparison of Mesh-based and Min(k^L disjoint) path latencies for $L = 3$ and different values of k. All legends sorted in the order of plots.

also plot the Onion routing latencies for $L = 3$ for reference. In all cases, nearly 90% of the mesh paths stay under 1 second delay, while only around 70% of the onion paths stay under 1 second for $L = 3$ (and only 40% for $L = 4$). We can notice that increasing k in the mesh improves the performance, but the performance improvement diminishes for larger values of k. We ran our experiments for larger values of L, and observed similar improvements. However, we noticed that Onion routing performance degrades significantly for larger L. We do not present the graphs for larger L for space reasons.

Performance of k^L Disjoint Onion Paths. Figure 5 compares the performance of k^L disjoint Onion paths with a single Onion path's performance. We see that k^L is significantly better, and that the performance improves with increase in the value of k, just like the performance of mesh.

Comparing k^L Disjoint Onion Paths with a Mesh Path. Figure 6 compares the performance of k^L disjoint Onion paths with a single mesh for the same values of k and L. The main point to take away from the graph is this: the performance of a single mesh is almost close to the performance of k^L disjoint Onion paths for same k and L. In fact, for nearly 80% of the paths, the performance of the mesh is slightly better than the performance of k^L paths. This is a huge advantage of the mesh considering the amount of traffic generated to test k^L paths and the loss in anonymity encountered in the process. The performances of mesh is consistently close to k^L for different values of k and L as shown in the Figure.

VI. Related Work

Prior work has examined the impact of timing analysis attacks on anonymous systems. Some assume global passive attackers, and use packet counting or inter-packet arrival delays to correlate flows [23]. The disclosure attack [8] requires a large number of observations to compromise endpoint anonymity, but has been improved by exploiting statistical properties of its observations [2]. Finally, the HittingSet attack on MIXes [9] used an alternative statistical model to reduce endpoint anonymity.

Other projects have examined the tension between anonymity and performance in anonymous protocols. [1] showed the vulnerability of Tor's biased path selection algorithms to manipulation. Snader and Borisov proposed opportunistic bandwidth measurements to improve end-to-end performance on Tor [17]. Finally, there are more powerful and general timing analysis attacks that assume active attackers that can compromise flows by inserting watermarks or stepping stones into flow, and analyzing traffic characteristics at the destination [18], [19], [20]. While powerful, these attacks are difficult to perform in practice.

VII. Conclusions and Ongoing Work

This paper proposes Rome, a user-managed approach to tuning tradeoffs between performance and anonymity for Chaum-MIX anonymous protocols. We show that together with the testdrive algorithm, route meshes provide efficient search for optimal paths using only end-to-end performance measurements. Even small meshes (k=2 or 4) have a dramatic effect on reducing end-to-end latency in our wide-area tests. Finally, our analysis shows that this dramatic increase in performance comes at a very low cost in anonymity.

We are currently investigating several interesting extensions to route meshes. First, testdrive can be easily extended to build node-disjoint optimal backup paths along with optimal paths with no additional measurement costs. Second, we are actively modifying Tor source code to evaluate Rome on the public Tor network. When completed, source code for Rome extensions to Tor will be made publicly available.

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] Danezis, G. The traffic analysis of continuous-time mixes. In *Proc. of PET* (May 2004).

[3] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *USENIX Security* (2004).

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

[5] Freedman, M. J., and Morris, R. Tarzan: A peer-to-peer anonymizing network layer. In *Proc. of CCS* (Nov. 2002).

[6] Holahan, C. Viacom vs. youtube: Beyond privacy. BusinessWeek, July 2008.

[7] Katti, S., Cohen, J., and Katabi, D. Information slicing: Anonymity using unreliable overlays. In *nsdi* (2007).

[8] Kesdogan, D., Agrawal, D., and Penz, S. Limits of anonymity in open environments. In *Proc. of IH* (October 2002).

[9] Kesdogan, D., and Pimenidis, L. The hitting set attack on anonymity protocols. In *Proc. of IH* (May 2004).

[10] 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.

[11] Motwani, R., and Raghavan, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[12] Nambiar, A., and Wright, M. Salsa: A structured approach to large-scale anonymity. In *Proc. of CCS* (Nov 2006).

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

[14] PUTTASWAMY, K. P. N., ET AL. Defending anonymity against predecessor attacks in bluemoon. In *Proc. of ICNP* (October 2008).

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

[16] SCIENTIFIC AMERICAN. Internet eavesdropping: A brave new world of wiretapping, August 2008.

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

[18] WANG, X., CHEN, S., AND JAJODIA, S. Tracking anonymous peer-to-peer voip calls on the internet. In *Proc. of CCS* (November 2005).

[19] WANG, X., AND REEVES, D. Robust correlation of encrypted attack traffic through stepping stones by manipulating of interpackets delays. In *Proc. of CCS* (2003).

[20] WANG, X., AND REEVES, D. Network flow watermarking attack on low-latency anonymous communication systems. In *Proc. of IEEE Symposium on Security and Privacy* (2007).

[21] WRIGHT, M., ET AL. An analysis of the degradation of anonymous protocols. In *Proc of NDSS* (February 2002).

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

[23] ZHU, Y., ET AL. Correlation attacks and countermeasures in mix networks. In *Proc. of PET* (May 2004).

[24] ZHUANG, L., ZHOU, F., ZHAO, B. Y., AND ROWSTRON, A. Cashmere: Resilient anonymous routing. In *Proc. of NSDI* (May 2005).