SIMULATION OF CIRCUIT CREATION IN TOR: PRELIMINARY RESULTS

WILLIAM BOYD, NORMAN DANNER, AND DANNY KRIZANC

Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT USA
e-mail address: whboyd@wesleyan.edu

Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT USA
e-mail address: ndanner@wesleyan.edu

Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT USA
e-mail address: dkrizanc@wesleyan.edu

Abstract. We describe a methodology for simulating Tor relay up/down behavior over time and give some preliminary results.

1. Introduction

Tor [Dingledine et al., 2004] is a low-latency anonymity network. To communicate anonymously with Bob, Alice forms a *circuit* through Tor consisting of three relays and tunnels her TCP traffic through that circuit. Layered (onion) encryption is used to ensure that each relay only knows its immediate neighbors in the circuit. In order to minimize latency, Tor does not implement techniques such as batching and re-ordering which can be used to provide very strong anonymity guarantees. As such, it is usually assumed that Tor is vulnerable to timing analysis attacks. In such an attack, Eve controls the first and last relay in a circuit and is able to identify when two streams at either end actually belong to the same communication flow by examining timing information of the data packets. Analyses of these (and other) attacks are often done by constructing ad-hoc simulations of the network. Not only are such simulations sometimes poorly justified, their ad-hoc nature makes it difficult to compare attacks and defenses, because each simulation is different. Even fairly sophisticated models of Tor, such as the one given by Feigenbaum et al. [2007] in terms of I/O automata, often fail to take into account time-dependent relay behavior, even though this sort of behavior can be crucial for attacks.

We propose the development of more principled simulations that capture emergent behavior of the network. In particular, many attacks involve manipulating circuits through compromised relays; detecting such attacks often requires a notion of what constitutes

1998 ACM Subject Classification: C.2.2 [Computer-Communication Networks] Applications—Tor; I.6.4 [Simulation and Modeling] Model Validation and Analysis; I.6.8 [Simulation and Modeling] Types of Simulation—Discrete event.

Key words and phrases: Anonymity; clustering; hidden Markov models; simulation; Tor.

"normal" relay performance and uptime. Replays of past behavior provide an accurate but necessarily limited model of behavior. In this short paper, we describe our work in progress to (i) characterize the behavior of individual Tor relays, (ii) capture this characterization in a generative model, specifically, a set of hidden Markov models, (iii) use these models to generate representative Tor relay behavior, and (iv) evaluate the "goodness-of-fit" of this generated behavior. In the longer term, we hope to have a generative model of many observable quantities of the Tor network such as relay failures, latency, throughput, etc. Our hope is that such a generative model could be used as a uniform testbed for analyzing attacks and defenses as well as proposals for changes to the Tor protocol itself. Our generative model is based on observations of the deployed network rather than explicitly simulating details of the Tor protocol, network links, etc., so it effectively adapts itself to changes in the network, without changes in the architecture of the model itself.

Our starting point is work of Danner et al. [2009], who analyze a denial-of-service attack described by Borisov et al. [2007]. In this attack, Eve compromises some number of entry and exit relays. If a circuit passes through a compromised relay but it is not the case that both the entry and exit relay are compromised, then Eve kills the circuit, causing the client to reform the circuit. In this way, Eve increases the probability that she controls both the entry and exit relays. From there she can perform data correlation attacks (see, e.g., Murdoch and Zieliński [2007]) to confirm that streams at either end of the circuit are actually the same flow. As an example of problems we wish to avoid, Borisov et al. associate a single failure rate to each relay, and then use that to analyze reliability and security of the network under this attack; but this a very poor approximation of actual relay behavior, as actual relay failure can be heavily time-dependent.

In analyzing the effectiveness of this denial-of-service attack and possible detection algorithms, we collected *lifecycle* data for relays in the deployed network. A *lifecycle* for a given relay R is a function $\ell_R : \{0, 1, \dots\} \to \{0, 1\}$. The idea is that we "probe" R some number of times. A probe consists of constructing a circuit of the form (G, R, E) and downloading a small file through the circuit, where G and E are relays that we control. Probe t *succeeds* ($\ell_R(t) = 1$) if the file is successfully downloaded and otherwise the probe *fails* ($\ell_R(t) = 0$). The probe of R may fail for many reasons; there may be transient network failures; R may refuse to allow a circuit (perhaps because of bandwidth limiting); R may not be in consensus during the probe. We consider all failure modes to be the same for this analysis, though it would be easy to treat different modes as distinct provided we can determine the reason for the failure.

Our analyses of the denial-of-service attack rely on replaying collected data of this form [Danner et al., 2009]. For example, we propose a detection algorithm that constructs circuits, records whether the circuit construction was successful, and then uses this information to detect the attack, making use of the fact that circuits that include compromised relays are more likely to fail. To simulate the execution of the algorithm, we "run" it on the replayed data under the assumption that a subset of the highly-reliable relays are actually under the control of an attacker (who would then "kill" uncontrolled circuits). Because some (replayed) relays are down at the "time" the detection algorithm is run, we get a more realistic assessment of the effectiveness of the detection algorithm in the presence of noise.

We are interested in lifecycle data because our measurements have shown that the following is a very good predictor of successful circuit creation: perform a single probe of each relay in the deployed network; then predict that a given circuit creation attempt will

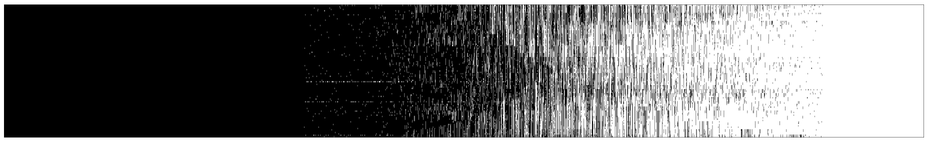

Figure 1: Visualization of lifecycle data for the deployed network. Each relay is represented as a vertical line. Each probe of the entire network is represented as a horizontal line. The pixel for relay R at probe t is white if $\ell_R(t) = 0$, black if $\ell_R(t) = 1$. Relays are sorted left-to-right in decreasing order of number of successes.

succeed if and only if each of the relays was successfully probed. Unsurprisingly, for any fixed relay R, the sequence ℓ_R is far from random; as an example, the lifecycles of the deployed Tor network as measured over a 30-hour period in mid-March 2011 (100 probes per relay) is visualized in Figure 1.

In this paper, we start with this collected lifecycle data to construct a *generative* model of relay lifecycles. The intent is that such a model displays the same emergent behavior as the collected data, while not being tied to replaying exactly the data that was collected. Our generative model is constructed as follows:

(1) Collect lifecycle data of the deployed network. We consider each relay's lifecycle to be a binary string, and all such strings have the same length. Let the set of lifecycle strings be S.
(2) Use an appropriate clustering algorithm to group the lifecycles into a comparatively small number of clusters.
(3) For each cluster C, train a hidden Markov model λ_C that produces lifecycles "similar" to the ones in C.
(4) To generate a new lifecycle, choose a λ_C with probability $|C|/|S|$ and use it to generate a lifecycle. To generate a set S' of new lifecycles, do this $|S'|$ times.

In the remainder of this note we describe the model in more detail. We also discuss an important but oft-neglected aspect of simulation: measuring how well a proposed simulation models the observed phenomenon. There is still much work to be done; in particular, we want to be able to model more interesting behavior than just lifecycles, such as latency and throughput. But the current work leads us to believe that our approach has a lot of promise.

2. Lifecycle clustering

We expect relays (as described by their lifecycles) to fall into a number of relatively well-defined categories. For example, there are many relays which exhibit perfect uptime over the course of an observation run, and some which exhibit regular uptime/downtime cycles as their administrators take them off-line at night or over weekends. A clustering algorithm partitions a set of observations (lifecycles) without an *a priori* definition of what constitutes a category. By clustering lifecycles and choosing a representative of each cluster, we obtain a simplified model of the network, in which each representative can be used to generate lifecycles comparable to those in the cluster it represents. Furthermore, we can easily modify the network model to provide insight into the behavior of the real network under various assumptions. If, for example, we wanted to observe network behavior when

```
function kmedoids($X$ : set of binary sequences , $k$ : int) : list of $k$ sets of binary sequences
begin
    $Z$ ← [{$x_1$},...,{$x_k$}] (* $x_i$ chosen at random from $X$ *)
    $C'$ ← [{},...,{}] : list $k$ sets of binary sequences
    do
        $C$ ← $C'$
        $C'$ ← [{},...,{}] : list of $k$ sets of binary sequences
        foreach $x$ in $X$ do
            $i$ ← the $j$ that minimizes $E(x, Z(j))$
            add $x$ to $C'[i]$
        endf
        for $i$ ← 0 to $k$ do
            $Z[i]$ ← center of $C'[i]$
        endf
    while $C' \neq C$
    return $C$
end
```

Figure 2: Pseudocode for k-medoids approximation with Lloyd's algorithm. $Z[i]$ is the center of cluster i and $C[i]$ and $C'[i]$ are the old and new i-th cluster. A bit of restructuring eliminates the unnecessary last calculation of centers and also makes it easy to determine when a lifecycle has moved from one cluster to another.

high-profile relay operators are attacked or their relays blocked from access, we could reduce the proportion of several high-uptime clusters in the model.

A clustering algorithm depends upon a metric on the objects being clustered. We have considered several, but here will just concentrate on a version of edit-distance. To that end, we define $E(s, s')$ to be the number of "edits" that must be made to s to obtain s'. The edits that we allow are flipping a single bit and rotation by one position (the bit that is "pushed off the end" is tacked on at the other end). Our rationale is that the first type of edit can help minimize the effect of transient phenomena when identifying similar lifecycles. Rotation also allows us to consider as similar two relays that have similar cyclic behavior that start at slightly different times. For example, it is well-known that many relays are shut down at night (and this can be seen in the visualization in Figure 1). We end up considering such relays to be similar if "night-time" is approximately the same, but they become less similar as that time diverges.

Xu and Wunsch [2005] provide a fine survey of a wide variety of clustering algorithms. Here we just consider *k-means* clustering, which groups objects into a pre-determined number k of clusters in a way that attempts to minimize the average value of $E(x, c)$ over all objects, where c is the center of the cluster containing x. An exact solution is not feasible, but the standard approximation algorithm, Lloyd's algorithm, gives good results in time $O(nk)$ and space $O(n+k)$ where n is the number of sequences. In Lloyd's algorithm, initial centers are selected, and then items are assigned to the cluster with the nearest center. New centers are selected for each cluster, items are again assigned to the cluster with the nearest center, and the process is repeated until it converges. k-means clustering is typically used with Euclidean distance, for which it is easy to define the center of a cluster. With edit-distance, computing this new center is not so straightforward. Thus we define the center of a cluster C to be the sequence $c \in C$ that minimizes $\sum_{s \in C}(E(s, c))^2$. An algorithm that defines "center" in this way is sometimes referred to as a *k-medoids* algorithm (see [Kaufman and Rousseeuw, 1990, Ch. 2]). The k-medoids algorithm we use is shown in Figure 2.

3. Hidden Markov models

Hidden Markov models are a commonly used statistical model for temporal processes that obey the Markov property: the behavior of the process at any given point in time depends on at most a constant, finite amount of its past state. Rabiner [1989] is a standard reference, and we follow his presentation. If we have in hand a sequence of observations and want to determine a Markov model that is likely to have generated that sequence, we generate a *hidden* Markov model, which consists of the following data:

- A finite set Q of states.
- An alphabet Σ of output symbols; these correspond to our observations (so in the case at hand, $\Sigma = \{0, 1\}$).
- A state transition probability distribution function $A : Q \times Q \to [0, 1]$ giving the probability of a transition from one state to another.
- An output symbol emission probability distribution function $B : Q \times \Sigma \to [0, 1]$ giving the probability of a given symbol being observed from a given state.
- A start state probability distribution function $\pi : Q \to [0, 1]$ giving the probability that a given state is selected as the first state in a sequence.

A (non-hidden) Markov model with states Q, state transition probability distribution A, and initial state q_0 is a special case of a hidden Markov model: $\Sigma = Q$, $B(q, q') = 1$ if $q' = q$ and 0 otherwise, and $\pi(q) = 1$ if $q = q_0$ and 0 otherwise.

An HMM λ generates an observation sequence $\mathcal{O} \in \Sigma^*$ as follows:

(1) Choose an initial start state using π; call this the current state s.
(2) Repeat the following some finite number of times:
    (a) Use $B(s, \cdot)$ to choose a symbol to emit.
    (b) Use $A(s, \cdot)$ to choose the new current state s.

Given an observation sequence, we would like to determine the HMM that is most likely to have generated that sequence. The solution to this problem breaks down to solving the following three problems:

(1) Given an HMM λ and observation sequence $\mathcal{O}$ determine $\Pr[\mathcal{O} \mid \lambda]$, the probability that λ generates $\mathcal{O}$. There is a straightforward dynamic programming algorithm known as the *forward* algorithm to compute this probability.
(2) Given an HMM λ and observation sequence $\mathcal{O}$ determine the state sequence for λ that is most likely to generate $\mathcal{O}$. The *Viterbi* algorithm is a dynamic programming algorithm to compute the state sequence.
(3) Given an HMM λ and observation sequence $\mathcal{O}$, adjust the parameters of λ to obtain a new HMM λ' that is more likely to generate $\mathcal{O}$. The *Baum-Welch* algorithm performs what is essentially a hill-climbing procedure to estimate new parameters based on the solutions to the previous two problems.

For details of all three algorithms, we refer the reader to Rabiner [1989].

Iterating the Baum-Welch algorithm finds a local maximum of $\Pr[\mathcal{O} \mid \lambda]$. Because the search space is very hilly, in practice one applies the algorithm from many different randomly-chosen initial positions and chooses the best local maximum. The Baum-Welch algorithm trains an HMM on a single observation sequence, whereas we will want an HMM that is trained on a set of such sequences. A simple solution to this problem is to train the HMM on the concatenation of the sequences in the sets. Assuming we do not allow too

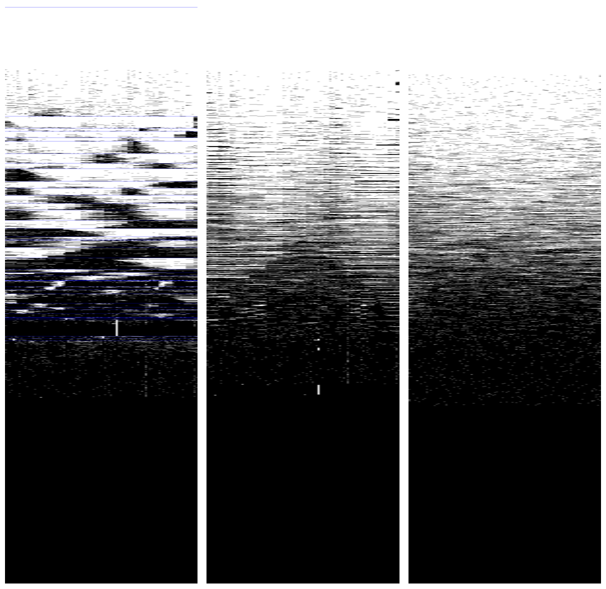

Figure 3: Collected data and our simulation. The collected data is the left and middle figures and the simulated data is the right figure. The middle figure is the same as Figure 1. The left figure shows the same data after clustering; clusters are sorted by failure rate of the center of the cluster, and lifecycles within a cluster are sorted by failure rate. Blue lines (visible in electronic version) separate the clusters. Pixel values are as in Figure 1.

many states in the resulting HMM, the spurious transitions at the concatenation boundaries should not have a significant effect on the result.

4. Preliminary results

We cluster the lifecycle data displayed in Figure 1 using k-medoids clustering with edit-distance. We follow a rule-of-thumb that says that $\sqrt{N/2}$ is often a reasonable number of clusters when there are N items in the original dataset. We observed about 3500 relays, so we use 41 clusters. For each computed cluster C, we concatenate the sequences in C into a single sequence, then apply the Baum-Welch algorithm to train an HMM λ_C for the cluster. In our runs, the median number of states is 8 (with a maximum of 10). We then produce a simulation of the Tor network by repeating the following 3500 times:

(1) Choose an HMM λ; λ_C is chosen with probability $|C|/3500$.
(2) Generate an observation sequence using λ as described above.

In Figure 3 we display a visualization of the original data along with our simulation. As we can see, some (but not all) of the emergent behavior seems to have been captured in this simulation. Although we do not see the "peak" of successfully-probed relays (corresponding to the above-mentioned diurnal pattern that has been already observed) strongly, it is visible. And likewise we see some of the additional uptime behavior of at the edges. It looks as though there may have been some transient problem in our data collection (indicated by the two visible vertical white lines) but that same problem does not show up in the simulated lifecycles; this might be considered to be either a feature or a bug of the model.

5. Assessment

To properly assess the quality of any simulation, we need a measure by which to do such an assessment. This is a general problem in simulation studies, and appears to be a very difficult problem. However, we believe we can make a start by proposing the following. We think that a quality measure ought to satisfy the following two properties:

(1) The best simulation of a data set is the data set itself.
(2) A simulation that cannot produce a given data set is a poor simulation.

Define a *model* $\mathcal{M}$ of lifecycle data to be any process that, given n, produces a set of n lifecycle sequences. Furthermore, for a given set S of lifecycle sequences, we require that $\Pr[S \mid \mathcal{M}]$, the probably that $\mathcal{M}$ produce S on input $|S|$, be defined. We can then define the quality of the model relative to a fixed set S_0 (such as our collected data) to be $\Pr[S_0 \mid \mathcal{M}]$. With this quality measure:

(1) The model $\mathcal{M}_0$ that just returns S_0 is the highest-quality model possible, since $\Pr[S_0 \mid \mathcal{M}_0] = 1$.
(2) A model $\mathcal{M}$ that cannot reproduce S_0 is a lowest-quality model, as $\Pr[S_0 \mid \mathcal{M}] = 0$.

With this quality measure, we compared several combinations of different clustering algorithms and distance metrics, and all resulted in seemingly-close quality. Other models that we expect to be poor in fact compare poorly to our model. An example is a model that produces $|S_0|$ sequences, each of which is a Bernoulli process that chooses failure with probability equal to the average failure rate of all relays represented in S_0.

This notion of quality is very preliminary and needs work. For example, $\mathcal{M}_0$ is the highest-quality model possible relative to S_0, but would be a lowest-quality model relative to a dataset S that is identical to S_0 except for a single change of one bit of one lifecycle. Nonetheless, we feel it is a useful starting point for discussion.

6. Conclusions

We have presented a generative model of Tor relay lifecycles based on collecting such data, clustering it, and then training hidden Markov models on the resulting clusters. Preliminary investigations indicate that this is a promising approach to simulating at least some properties of Tor. We have also presented an initial proposal of quality-measurement for any proposed model.

More work needs to be done to determine appropriate distance metrics and clustering algorithms. Showing that we can use a similar methodology to accurately simulate more sophisticated phenomena (such as inter-relay latency and throughput) is a necessary step. Finally, we need to understand how to define a generative model in general, and to compare different generative models. Little work has been done in this area, though Kleinberg [2003] and Carlsson and Memoli [2010] provide some very interesting ideas on the related question of assessing the quality of clustering algorithms.

References

N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, pages 92–102, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-703-2. doi: 10.1145/1315245.1315258.

G. Carlsson and F. Memoli. Classifying clustering schemes. 2010. http://arxiv.org/abs/1011.5270v2.

N. Danner, D. Krizanc, and M. Liberatore. Detecting denial of service attacks in Tor. In R. Dingledine and P. Golle, editors, *Financial Cryptography and Data Security: 13th International Conference, FC 2009*, volume 5628 of *Lecture Notes in Computer Science*, pages 273–284. Springer-Verlag, 2009. doi: 10.1007/978-3-642-03549-4_17.

R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

J. Feigenbaum, A. Johnson, and P. Syverson. A model of onion routing with provable anonymity. In S. Dietrich and R. Dhamija, editors, *Financial Cryptography and Data Security: 11th International Conference, FC 2007*, volume 4886 of *Lecture Notes in Computer Science*, pages 57–71. Springer-Verlag, 2007.

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Mathematical Statistics. Wiley, 1990.

J. Kleinberg. An impossibility theorem for clustering. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 446—453, 2003.

S. J. Murdoch and P. Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In N. Borisov and P. Golle, editors, *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies, PET 2007, Ottawa, Canada*, volume 4776 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 2007. doi: 10.1007/978-3-540-75551-7_11.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. doi: 10.1109/5.18626.

R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005. doi: 10.1109/TNN.2005.845141.