Confiabilidade de Sistemas Distribuídos

Rafael Mira
59243

Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa

June 14, 2025

1 Introduction

This work presents an implementation of the Trust-Aware Path Selection (TAPS) algorithm, focusing specifically on the TrustAll variant with the "Countries" policy, which assumes all clients use the same path selection logic.

TAPS is designed to improve Tor's security against deanonymization attacks. Standard Tor users are vulnerable to adversaries who can perform effective traffic-correlation attacks by observing traffic at both the beginning and end of a circuit. To address this, TAPS allows users to select paths based on a trust model. The primary goal is to choose guard and exit nodes to minimize the probability of an adversary observing both ends of the circuit, while simultaneously ensuring the user's path choices blend in with others to maintain anonymity.

2 Implementation

My implementation consisted in following the TAPS paper to try to mimic the suggested implementation based on trust and still performing a bandwidth-weighted choice to balance security with performance. I also tried to do some very small changes to the original algorithm by trying to explicitly filter nodes based on the ASN and the Country. I wanted to try and reduce even further the chance of a guard or exit node to be in adversary countries.

3 Edge Cases Analysis

3.1 No valid paths

If the algorithm cannot find a valid guard, exit, or middle node for the given input, it will not return a circuit and will log the failure. This behavior is valuable for clients who require high trust assurance and would prefer the connection to fail rather than use a path that does not meet their standards. These users can then choose to lower their trust requirements and try to build a circuit again.

3.2 How to calculate trust for Countries in multiple Alliances

This scenario was handled by attributing the minimum trust found in each of the alliances that the country belongs to. E.g., If Russia and Portugal form an alliance and Russia and USA also form an alliance, Russia trust will be the lowest value found in input for the alliances.

What I could have tested here is the effect of applying a specie of discount to countries in shared alliances. E.g., Russia-Portugal have trust of 0.4 and Russia-USA have trust of 0.2, Russia will have trust of 0.2 because it's the lowest of the two but Portugal will have 0.4. I could potentially apply some penalty to Portugal for being in an alliance with a lower trusted country.

3.3 How to calculate trust for Countries without Alliances

The algorithm handles this via a DEFAULT_TRUST_SCORE parameter, and testing revealed this value has a significant impact on the final circuit's security. This parameter is within this bounds [0;1].

My evaluation showed that for unlisted countries, setting a default trust of 1.0 (maximum trust) caused the algorithm to select guard nodes in adversary countries far less frequently. When the default trust was low, such as 0.1 or 0.5, the percentage of guards chosen in adversary countries was noticeably higher. This behaviour makes sense according to the _find_secure_relays function.

The key is that after calculating each node's security, the algorithm orders them from the most trusted to the least. When a low default trust is used, nodes in explicitly defined adversary countries can end up ranked relatively high on the list. Because they are processed first, this increases their chance of getting into the candidate pool for guard or exit positions. On the other hand, using a default trust of 1.0 effectively segregates the adversary nodes by ranking them very low, making it highly improbable that they will be selected.

To test the impact of the trust parameters, I ran a series of unit tests with different inputs. In each test, a path was evaluated based on whether the chosen guard and exit nodes were located in a country I considered an adversary. The definition of an "adversary" was dynamic, based on a trust threshold that was varied from 0.1 to 0.9. My results showed that the algorithm avoided placing guard nodes in adversary countries in almost all cases. The only exceptions occurred when the definition of an adversary was extremely strict, i.e., when a country was considered an adversary if its trust score was below 0.8 or 0.9. These infrequent failures represent abnormal scenarios, testing the algorithm to its limits under unusually high trust requirements.

3.4 Client and Destiny in the same country and the country is untrustable

In this scenario, the current algorithm's behaviour is to do the best of a bad situation. If a circuit exists, the algorithm will return it, but since the low-trust country will be a factor in every possible guard_security and exit_security calculation, all potential paths will have a low security score.

A potential solution is to introduce a user-configurable parameter, e.g. MINIMUM_ACCEPTABLE_TRUST, that would be a baseline for accepting a guard or exit node in the formation of a circuit. This parameter would be used within the _find_secure_relays function. After calculating the nodes trust scores, we could filter them based on this minimum trust value. If there the list of nodes was empty, this would not return a path, like in section 3.1. But even if there were nodes under this conditions, this possible implementation has some challenges.

One of them is that by imposing this restriction on the required trust, we would decrease the number of nodes in valid conditions to be a guard or a exit node. We could try to be more flexible by lowering the required bandwidth which would lead to a possible slower connection but it would increase the probability of forming a path.

Another challenge is that on the node selection, the algorithm becomes more deterministic. Deterministic in the sense that by decreasing the number of possible nodes a client can connect, the traffic patterns of that user would become more distinct than the others and he would be more susceptible to have his identity revealed.

This was just theoretical discussion as I didn't implemented neither tested this theory, i just found it a interesting point of view.

4 Evaluation

The metrics below are relative to the evaluate.py file where I run some experiments and considered these parameters when testing:

- For the security and performance metrics, I run 1000 times the algorithm and this is the aggregate. I considered a adversary the countries that had 0.5 or a lower trust score
- For the load distribution I ran it 10K times to have more fine grained results

Table 1: Comparison of TAPS Path Selection Strategies

Metric	**Pure TAPS**	**Hybrid (ASN & Country)**	**Hybrid (ASN Only)**
Security Metrics			
Guard in Adversary (%)	0.30	0.50	0.20
Middle in Adversary (%)	12.30	11.40	11.40
Exit in Adversary (%)	6.70	7.20	7.70
Path Correlation Vulnerability (%)	4.50	3.80	6.60
Performance Metrics			
Avg. Path Bandwidth (MB/s)	14.77	14.11	14.34
Load Distribution Metrics			
Unique Guards Chosen	3.606	3.573	3.555
Unique Exits Chosen	2.409	2.429	2.419

The results show a trade-off between security and performance. The Pure TAPS strategy provides the highest average bandwidth while still effectively avoiding adversaries at the guard position.

Adding a Hybrid (ASN & Country) filter makes the algorithm safest against path correlation attacks, reducing vulnerability to 3.80%. But comes at a bandwidth cost.

The Hybrid (ASN Only) filter shows the worst scenario, with slightly better performance (14.34 MB/s) than the full filter but weaker protection against correlation (6.60%). In this case, the pure TAPS implementation is better.

Regarding the decrease in performance, it's similar to the security scenario above, by removing potentially high-bandwidth exit nodes from the pool, we are lowering the average speed of the available candidates. The algorithm can no longer pick the absolute best exit if it's been filtered out, leading to a slightly slower average path.