This week update:

This was a busy week; this PDF contains all my updates.

If you prefer text-based meetings or would like to change the frequency to once every two weeks, please feel free to do so.

Summary:

I conducted some research on Betweenness Ranking (Centrality):

1. Attempting to convert ranking to a percentage to evaluate the validity of LoP.
2. Finding the best efficiency of this algorithm, O(n^2), and discussing approaches to make it log time.

After this, I feel that Betweenness Centrality may serve as an auxiliary to the Connectivity-based approach. So, here is the structure:

I think the Clustering-based approach is better than the Connectivity-based approach, but both approaches seem to create a brand new topic, which is not included in the first 15 papers I have read (they all focus on PoL). That's why I added a few papers for me to read during the next week, which will mainly relate to algorithms about colluding attacks and outlier detection in P2P networks.

Papers:

- Attack Robustness and Centrality of Complex Networks
- Outlier Detection in Network Data using the Betweenness centrality
- Collusion in peer-to-peer systems
- Blockchain Based Zero-Knowledge Proof of Location in IoT

Implementation(Outlier Detection):

Approach1: Connectivity-based:

Betweenness centrality (Betweenness ranking):

- Definition: a way of detecting the amount of influence a node has over the flow of information in a graph.
- Definition: a measure used in network analysis to identify the importance of individual nodes (or vertices) within a network.
- Meaning: Nodes with high betweenness centrality have a higher likelihood of occurring on a large number of shortest paths, indicating their significance in maintaining the overall connectivity of the network.
- Calculation: take every pair of the network and count how many times a node can interrupt the shortest paths (geodesic distance) between the two nodes of the pair.

- Note: The geodesic distance depends on the **shortest path** over existing edges, while the Euclidean distance depends on the **straight-line** distance.

Convert Int to %:

- [Paper15: APPLAUS]
  - It is often normalized by **dividing by the maximum possible number** of shortest paths that could pass through any node in the network, which depends on the network's size and structure.
  - Did not provide details about the attacks.
- [Paper02: Blockchain-based Proof of Location]
  - Follows APPLAUS
  - But they do talked about ROBUSTNESS A NALYSIS -> Colluding with other peers
    - didnt depends on betweenness ranking, mainly depends on limitation of bluetooth range P2P verification
    - When they talk about "letting the blockchain compute", there is a suspicion that all peers should calculate the betweenness ranking (mobile phone mining will consume a lot of resources)

      > Here is one example this paper doesn't makes sense:
      >
      > Two peers collude to build a false proof of location for one of them. The proof **may be** received by a honest peer that is far from the considered location and cannot contact the colluding nodes with the short-range communication technology. In this case, the honest peer may either immediately discard the proof of location (conservative approach) or make a more contemplated decision, **by evaluating** the betweenness of the two suspect peers, according to the procedure illustrated in Subsection 3.2.

    - I have multiple problems with this paper, I'll switch to other papers
- [Paper17: Outlier Detection in Network Data using the Betweenness centrality]
  - given Betweenness Ranking(Centrality), find the outliers based on **p-value method**
    - To compute the p-value, we would need to perform a statistical test on the betweenness centrality values of the nodes, like
      - t-test
      - chi-squared test
      - this paper: "hypothesis testing"
    - Soso need more time to understand how this statistical stuff because they didn't define "Outlier"
  - experiment: p-value was taken as 0.05. i.e., these experimental results have 95% confidence.
    - note: 95% = 1-0.05 such that if the p-value for a data point is less than or equal to 0.05, it is considered an outlier with a 95% confidence level.
  - Did not provide details about the attacks, definition of outlier.

Computationally Expensive:

- O(mn^3) for unweighted graphs
- O(mn) time, O(n^2) space if using breadth-first search
- **paper17**: O(n^2) time + O(n^2) for p-value method to compare each data point with other data points
- **Paper APPLAUS:**
  - B(v) = $\sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$
  - Improvement 1:
    - The closer vertices can be computed from the dependencies of the farther vertices
    - Combine with **paper15**: O(mn) time, O(n+m) space, which is still slow:

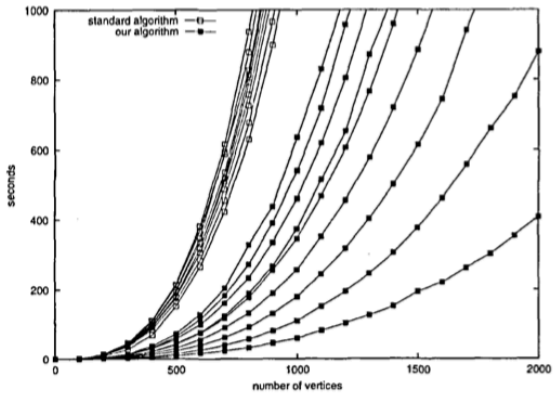

FIGURE 3 Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with 100–2000 vertices and densities ranging from 10–90%.

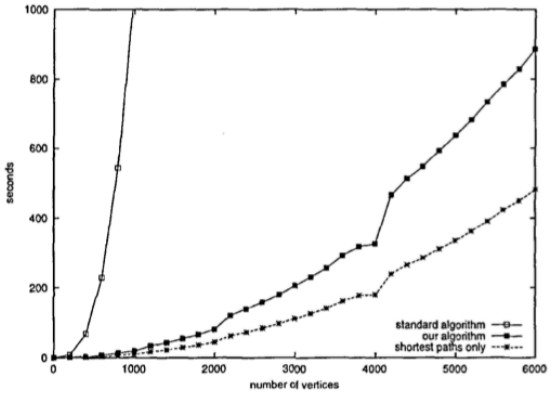

FIGURE 4 Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with constant average degree 20. The funny jumps are attributed to LEDA internals.

  - Improvement 2:
    - Considering all the concurrent (within 60 seconds delay) location proofs in a region, we first describe how to construct a pseudonym-correlation graph.
- Other Trade-Offs (Better than Improvement 2):

> Instead of setting hardcoded ranges for temporal and spatial regions, there are better approaches to make the algorithm run in logarithmic time while the result reflects more accurate betweenness values across the entire network.
>
> Take note of the following methods that create a betweenness ranking for the entire network at any time (granularity may depend on the initial request). Once calculated, these ranking can be resued/shared for different location queries.
>
> This is somewhat similar to B2B. Depending on the user's purpose, they could use approximation, which fast and will improves over time. Alternatively, they could calculate the requested region and then, when not busy, connect all regions.

  - Approximation algorithms: Instead of calculating betweenness centrality for all pairs of nodes, approximation algorithms compute the betweenness centrality using a random sample of node pairs or a fixed number of randomly chosen nodes as sources. This reduces the computational complexity and can provide reasonably accurate estimates of betweenness centrality in many cases. One

popular approximation algorithm is the Brandes' algorithm, which has a time complexity of O(nm), where n is the number of nodes and m is the number of edges.

- Parallel and distributed computing: Another way to address the efficiency issue is by leveraging parallel and distributed computing resources. By dividing the task of computing betweenness centrality among multiple processors or machines, the overall computation time can be significantly reduced.
- Incremental and decremental algorithms: In dynamic networks where nodes and edges are frequently added or removed, incremental and decremental algorithms can be used to update betweenness centrality values efficiently without recalculating them from scratch.
- Using heuristics: In some cases, it might be possible to apply heuristics or domain-specific knowledge to reduce the search space and focus on the most important nodes, thereby reducing the computational complexity.

Approach2: Clustering-based (Correlation clustering):

> this is better approch, and I will attach it's image from past weeks

5.2 Correlation Clustering

The above approach only considers two location proofs correlated if they occurred at the same time (e.g., within 60 seconds). However, in most case, a node may have to wait for a time period until its next location proof updating cycle. If the time delay between two location proofs is not too large, we should still consider them correlated. Therefore, if we consider each location proof as a vertex, we have the following definition of temporal-weighted proof-correlation graph.

Definition 7. *A weighted proof-correlation graph for region R is an undirected graph $G = (V, E)$, with vertex set $v_1, v_2, \ldots, v_n$, where $n = |V|$. Each vertex $v_i \in V$ represents a location proof LP_i in region R. Each positive edge $(v_i, v_j) \in E$ denotes that location proofs LP_i and LP_j share prover or witness while each negative edge $(v_i, v_j) \in E$ denotes that LP_i's prover is actually LP_j's witness, or vice versa. For each edge (i, j), let t_{ij}*

be the time difference between two location proofs represented by v_i and v_j, where $0 \leq t_{ij} < +\infty$, then we define weight function c_{ij} for each edge (i, j) as follows:

$$c_{ij} = \frac{1}{1 + t_{ij}}. \quad (12)$$

weight = 100% = low delay
weight = 0% = high delay

The weight function c_{ij} has both $0 \leq c_{ij} \leq 1$ for positive and negative edges. The smaller the time interval t_{ij} is, the larger the weight c_{ij} is.

Fig. 5b shows one example of weighted proof-correlation graph. Location proof v_1 and location proof v_2 share the same prover node, so the edge between them is positive with weight of 1. On the other hand, location proof v_5's prover is also location proof v_6's witness, so the edge is negative. Our goal is to have location proofs with position edges grouped together (e.g., v_1 and v_2), and to separate the location proofs with negative edges (e.g., v_5 and v_6). The reason is intuitive: the location proofs with negative edges are abnormal, since within similar time interval and similar space, there should be another location proof consisting of v_5's prover and v_6's witness. Fig. 5b shows how correlation clustering works on the graph. Since this problem is NP-hard, we transfer the above proof-correlation graph to a double-labeled graph.

With $G=(V, E)$, each edge (i, j) has a weight function c_{ij}. Let $e(u, v)$ denote the label $(\langle+\rangle, \langle-\rangle)$ of the edge (u, v). Let $E^{\langle+\rangle}$ be the set of positive edges (where $0 \leq t_{ij} \leq 1$) and let $G^{\langle+\rangle}$ be the graph induced by $E^{\langle+\rangle}$, $G^{\langle+\rangle} = (V, E^{\langle+\rangle})$. We then define $E^{\langle-\rangle}$ and $G^{\langle-\rangle}$ for negative edges in the same way. In

50% <= cij <= 100%

The *cut* of a circle is the cut induced by the set of vertices included in the circle. The *volume* of a set of nodes S, denoted by $vol(S)$, is the weighted distance of the edges with both endpoints in S:

positive?

our scenario, the weight value from one node to another can be treated as a measure of similarity, with high weight indicating similarity and low weight indicating dissimilarity. We can perform a correlation clustering over the graph, grouping nodes together who have higher positive weight with others, and separating nodes who have higher negative weight with others. Since the time complexity is too high for correlation clustering, we focus on approximation algorithms to achieve better performance.

Let OPT represent the optimal clustering algorithm, in which positive labeled edges are always in one cluster, while negative labeled edges are always between different clusters. Our goal is to propose an approximation algorithm close to OPT, by minimizing the weight of positive edges between clusters and the weight of negative edges inside clusters. Also, the approximation algorithm should maximize the weight of positive edges inside clusters and maximize the weight of negative edges between clusters. Given a clustering algorithm S, the difference between S and OPT is denoted as $d(S)$ which is the sum of the weights of negative labeled edges inside a cluster, plus weights of positive labeled edges between clusters in S. We then formulate the problem with linear programming and then give an approximation algorithm. Consider assigning a variable x_{ij} to each pair of vertices ($x_{ij} = x_{ji}$); $x_{ij} = 0$ if v_i and v_j are in a common cluster, and $x_{ij} = 1$ otherwise. As $1 - x_{ij}$ is 1 if edge (i, j) is within a cluster and 0 if edge (i, j) is between clusters, we have the following:

$$d(S) = \sum_{(i,j)\in E^{(-)}} c_{ij}(1 - x_{ij}) + \sum_{(i,j)\in E^{(+)}} c_{ij}x_{ij}. \quad (13)$$

Our goal is to find an assignment of x_{ij} to minimize the difference $d(S)$. We relax the requirement and get the following linear program:

$$\text{minimize} \sum_{(i,j)\in E^{(-)}} c_{ij}(1 - x_{ij}) + \sum_{(i,j)\in E^{(+)}} c_{ij}x_{ij}, \quad (14)$$

$$\text{subject to } x_{ij} \in [0, 1], \quad (15)$$

$$x_{ij} + x_{jk} \geq x_{ik}, \quad (16)$$

$$x_{ij} = x_{ji}. \quad (17)$$

We present a $O(logn)$ approximation solution to this linear program, which is the best approximation factor that can be achieved as proved in [7].

First we define a *circle* $C(i, r)$ with radius r around node v_i which consists of all nodes v_j such that $x_{ij} \leq r$. The *cut* of a set of nodes S, denoted by $cut(S)$, is the weight of the positive edges with exactly one endpoint in S:

$$cut(S) = \sum_{|v_j,v_k\cap S|=1,(j,k)\in E^{(+)}} c_{jk}. \quad (18)$$

$$vol(S) = \sum_{v_j,v_k\subset S,(j,k)\in E^{(+)}} c_{jk}x_{jk}. \quad (19)$$

Therefore, the *volume* of a circle is the volume of $C(i, r)$ including the fractional weighted distance of positive edges leaving $C(i, r)$. In other words, if $(j, k) \in E^{(+)}$ is a cut positive edge of circle $C(i, r)$ with $v_j \in C(i, r)$ and $v_k \notin C(i, r)$, then (j, k) contributes $c_{jk} \cdot (r - x_{ij})$ weight to the volume of circle $C(i, r)$. We include an initial volume I to the volume of every circle (i.e., circle $C(i, 0)$ has volume I). The correlation clustering algorithm on a weighted proof-correlation graph is shown in Algorithm 4. One advantage of this approach is that the correlation clustering algorithm can handle outliers naturally. Outliers will usually have a cluster of their own. Thus, when the algorithm ends, we can simply discard small isolated clusters.

Algorithm 4. Correlation clustering on proof-correlation graph

Input: queue G with all the nodes $v_i \in V$, weight function c_{ij} for each edge (i, j);

1: **while** G not empty **do**
2: dequeue $v_i \leftarrow G$;
3: $r = 0$;
4: **while** $cut(C(i, r)) > c \cdot ln(n + 1) \times vol(C(i, r))$ **do**
5: $r = r + min\{(d_{ij} - r) > 0 : v \notin C(i, r)\}$;
6: **end while**
7: output $C(i, r)$ as one of the clusters;
8: remove $C(i, r)$ and related edges from G;
9: **end while**

6 Performance Evaluations

In this section, we study the feasibility of deploying APPLAUS such as the computation and storage constraint, power consumption, and the proof exchange latency. We also use simulations to evaluate the performance of APPLAUS.

6.1 Prototype Implementation

To study the feasibility of our scheme, we have developed a prototype of APPLAUS based on the techniques presented in the previous sections. The prototype has two software components: client and server. The client is implemented in JAVA on Android Developer Phone 2 (ADP2), which is equipped with 528 MHz chipset, 512 MB ROM, 192 MB RAM, Bluetooth, and GPS module, and running Google Android 1.6 OS. It can communicate with the server anytime through AT&T's 3G wireless data service. The server is implemented on a T4300 2.1 GHz 3 GB RAM laptop. It stores the uploaded location proof records and manages corresponding indices using MySQL 5.0. We use two android phones to communicate with each other to test our solution.

The client code consumes only 80 KB of data memory. When running, less than 2.5 percent of the available