Abstract—Network topology discovery is the basis for any network management application. The problem of estimating internal structure and link-level performance from end-to-end measurements is known as network tomography. This paper proposes a novel approach to discover network characteristics, in particular, tree topology from end-to-end path metrics between OD (Origin – Destination) pairs. The approach is based on Prüfer encoding and decoding techniques to determine the topology.

Keywords-component; Network Tomography, OD, Path, Prüfer sequence, Topology inference, Traffic flow

I. INTRODUCTION

At the advent of advanced networking services such as Telepresence, IPTV and mobile interactive applications, it is critical that all components in the overall services delivery system provide comprehensive end-to-end quality of experience for all subscribers. This application layer requirement can be eventually mapped to a network layer requirement. Assessing and predicting internal network performance is of fundamental importance in traffic engineering problems. The problem of estimating internal network structure and link-level performance from end-to-end measurements is called network tomography. This paper describes network topology discovery from the path metrics between Origin-Destination (OD) pairs of a network. Inferring network topology by network discovery is the critical step in network tomography [2] and is essential to any network management applications. This paper focuses on the first step of determining the links in the network, given end to end measurements, prior to deriving link level metrics by Network Tomography.

This paper assumes the following:

1. We are interested in discovering the logical topology instead of physical topology. Physical topology represents the topology model of layer-1 (the physical layer) whereas the logical topology represents the topology model at layer-2 or layer-3 of the OSI model. The logical topology discovery can be done in two different ways: (i) setting up probes in the network, collecting management information at internal nodes periodically and generate topology and report other network management related information. (ii) Inferring the topology and other network characteristics from end-to-end measurements. We will focus on approach (ii) to infer the topology.

2. The underlying network is actually a tree at any instance of communication. It means that there exists a unique path between any OD pairs in the network at a given point of time.

3. The existing path discovery tools such as traceroute, ping will provide the path metrics between any OD pair in the network.

4. The network considered is undirected. It means that the forward and reverse flows are through the same link, though in reality it will be through different links. The proposed approach can be extended to directed network.

5. The network considered is connected or strongly connected if there exists a path between any pair of nodes in the network.

The paper is organized as follows. In section II the problem is defined. Path metric between OD pairs is defined in section III. Section IV describes Prüfer encoding and decoding. Section V proposes topology discovery algorithm using Prüfer encoding technique from path metric. An example is shown in section VI. Section VII concludes the paper.

II. PROBLEM STATEMENT

Graphs are the mathematical constructs best suited to represent networks. A network in its simplest form is a set of nodes or vertices joined together in pairs by edges. In the context of communication networks, vertices are Autonomous Systems, routers, switches, computers, workstations whereas the edges are links connecting a pair of such vertices. A set of connected links leading from an origin node to a destination node is referred to as a “path”. Consider a fixed or static routing protocol in that for each OD pair, the same unique path always carries the traffic.
between origin and destination and vice-versa. The path length or OD distance is measured as the number of links connecting an OD pair.

The problem is to infer the topology (tree) and its internal nodes given the path measurements or distance between every OD pairs in a network.

III. PATH METRIC

Let \( T = (V, E) \) denote the tree with node set \( V \) and link set \( E \). Let \( R \subset V \) denote the set of leaf nodes in the tree. Let \( R \times R \) represent distinct OD pairs of the network. A walk in a graph is a finite non-null sequence \( W = v_0e_1v_1e_2v_2...e_kv_k \), whose terms are alternately vertices and edges, such that, for \( 1 \leq i \leq k \), the ends of \( e_i \) are \( v_i \) and \( v_{i+1} \). In other words, a walk of length \( k \) is a sequence of vertices \( v_0, v_1, ..., v_k \) such that \( v_i \) is adjacent to \( v_{i+1} \) for each \( i \). If the edges \( e_1, e_2, ..., e_k \) of a walk are distinct, it is called a trail. If the vertices \( v_0, v_1, ..., v_k \) are distinct, it is called a path. A closed walk is a walk of length \( k \) such that \( v_0 = v_k \). A cycle is a closed walk where none of the vertices repeat except for the first and the last. The degree of a vertex is defined as the number of vertices its adjacent to, or the number of edges incident with it. Refer [1].

A graph is said to be connected if for all pairs of vertices there exists a walk. A tree on \( n \) vertices is a connected graph that contains no cycles. The distance matrix or path length matrix \( D \) for OD pairs is defined as follows:

Entries \( d_{ij} = d_{ji} = \) path length between OD pair \( i \) and \( j \), where \( i \) and \( j \) ranges from 1 to \( |R| \). \( d_{ii} = 0 \). Refer [3]. This matrix is a symmetric matrix.

This path metric could be obtained using tools like traceroute. The “traceroute” program verifies whether an end-to-end IP path is operational, and provides information on the intermediate systems to be found along the IP (Layer-3) path from the origin to the destination. Traceroute uses ICMP echo response with different TTL (Time-To-Live) packets to determine the path. In reality, however, some routers discard ICMP (Internet Control Message Protocol) messages and thus make discovery incomplete. In such cases, active probing can be used to determine the path between given OD pair. Probing techniques such as those used in [4] will be considered.

IV. PRÜFER SEQUENCE

A Prüfer sequence is a sequence of \( n-2 \) numbers or labels, each being one of the numbers 1 through \( n \), with repetitions allowed. It should be noted that there are \( n^{n-2} \) Prüfer sequences for any given \( n \). This can be easily proved: by definition, there are \( n \) ways to choose each element of Prüfer sequence of length \( n-2 \). Since there are \( n-2 \) elements to be determined, in total we have \( n^{n-2} \) ways to choose the whole sequence. Given a labeled tree with vertices labeled by \( 1, 2, 3, ..., n \), the Prüfer encoding algorithm given below outputs a unique Prüfer sequence of length \( n-2 \). Refer [5].

**Algorithm: Prüfer encoding**

**Input:** A labeled tree with vertices labeled \( 1, 2, 3, ..., n \)

**Output:** A Prüfer sequence

Repeat \( n-2 \) times

\( v = \) the leaf with the lowest label

Print the label of \( v \)'s unique neighbor in the output

Remove \( v \) from the tree

Example:

![Prüfer decoding](image1)

**Figure 1** Prüfer encoding

The Prüfer decoding algorithm provides the inverse algorithm, which finds unique labeled tree \( T \) with \( n \) vertices for a given Prüfer sequence of \( n-2 \) elements.

**Algorithm: Prüfer Decoding**

**Input:** A labeled tree with vertices labeled \( 1, 2, 3, ..., n \) \( P = \) the input Prüfer sequence

\( n = |P| + 2 \)

\( V = \{1, 2, 3, ..., n\} \)

Start with \( n \) isolated vertices labeled \( 1, 2, 3, ..., n \)

for \( i = 1 \) to \( n-2 \) do

\( v = \) the smallest element in \( V \) that does not occur in \( P \)

Connect \( v \) to \( p_i \)

Remove \( v \) from \( V \)

Remove \( p_i \) from \( P \)

Connect the last two vertices remaining in \( V \)

Example:

\( P = (3, 3, 4, 4) \)

\( V = (1, 2, 3, 4, 5, 6) \)

\( E = \{(1, 3); P = (3, 4, 4); V = (2, 3, 4, 5, 6) \}

\( E = \{(1, 3), (2, 3); P = (4, 4); V = (3, 4, 5, 6) \}

\( E = \{(1, 3), (2, 3), (3, 4); P = (4); V = (4, 5, 6) \}

\( E = \{(1, 3), (2, 3), (3, 4), (4, 5); P = []; V = (4, 6) \}

\( E = \{(1, 3), (2, 3), (3, 4), (4, 5), (4, 6) \}

Example:

![Prüfer decoding](image2)

**Figure 2** Prüfer decoding
Key observation:

1. Any vertex $v$ of $T$ occurs $d(v)$-1 times in Prüfer sequence $P$, where $d(v)$ is the degree of vertex $v$.
2. The vertices of degree 1, i.e., the leaves, in $T$ never appear in $P$.

V.  TOPOLOGY DISCOVERY

In our problem the key assumption is that we have path length between all OD pairs in the network represented as a matrix. The OD pairs denote the leaves in the underlying tree $T$. The key fact is that there exists a unique tree $T$ for the given path matrix between all its leaves. This can be easily shown that if there is another path length between a given OD pair, then there is a cycle, which is a contradiction by the definition of tree. The idea is to relate this uniqueness relationship between path matrix, Prüfer sequence and labeled tree.

The leaf nodes are labeled by 1, 2, 3, ..., $r$ and denoted by the set $R$. The problem is to find the exact number of internal nodes connecting these leaf nodes and derive the Prüfer sequence.

Since the lowest label among the leaves is well defined, we consider $m$ to be the first internal node connecting to the lowest label. The label $m$ is $r+1$. Thus we generate unique labels in the increasing order for every internal node discovered. Let us consider the lowest label as $l$. Now $m$ is in the path of all OD pairs that has $l$ as origin or destination, and subsequently the path length to and from $m$ is reduced by one as compared to path lengths to and from $l$. As per Prüfer encoding, $l$ can removed and its unique neighbor $m$ is added to the Prüfer sequence. The vertex $m$ is also added to the path matrix with appropriate path lengths to the rest of the vertices.

Now there are two possible scenarios with respect to $m$: i. $m$ is connected to another leaf node with lowest label (other than $l$). This can be seen by checking if there exists a path with length 1 between a leaf node and $m$. ii. $m$ becomes a leaf node after removal of $l$.

This procedure is repeated with the next lowest label in $R$.

Once all leaves are considered, the set of newly discovered nodes, say $S$ form two partitions viz. leaves and intermediate nodes. The new set of leaves $R'$ from $S$ is determined by checking the transitive relationship between all newly discovered nodes. Since removal of a leaf node in a tree will not change the properties of the tree, the next lowest labeled node is well defined from this new set $R'$. This procedure is repeated with $R'$. The algorithm stops when there are exactly two nodes with path length 1.

---

Algorithm: Prüfer Topology Discovery

**Input:** Leaf nodes $R = (1, 2, 3, \ldots, r)$
Path Matrix $D[1..r, 1..r]$

**Output:** Prüfer sequence $P$

```
last_node = r; P = {};
repeat:
    while (R not empty)
        begin
            S = {}
            k = leaf with the lowest label from R
            if $k$ has a neighbor in $P$, say $P_i$
                then
                    begin
                        m = $P_i$
                        Append $m$ to Prüfer sequence $P$
                        Remove $k$ from path matrix
                        if there is only one $d_{ij} == 1$ in $D$, goto done:
                    end
                else
                    begin
                        increment last_node
                        m = last_node
                        Consider $m$ as the neighbor of $k$
                        Append $m$ to Prüfer sequence $P$
                        Remove $k$ from path matrix
                        Add $m$ to the path matrix
                        $d_{im} = d_{mi} = d_{ik} - 1$, $i$ belongs to {1, 2, .. m-1}
                        Append $m$ to $S$
                    end
            end
        R' = getleaves($S$, $D$)
        if $|R'| == 2$ and $d_{ij} == 1$, goto done:
        else $R = R'$; goto repeat:
    done: Print $P$
```

getleaves() is a procedure that partitions the intermediate nodes generated into set of leaf nodes and non-leaf nodes after removing the previous set of leaf nodes. It is determined by checking the transitive relationships (triangular inequality) between the generated intermediate nodes. That is, if a path between a pair of vertices $u$ and $v$ passes thru an intermediate vertex $w$, then dist($u$, $v$) = dist($u$, $w$) + dist ($w$, $v$). A node $m$ is considered as leaf node if $m$ does not appear as an intermediate node between any pair of vertices discovered so far.

Algorithm: Get Leaves

**Input:** Intermediate nodes Set $S$ and Path matrix $D$

**Output:** Set of leaf nodes

```
Mark all nodes in $S$ as leaf nodes
for $i$ ranges on $|S|$
    for $j$ ranges on $|S|
        for $k$ ranges on $|S|
            if $d_{ij} == d_{ik} + d_{kj}$ then mark node $k$ as non-leaf
    return all nodes that are marked as leaf nodes in $S$
```
The correctness of the algorithm is based on the proof of correctness of Prüfer encoding and decoding algorithms and the fact of unique tree representation for the given path matrix and Prüfer sequence.

The time complexity of this algorithm is derived as follows:

The algorithm begins with \( R \) leaf nodes. After processing the first set of \( R \) leaf nodes, \( R' \) will be at least 2 and at most \( R \). This is due to the fact that after removing all leaf nodes, say \( R \) in a tree, the remaining tree will have at most \( R \) nodes and at least 2 nodes. The complexity for inserting an internal node will be \( O(R) \), and the complexity for the procedure \( \text{getleaves} \) will be \( O(R') \). At the end of each iteration, the path length is guaranteed to reduce by 2, as at least 2 leaves are removed. Therefore, at the \( i \)-th iteration, the path length is guaranteed to reduce by \( 2 \), as at least 2 leaves are removed. Thus, the complexity for the \( i \)-th iteration is \( O(R') \).

VI. EXPERIMENTAL RESULTS

The following path matrix \( D \) was obtained for a simulated network where ICMP enabled to discover Layer-3 distance metric.

\[
D[1..8,1..8] = \\
0 2 2 4 5 6 6 4 \\
2 0 2 4 5 6 6 4 \\
2 2 0 4 5 6 6 4 \\
4 4 4 0 3 4 4 2 \\
5 5 5 3 0 3 3 3 \\
6 6 6 4 3 0 2 4 \\
6 6 6 4 3 2 0 4 \\
4 4 4 2 3 4 4 0 \\
\]

\[
3 \\
1 \\
2 \\
9 \\
13 \\
7 \\
10 \\
11 \\
12 \\
8 \\
4 \\
5 \\
6 \\
\]

\[ P = (9 \ 9 \ 10 \ 11 \ 12 \ 10 \ 13 \ 11 \ 10) \]

**Figure 3**

**Theorem 1**

The above algorithm produces a unique Prüfer sequence associated with the path matrix \( D[1 .. r, 1 .. r] \), where \( r \) is the number of leaf nodes for a tree \( T \).

**Proof:**

Let \( T = (V, E) \) be a tree, where \( V = \{1, 2, \ldots, n\} \) and \( E \) is a set of \( n-1 \) edges.

Let \( A \) be the adjacency matrix that represents the tree \( T \). The adjacency matrix of a tree \( T = (V, E) \), where \( V = \{1, 2, \ldots, n\} \), is the matrix \( A[1 .. n, 1 .. n] \) given by

\[
A[i, j] = \begin{cases} 
1 & \text{if } (i, j) \text{ belongs to } E \\
0 & \text{if } (i, j) \text{ does not belong to } E.
\end{cases}
\]

Let \( R = \{1, 2, 3, \ldots, r\} \) be the set of internal leaf nodes in our algorithm. Let \( K = \{m, m+1, m+2, \ldots, m+k-1\} \) be the set of internal nodes, where \( m = r+1 \). \( |R| = r \) and \( |K| = k \), \( r \geq 2 \), \( k \geq 1 \). This implies the total number of nodes \( n = r + k \).

Let \( P \) and \( Q \) be the two different Prüfer codes of length \( n-2 \) obtained from the path matrix \( D \) using our algorithm. \( P \) and \( Q \) are Prüfer codes of any sequence of labels taken from \( k \) labels in \( K \), with repetitions allowed and any member of \( K \) appears at least once.

Since the codes \( P \) and \( Q \) differ in at least one position and also any OD pair there is a unique path, it suffices to show that any change in the code produces a different adjacency matrix, which in turn produces a different distance matrix.

Let \( \text{last}[1 .. n-2] \) be a linear array that marks the rightmost (last) occurrences of labels in the code. The value of \( \text{last}[i] \) is 1 if \( i \) is the rightmost position in the code for the node in \( P[i] \) and it is 0 otherwise.

Let \( R' \) be the new set of leaf nodes, which has been obtained from the removal of previous set of leaf nodes. This set is created by adding \( P[i] \) to \( R' \) when \( \text{last}[i] = 1 \), for the previous string of sequence scanned in the code.

Initially, the adjacency matrix \( A[1 .. n, 1 .. n] \) is initialized to zero. The entry \( A[i, j] \) is set to 1 for the following conditions:

When \( i \) belongs to \( R \),

The first \( r \) nodes in the Prüfer code are incident to leaf nodes in \( R \).

\[ A[i, P[i]] = 1, \text{ where } i = 1, 2, \ldots, r \]

When \( i \) belongs to \( K \),

Consider \( R' \), which is the new set of leaves, such that \( 2 \leq |R'| \leq |R| \)

\[ A[i, j] = 1, \text{ where } i \text{ belongs to } R' \text{ and } j = P[i], \text{ where } P[i] \text{ is adjacent to } i \text{ and } i \text{ indexes into corresponding labels in the} \]
Proposed a new algorithm to discover the topology (tree) using Prüfer code from the distance matrix (distance between leaf nodes). In this case, the Prüfer code will be different for both codes in at least one iteration, which in turn will have different values for i and j to make A[i][j] = 1. Iteration is defined as discovering a set of leaf nodes from the last array, and associating neighbors from the code. Hence in both cases, change in any one position of the codes will have a change in the adjacency matrix, and hence change in the structure of the tree. Hence P and Q should be identical if the above procedure needs to generate the same adjacency matrix.

Hence the proof.

The above algorithm is verified by computing the distance between all pairs of vertices using Floyd Warshall algorithm on the adjacency matrix obtained from the Prüfer code.

The adjacency matrix for the Prüfer code obtained in the above example (See figure 3):

\[ P = \{9 \ 9 \ 9 \ 10 \ 11 \ 12 \ 10 \ 13 \ 11 \ 10\} \]

\[ A[1..13,1..13] = \]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Applying Floyd Warshall algorithm on the above adjacency matrix produces the following distance matrix that represents the path length between every pair of vertices:

\[ D[1..13,1..13] = \]

\[
\begin{bmatrix}
0 & 2 & 2 & 4 & 5 & 6 & 6 & 4 & 1 & 3 & 4 & 5 & 2 \\
2 & 0 & 2 & 4 & 5 & 6 & 6 & 4 & 1 & 3 & 4 & 5 & 2 \\
2 & 2 & 0 & 4 & 5 & 6 & 6 & 4 & 1 & 3 & 4 & 5 & 2 \\
4 & 4 & 4 & 0 & 3 & 4 & 4 & 2 & 3 & 1 & 2 & 3 & 2 \\
5 & 5 & 5 & 3 & 0 & 3 & 3 & 3 & 4 & 2 & 1 & 2 & 3 \\
6 & 6 & 6 & 4 & 3 & 0 & 2 & 4 & 5 & 3 & 2 & 1 & 4 \\
6 & 6 & 6 & 4 & 3 & 2 & 0 & 4 & 5 & 3 & 2 & 1 & 4 \\
4 & 4 & 4 & 2 & 3 & 4 & 4 & 0 & 3 & 1 & 2 & 3 & 2 \\
1 & 1 & 1 & 3 & 4 & 5 & 5 & 3 & 0 & 2 & 3 & 4 & 1 \\
3 & 3 & 3 & 1 & 2 & 3 & 3 & 1 & 2 & 0 & 1 & 2 & 1 \\
4 & 4 & 4 & 2 & 1 & 2 & 2 & 3 & 1 & 0 & 1 & 2 & 1 \\
5 & 5 & 5 & 3 & 2 & 1 & 1 & 3 & 4 & 2 & 1 & 0 & 3 \\
2 & 2 & 2 & 2 & 3 & 4 & 2 & 1 & 1 & 2 & 3 & 0 & 0 \\
\end{bmatrix}
\]

From the above matrix, it is seen the sub matrix \( D[1..8,1..8] \) is the actual distance matrix between leaf nodes.

VII. CONCLUSION

In this paper we have proposed a new algorithm to discover the topology (tree) using Prüfer code from the distance matrix (distance between leaf nodes). We also verified the correctness of this algorithm by deriving the adjacency matrix from the Prüfer code and computing the distance between every pair of vertices using Floyd Warshall algorithm.

It will be interesting to study the behavior of the algorithm when the underlying topology is a graph. The probing techniques to discover the path matrix in case of ICMP failures needs a separate treatment and is not covered in this paper.

REFERENCES