Distributed Partial Inference under Churn

Sriharsha Gangam
Department of Computer Science
Purdue University
Email: sgangam@cs.purdue.edu

Sonia Fahmy
Department of Computer Science
Purdue University
Email: fahmy@cs.purdue.edu

***Abstract*—Effective network measurement can significantly improve application performance. One of the main challenges in obtaining network measurements is to achieve high accuracy while consuming few network resources. To address this problem, several inference mechanisms have been proposed. These mechanisms can provide the $O(n^2)$ end-to-end measurements among n nodes using $O(n)$ measurements, with some loss in accuracy.**

We construct a *measurement request graph* where a *measurement request* issued by an application (e.g., for delay between two nodes) is represented by an edge between the nodes. When the measurement request graph is sparse, an inference mechanism operating over all the nodes (complete inference) incurs unnecessary cost. Given a measurement request graph, our goal is to optimize the number of measurements as well as improve overall accuracy by applying inference only to dense sub-graphs, while taking the other measurements directly. We call this technique *partial inference*. Previous work only considered static measurement request graphs. However, the measurement request graph can be dynamic when nodes frequently join and leave the network. This paper designs and evaluates a distributed algorithm where each node decides if it should participate in an inference mechanism based on limited information.

I. Introduction

Over the past few years, several inference mechanisms [6], [13], [14], [17] have been proposed to infer network properties (e.g., end-to-end delays). The underlying methodology in most of these mechanisms is that each node contacts a constant number of nodes on average to take measurements and exchange information. The remaining measurement values are inferred so that all pair-wise measurements are obtained. More formally, inference mechanisms provide all-pairs ($O(n^2)$, where n is the number of nodes) measurement data, by taking fewer measurements – a value proportional to the number of nodes participating in the measurements ($O(n)$).

We use k to represent *the average number of measurements taken per node by an inference mechanism*. Vivaldi [6] suggests that nodes take measurements to 32 neighbors for reasonable accuracy. Similarly, GNP [13] recommends that the nodes take measurements to 15 landmarks. Thus, an inference mechanism similar to [6], [13], [14], [17] takes nk measurements in steady state, where n is the number of nodes and k is an inference mechanism parameter.

Measurement services such as [10]–[12], [15], [19] can be deployed to provide estimates of network characteristics to applications. When a measurement service receives a request for measurement data between two nodes, we refer to this as a *measurement request*. These measurement requests can be modeled via a *measurement request graph*, where each measurement request is represented by an edge between the nodes. A measurement service accepts measurement requests from applications for various measurement data between different nodes in the network. To handle these requests, a measurement service may utilize inference mechanisms to reduce measurement traffic. Using an inference mechanism to satisfy every measurement request (*i.e., complete inference*) may, however, result in taking unnecessary measurements. This is especially the case when either (a) the measurement request graph is sparse, or (b) the constant k of the inference mechanism is large. In these cases, an inference mechanism will take more measurements than what is actually required. For example, a node may be interested in taking measurement to only one other node; in that case, it is beneficial to take a *direct measurement* rather than include it in an inference mechanism. Further, accuracy of inference mechanisms is lower than direct measurements.

In this work, we identify nodes that should participate in an inference mechanism (each taking k measurements on average) in order to reduce the total number of measurements taken. We assume that the value of k is an input parameter to our problem, and that it is a constant specific to an inference mechanism. The term *measurement* denotes the measurement of any network characteristic (e.g., path delay or loss). The measurement can be taken directly, or inferred by an inference mechanism that uses kn overall measurements and supplies all-pairs measurement data. We use the term *node* or *vertex* to represent a machine involved in taking measurements.

In our previous work [3], we proposed and evaluated an algorithm for detecting "dense" clusters in a measurement request graph. The main idea was to execute a minimum spanning tree algorithm multiple times (with random edge weights) and remove edges which are deemed "critical." The remaining graph represents dense clusters and employs inference in order to reduce the total number of measurements. Our algorithm was centralized and operated on static measurement request graphs. In this paper, we design a distributed algorithm that performs well in the presence of churn.

In section II, we define the problem. In sections III and IV, we give a simple algorithm called $k-Core$ to solve the problem, and give a theoretical bound on its performance. In section V, we develop a distributed version of the $k-Core$ algorithm, and show how it handles node churn. In section VI, we evaluate the performance of this distributed algorithm.

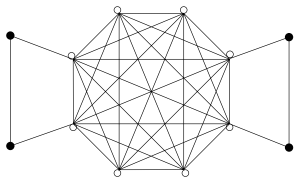

Fig. 1. Example showing the optimal solution for a measurement request graph. The inference parameter $k = 3$. The white vertices are *inference-vertices*, and the black vertices are *direct-vertices*.

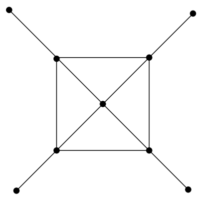

Fig. 2. An optimal solution identifies all vertices to be direct-vertices. The inference parameter $k = 3$.

Section VII concludes the paper.

II. Problem statement

Given a *measurement request graph* $G(V, E)$ (where V is a set of vertices representing nodes involved in measurement requests, and E is a set of edges representing measurement requests) and the inference mechanism parameter k, our problem is to find a sub-graph $G_i(V_i, E_i)$ such that when connected components in G_i participate in inference, and nodes in $G-G_i$ take direct measurements, the total number of measurements, M, is minimized. Thus, M is given by

$$M = k|V_i| + (|E| - |E_i|), and$$

$$E_i = \{\overline{vw}/v \in V_i, w \in V_i\}$$

The first term in the equation, $k|V_i|$, corresponds to measurements taken by an inference mechanism, and the second term, $|E| - |E_i|$, corresponds to direct measurements. If a vertex v participates in inference ($v \in V_i$), we call it an *inference-vertex*; otherwise we call it a *direct-vertex* ($v \in V - V_i$). Note that the sub-graph G_i may be disconnected, even if the input graph G is connected. Note also that both vertices v and w of edge $\overline{vw}$ have to be inference vertices for the edge to be in the term E_i.

In the context of a measurement service, using an inference mechanism on the entire measurement request graph (*i.e., complete inference*) is not always efficient. It is often beneficial to use inference on a subset of the graph (*i.e., partial inference*) to reduce the total number of measurements taken and increase accuracy. Consider the graph in Fig. 1 for $k = 3$. A vertex in this measurement request graph corresponds to an end node (host) in the network. An edge in the measurement request graph represents a measurement request between the vertices, e.g., computing the loss rate on the path between the two hosts (the hosts may, of course, be connected via multiple physical links in the underlying network). Using inference on the entire graph will require $k \times |V| = 3 \times 12 = 36$ measurements. Using direct measurements on the graph will require $|E| = 34$ measurements, since there are 34 edges in the graph. The optimal solution identifies the vertices colored white as inference-vertices. This partial inference configuration will require a total of $k|V_i| + (|E| - |E_i|) = 3 \times 8 + 6 = 30$ measurements to be taken. Additionally, the 6 direct measurements may benefit from higher accuracy.

III. The k-Core algorithm

The $k-Core$ algorithm gives an approximate solution to the problem defined in the previous section, which is NP-hard [3]. We define the $k-Core$ of graph G as the graph resulting from repeated deletion of vertices (and their incident edges) whose degree is $\leq k$. This process can be viewed as a decomposition technique to find densely connected vertices in a graph. The resulting sub-graph (with all vertices of degree greater than k) is identified as a good candidate to utilize an inference mechanism. The deleted edges represent direct measurements.

The pseudo-code of this method is given in Algorithm 1. An optimized $O(|E|)$ $k - Core$ algorithm is given in [2], where $|E|$ is the number of edges. The $k - Core$ algorithm is intuitive. Suppose a vertex v has degree $\leq k$. The vertex v will take $\leq k$ measurements if it is a direct-vertex. However, it can take k measurements if it is an inference-vertex.

Algorithm 1 k-Core ($G(V, E)$,k)

```
repeat
  $|V\_prev| \leftarrow |V|$
  for each $v \in V$ do
    if $degree(v) <= k$ then
      $V \leftarrow V \setminus \{v\}$
      for each edge $\overline{vw}$ incident on $v$ do
        $E \leftarrow E \setminus \{\overline{vw}\}$
      end for
    end if
  end for
until $|V\_prev| = |V|$
```

IV. Bounds on performance of the k-Core algorithm

The $k - Core$ algorithm gives an optimal solution in many cases. Consider Fig. 2 with $k = 3$. The algorithm identifies all vertices as direct-vertices, and the number of measurements is 12. The $k-Core$ algorithm also gives the optimal solution for Fig. 1. We conducted several simulations with the $k - Core$ algorithm, and found that it gives close to optimal solutions on graphs representing peer connections in the UUSee peer-to-peer video streaming service [4].

The $k-Core$ algorithm does not, however, always give the optimal solution. The algorithm fails for the example graph shown in Fig. 3 for $k = 3$. Since there is no vertex with degree ≤ 3, all vertices will be designated as inference-vertices, and

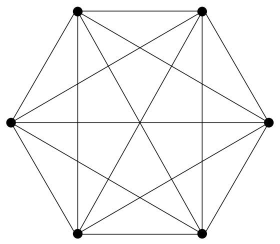

Fig. 3. An example graph where the $k - Core$ algorithm fails to give an optimal solution for $k = 3$.

the number of measurements in this case will be $6 \times 3 = 18$. The optimal solution identifies each vertex as a direct-vertex, and the optimal number measurements is 15 ($|E|$). The following results formalize some properties of the $k - Core$ algorithm. We omit the proofs for brevity; they can be found in [7].

Lemma 1. *If the degree of a vertex in a measurement request graph is $\leq k$, then an optimal solution to the problem chooses that vertex to be a participant in direct measurements.*

Corollary 1. *Any vertex chosen by the $k - Core$ algorithm to be a participant in direct measurements also belongs to the set of direct-vertices chosen by an optimal solution.*

Theorem 1. *Let M_o be the optimal total number of measurements value for a given measurement request graph $G(V, E)$ and inference parameter k. Let M_c be the number of measurements value using the $k - Core$ algorithm. Then*

$$M_o \leq M_c < 2M_o$$

V. Handling Churn

If the $k - Core$ algorithm is centralized, it must be cognizant of all the measurement requests. This can be known by sending control messages for each measurement request to a centralized location that executes the algorithm. Each measurement node is classified to be an inference-vertex or direct-vertex and the centralized location then informs each measurement node of its classification. This approach is not scalable and creates a single point of failure. Additionally, in the presence of churn, the algorithm may be using stale and inconsistent information, since the measurement request graph changes over time.

Due to these reasons, we need to develop a scalable distributed protocol to apply the $k - Core$ concept at each measurement node, taking into account that the set of nodes in the graph can dynamically change. Each pair of measurement nodes corresponding to measurement request must therefore exchange their state information.

A. The distributed k-Core algorithm

We design an algorithm that is executed at each node to decide if it should participate in an inference mechanism or not. The intuition behind this "Distributed $k - Core$ Algorithm" is similar to that of the $k - Core$ algorithm. The decision taken by each node whether to be an inference-vertex or a direct-vertex is based on how many measurements it would take. A node decides to join an inference mechanism based on the degree and state of the neighbors of that node. For example, if the number of neighbors of a vertex v that are participating in inference exceeds k, the algorithm will identify the vertex v to be a participant in inference.

Consider the case when all nodes are initially inference-vertices. Without churn, the distributed $k - Core$ algorithm is similar to the centralized version: Changing state to direct measurement corresponds to removing a vertex with degree $\leq k$ in the $k - Core$ algorithm. The set of neighbors which are direct-vertices corresponds to the vertices which have been removed from the graph in the $k - Core$ algorithm.

Our distributed $k - Core$ algorithm is given in Algorithm 2. Each node maintains a *state*: *Inference* or *Direct*. The state indicates if the vertex is an inference-vertex or a direct-vertex. The variables $N_{n,i}$ and $N_{n,d}$ (initialized to zero) represent the number of neighbors of a vertex n that are participating in inference and direct measurements respectively. The variable $D_{n,2k}$ (also initialized to zero) represents the number of neighbors of node n with $degree > 2k$.

Algorithm 2 Distributed k-Core

```
for each neighbor $nb$ of $n$ do
    Get $State(nb)$ and $Degree(nb)$
    if $Degree(nb) > 2k$ then
        $D_{n,2k} \leftarrow D_{n,2k} + 1$
    end if
    if $State(nb) = Inference$ then
        $N_{n,i} \leftarrow N_{n,i} + 1$
    else if $State(nb) = Direct$ then
        $N_{n,d} \leftarrow N_{n,d} + 1$
    end if
end for
if $N_{n,i} > k$ or ( $D_{n,2k} > 2k$ and $N_{n,i} + N_{n,d} > 2k$ ) then
    $State(n) \leftarrow Inference$
else
    $State(n) \leftarrow Direct$
end if
```

Compared to the $k - Core$ algorithm, an additional test: ($D_{n,2k} > 2k$ and $N_{n,i} + N_{n,d} > 2k$) is used so that the algorithm is stable in the presence of churn. Suppose that the graph has all direct-vertices and many vertices join the graph. Without this test, the higher degree direct-vertices would never be able to change their state to *Inference*.

B. Message passing

To execute the distributed $k - Core$ algorithm, a node has to detect changes in the degrees and states of its neighbors in response to churn. The nodes need to exchange messages containing degree and node state information. We consider two basic ways to exchange these messages:

1) Messages are sent each time a node receives a signal indicating a change in topology or state of its neighbors.

2) Messages are sent periodically irrespective of any topology or state change of its neighbors.

In the first case, a signal is sent by a newly joining node or a departing node to its neighbors indicating a change in the measurement request graph. Upon receiving this signal, a node will exchange messages with its neighbors and execute Algorithm 2. If there is a change in state from *Inference* to *Direct* or vice versa, a node must notify its neighbors via a signal. This message passing technique responds well to churn and is effective for determining an accurate and up-to-date set of inference vertices. However, if the arrival and departure rates are high, there will be an explosion in the number of signals.

In the second message passing technique, each node communicates periodically with its neighbors and executes Algorithm 2. We refer to the periodicity of message exchange as the *time step*. The *time step* is also a measure of how frequently the algorithm runs at each node. This strategy can be effective if the arrival and departure rates are high, but the *time step* should accurately reflect churn. It should not be too large (stale topology information is used) or too small (results in unnecessary computation and network overhead).

The network cost associated with signaling or messaging neighbors is critical. We note that the information that is being sent in each message or signal is on the order of bits. These messages or signals can be easily piggybacked onto other traffic in the network. Additionally, once the inference vertices have been determined and are known to be stable for some time, inference can be further optimized.

VI. Evaluation

We evaluate the distributed $k-Core$ algorithm using both message passing techniques (signaling and time step). The problem of identifying vertices to perform inference depends on the inference parameter k, the measurement request graph, and the churn. For the measurement request graph, we use graphs representing peer connections of the UUSee live video streaming service. For the churn, we use synthetic churn as well as churn data from the Skype network. We experiment with a wide range of values (3 to 60) of the inference parameter k.

A. Simulation parameters

1) Input measurement request graphs: We use UUSee graphs [3] as measurement request graphs. These graphs were generated to match the node degree distribution and clustering properties of the UUSee service, as published in [18]. They represent the peer connections among nodes in the UUSee live streaming video service, and they have about 2500 vertices and 53000 edges. A detailed discussion of how the graphs are generated can be found in [3].

2) Synthetic churn model: We use a synthetic churn model where node arrivals follow a Poisson process. The node staying times in the network follow a Pareto distribution, with minimum staying time 90 s and Pareto shape parameter 1.42. These parameters are consistent with distributions found in live streaming applications [5], [16]. In our simulations, the inter-arrival time for a node is exponentially distributed with a mean of 90 s unless otherwise stated. This value results in about 70% of the live nodes being at equilibrium.

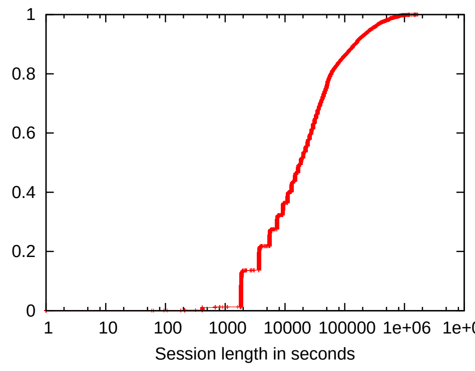

Fig. 4. Session time CDF of nodes in the Skype network.

3) Experimental churn: We use an experimental churn data set from the Skype network [8], [9]. In this data set, arrival and departure times are provided for 2081 nodes, for a period of 28.7 days. There are about 690 nodes up (on average) at a given time. The *cdf* of session times of Skype peers is shown in Fig. 4. Note that most of the nodes have a session length of more than 1000 seconds.

4) Delays among nodes: To observe the convergence effects of the algorithm, we simulate delay between the nodes so that the time taken for message passing is realistic. We use a subset of the MIT king latency dataset [1] which includes latency between sets of DNS servers. This subset has about 53000 RTT values with a mean of 133 ms.

B. Performance of the distributed k-Core algorithm

The objective of the distributed $k-Core$ algorithm is to dynamically identify sub-graphs in a measurement request graph suitable for an inference mechanism. Fig. 5 compares the performance of our algorithm with that of complete inference (when all vertices participate in inference) and direct measurements (without any inference mechanism) over time. The synthetic churn model was used in these simulations. The results were consistent with different sets of UUSee graphs and different churn parameters. The inference parameter k was chosen to be 15 for this simulation. For smaller values of k, the number of measurements taken in partial inference and complete inference will be closer because more vertices tend to participate in inference. Our algorithm gives close to optimal classification of inference vertices because it leverages the same basic intuition as the $k-Core$ algorithm (which gives nearly optimal results without churn [4]).

The larger the value of the inference parameter k, the more expensive the inference mechanism is (in terms of number of measurements). Fig. 6 compares the percentage of inference vertices over time and for different values of k. As expected, for larger values of k, the percentage of inference vertices decreases. A larger value of k implies the need for a “denser”

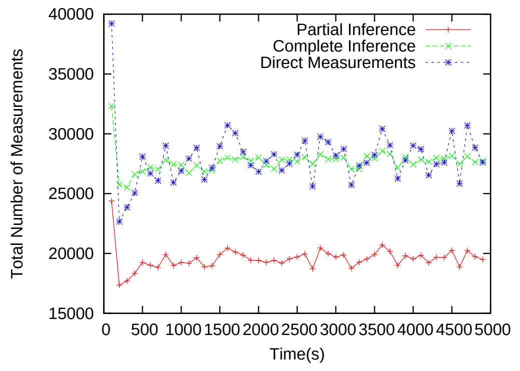

Fig. 5. Partial inference cannot take more measurements than complete inference or direct measurements ($k = 15$).

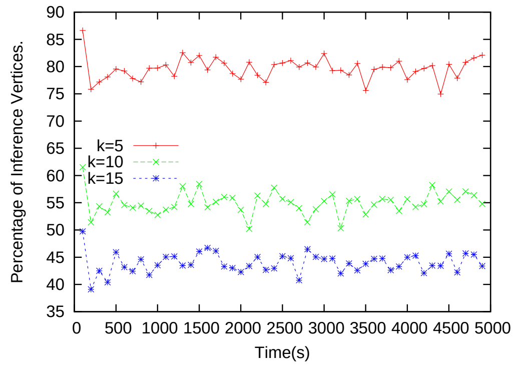

Fig. 6. The percentage of nodes participating in inference for different values of k.

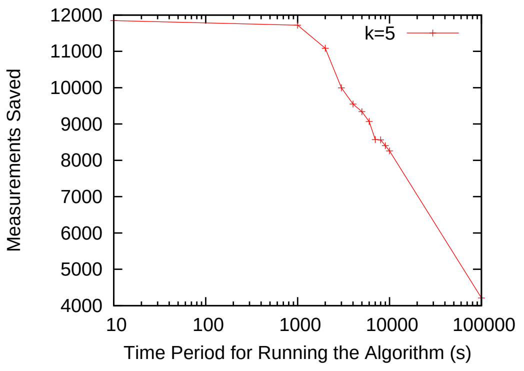

Fig. 7. *Measurements saved* for different time step values simulated with experimental churn of the Skype network.

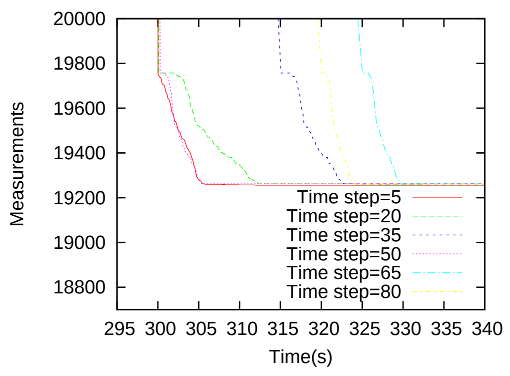

Fig. 8. Transient behavior for various algorithm time step values. At the 300 s time instance, one fourth of the vertices leave the network.

(in terms of degree) subset of nodes in order to participate in inference.

C. Time step choice

In the time step message passing technique, a node sends messages to its neighbors periodically. The overhead of the algorithm depends on the number of messages exchanged by the nodes, which is directly related to the time step choice. We need to identify the time step value so that as few messages as reasonable are exchanged without using stale information.

Our algorithm tries to "save" measurements by switching states among inference or direct whenever appropriate. Let ΔM denote the change in the number of measurements due to this state change. We define *measurements saved* as the sum of all the ΔM values over all nodes for a sufficiently long time period. This value serves as a performance metric of the algorithm. A higher value of *measurements saved* implies that the algorithm has responded well to churn.

1) Time step for the synthetic churn data set: For several arrival and departure parameters of the synthetic churn model, we find that, as expected, the *measurements saved* decreases with increasing time step values. However, we observed that for low churn (larger inter-arrival times and smaller Pareto shape parameter), the time step value does not significantly affect the performance. Further, for larger values of k (from 40-45 onwards), we observed that the performance is best when the time step value is close to the minimum staying time of the nodes.

2) Time step for the Skype churn data set: In the Skype churn data set, there are fewer live nodes at each point in time and this makes the measurement request graph less dense. We use the inference parameter $k = 5$ for this experiment so that there are sufficient vertices participating in inference and in direct measurements. Note that the time step value of 1000 seconds is optimal for this churn model, as can be seen in Fig. 7. We believe that this is because most of the nodes have a session time greater than 1000 seconds as shown in Fig. 4.

D. Convergence of the algorithm

1) Transient performance: Fig. 8 shows the transient behavior of the algorithm for $k = 15$. At time 300 seconds, one fourth of the nodes leave the network and the total number of measurements drops sharply. The plot shows the time taken by the algorithm to converge for different time step values. Convergence is attained when the number of measurements becomes constant. As expected, it takes a longer time to converge if the time step is larger. Note that the convergence times are less than the time step values. We also evaluate the signaling based message passing technique when one fourth of the nodes instantly leave the network. We observe that convergence time varies from 2 to 3 s, which is a reasonable value for the size of the graphs we use (with 2500 vertices and 53000 edges).

2) Startup convergence: We now evaluate the convergence of the algorithm when all nodes are initially inference-vertices,

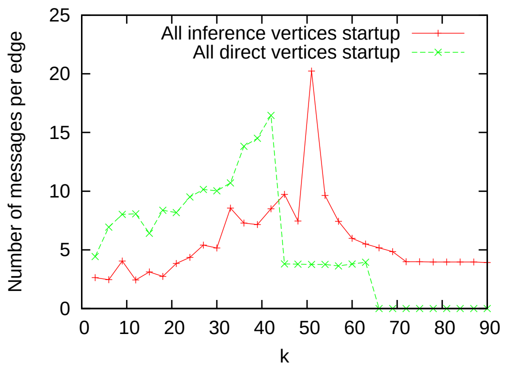

Fig. 9. The number of messages per edge for different values of the inference parameter k.

or all the nodes are initially direct-vertices (i.e., the *startup convergence*).

When the signaling based message passing technique is used, we observe that the time to converge lies between 2 to 3 seconds for most values of k. For the time step based message passing technique, we plot the average number of messages per edge required for the algorithm to converge in Fig. 9. A time step of 5 s was used in the simulation. We compare the performance with different values of k. When the value of k is small relative to the average degree, most of the vertices will have a $degree > k$ and converge to inference-vertices after a few messages. Similarly, for larger values of k, most of the vertices have a $degree \leq k$ and quickly become direct-vertices. In between these two extremes, there is a larger set of nodes which require a higher number of messages to converge. This results in the network overhead graph shown in Fig. 9.

VII. Conclusions

In this work, we tackle the problem of identifying nodes suitable for participating in inference in the context of a measurement service. We design a simple algorithm, $k-Core$, to address the problem. We give bounds on the number of measurements with this algorithm. We then develop a distributed version of the algorithm by maintaining and exchanging node and neighbor information at each node. We develop a simulator which implements signaling as well as time step based message passing techniques for the algorithm. We evaluate the performance of both versions of the algorithm with synthetic and experimental churn models. We also vary the inference parameter k. We find that the algorithm rapidly converges in our experiments. The minimum staying time or session time of the nodes appears to give a good estimate for setting the algorithm time step. In our future work, we plan to integrate and deploy this algorithm in a measurement service.

Acknowledgments

This research has been sponsored in part by Hewlett-Packard, and by BBN Technologies (GENI project 1723). We would like to thank Ethan Blanton for many insightful discussions on earlier versions of this work, and for providing the graphs used for evaluation.

References

[1] MIT King data set. http://pdos.csail.mit.edu/p2psim/kingdata/.

[2] Vladimir Batagelj and Matjaž Zaveršnik. An $o(m)$ algorithm for cores decomposition of networks. In *Proceedings of Recent Trends in Graph Theory, Algebraic Combinatorics, and Graph Algorithms*, September 2001.

[3] Ethan Blanton, Sonia Fahmy, and Greg N. Frederickson. On the utility of inference mechanisms. In *ICDCS '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, pages 256–263, 2009.

[4] Ethan Blanton, Sonia Fahmy, Greg N. Frederickson, and Sriharsha Gangam. On the cost of network inference mechanisms. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2010. Accepted for publication.

[5] Yang-hua Chu, Aditya Ganjam, T. S. Eugene Ng, Sanjay G. Rao, Kunwadee Sripanidkulchai, Jibin Zhan, and Hui Zhang. Early experience with an internet broadcast system based on overlay multicast. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 12–12, 2004.

[6] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2004.

[7] Sriharsha Gangam and Sonia Fahmy. Performance bounds on the k-core algorithm. Technical report, Purdue University, 2009. http://www.cs.purdue.edu/homes/sgangam/files/pb.pdf.

[8] P. Brighten Godfrey. Available traces. http://www.cs.illinois.edu/∼pbg/availability/.

[9] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, Santa Barbara, CA, February 2006.

[10] Harsha Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: an information plane for distributed services. In *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 26–26, 2006.

[11] Harsha V. Madhyastha, Ethan Katz-Bassett, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane nano: path prediction for peer-to-peer applications. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 137–152, 2009.

[12] Akihiro Nakao, Larry Peterson, and Andy Bavier. A routing underlay for overlay networks. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 11–18, 2003.

[13] T. S. Eugene Ng and Hui Zhang. Towards global network positioning. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 25–29, 2001.

[14] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Workshop on Peer-to-Peer Systems*, 2003.

[15] Neil Spring, David Wetherall, and Tom Anderson. ScriptRoute: A public internet measurement facility. In *Proc. of USITS*, 2002.

[16] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. An analysis of live streaming workloads on the internet. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 41–54, 2004.

[17] W. Theilmann and K. Rothermel. Dynamic distance maps of the internet. In *Proceedings of INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 275–284 vol.1, 2000.

[18] Chuan Wu, Baochun Li, and Shuqiao Zhao. Magellan: Charting large-scale peer-to-peer live streaming topologies. In *Proc. of 27th International Conference on Distributed Computing Systems (ICDCS '07)*, pages 62–62, June 2007.

[19] Praveen Yalagandula, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Sung-Ju Lee. S3: a scalable sensing service for monitoring large networked systems. In *INM '06: Proceedings of the 2006 SIGCOMM workshop on Internet network management*, pages 71–76, 2006.