BridgeFinder: Finding communication bottlenecks in distributed environments

Dirk Bradler
TU Darmstadt, Germany
bradler@cs.tu-darmstadt.de

Lachezar Krumov
TU Darmstadt, Germany
krumov@cs.tu-darmstadt.de

Max Mühlhäuser
TU Darmstadt, Germany
max@cs.tu-darmstadt.de

Jussi Kangasharju
University of Helsinki, Finland
jussi.kangasharju@cs.helsinki.fi

***Abstract*— Nodes in mobile networks are usually unevenly distributed over space. Several dense clusters of nodes are interconnected by a few nodes in sparsely occupied areas. Removing vital nodes along such *bridges* would partition the network and severely reduce the overall connectivity. Consequently, detecting and protecting those few vital nodes is crucial for keeping the network operational. In order to achieve this task, we present our novel approach: BridgeFinder. Most importantly, BridgeFinder allows us to calculate good estimates for global graph measures, while operating as a fully distributed algorithm and causing only very little messaging overhead. It is based on an extended gossiping protocol and is significantly faster and more precise than existing mechanisms. The results of our extensive evaluation show that BridgeFinder is indeed very effective in detecting the few crucial for the network operation nodes.**

***Index Terms*—Robustness, Distributed Control, Computer Network Reliability, Mobile Communication**

I. Introduction

Our main target environments are distributed wireless environments, such as wireless multihop networks. They impose severe challenges upon any distributed application. Due to many unpredictable factors, established connections or even nodes may suddenly disappear. Because communication flow depends on the network connectivity, it is crucial to identify and protect the few critical peers within the network. Those are the articulation points, which failure leads to malfunction or complete breakdown of the network, as they constitute the few *bridges* keeping the network together.

Standard approaches for identifying such vital links require global network knowledge, which is unavailable in mobile multihop networks. Even if it were, one would require $O(N^3)$ running time, where N is the number of nodes within the network (by using Floyd-Warshall for example). Obviously, such approaches are of extremely limited practical use.

In order to overcome these challenges, we developed BridgeFinder. It is compliant with the Push-Sum gossiping algorithm [1]. BridgeFinder identifies critical paths between densely connected clusters based only on local knowledge. Our extensive evaluation shows that BridgeFinder very efficiently identifies the critical nodes. Furthermore, it can be integrated in the regular maintenance and application traffic and therefore produces almost no additional messaging overhead.

There are two requirements the underlying network has to fulfill in order for BridgeFinder to function. First, a node must be able to communicate with other nodes in the network, at least with its direct neighbors. Second, the links in the network must be undirected, i.e., communications must be bidirectional. In general, BridgeFinder operates on any network that fulfills these two criteria. Both criteria are typically fulfilled by mobile multihop networks, at least to the extent required by BridgeFinder.

BridgeFinder detects critical peers, crucial for communication within the network. Therefore, the fewer the nodes on which communication depends, the higher the benefit of using our approach. Compared to algorithms based on global knowledge in static networks, BridgeFinder is less accurate than standard approaches for detecting critical nodes. *However*, its most significant advantage is that it requires no global knowledge and can operate on continuously changing underlying networks. That makes it very attractive for distributed environments, like wireless ad-hoc networks, P2P overlay networks and wireless sensor networks.

Once the critical peers are known, partitioning can be avoided by establishing additional links among susceptible clusters connected by those critical peers. In any complex network, maintaining information flow is equal to the critical nodes remaining operational. The higher the importance of a node, the higher is the impact of losing it and reversely the higher the benefit of detecting and protecting that node a priori.

The rest of this work is divided as follows: Section II elicits the properties of critical peers within distributed systems and how they are classified using centrality measures known from graph theory. The BridgeFinder algorithm is presented in Section III and is evaluated in Section IV. Its convergence speed is investigated in Section V. Related work is discussed in Section VI and Section VII concludes the paper.

We have analyzed the runtime characteristics of the proposed approach in a varietey of attack scenarios and developed a guarding mechanism protecting BridgeFinder from malicious nodes. The results are available as supplementary material [2], which also presents a generic IPv6 implementation of BridgeFinder with optional headers, showing its broad field of application.

II. Properties of Critical Peers

There are two categories of critical peers: global and local [3]. If a globally critical peer fails, the network is partitioned into two or more components. If a locally critical

peer fails, this peer along with its neighbors are isolated from the network. The rest of the network remains operational.

There is variety of specialized algorithms for detecting the second type of peers [3], [4], [5]. Depending on the accuracy of their results, they produce different computational overhead. Still, all of them require only the neighbor list of each peer and no further knowledge about the network.

Globally critical peers are much more important. Their failure affects the function of the whole network. There are no local and hence efficient algorithms for detecting such peers.

The crucial observation on which our approach relies is that critical peers play a significant role in distributing information within the underlying network. Our results show that with very high probability a critical peer either lies on many communication paths among other nodes or has very short communication paths to almost all other nodes, and often even both. Detecting peers with such properties is equivalent to detecting globally critical peers.

A straightforward question arises: How do we describe and detect peers playing such an important role in supporting communication flow? The answer is to use centrality measures from graph theory. Note that these measures require global network knowledge and are very expensive to compute. Overcoming these two issues in distributed manner is BridgeFinder's main contribution.

The first important measure is *betweenness centrality* or simply betweenness [6], [7]. The betweenness of a node is proportional to the number of shortest paths going through this node. Nodes that occur on many shortest paths within the network have higher betweenness. Consider a graph $G = (V, E)$, where V is the set of nodes and E the set of edges connecting them. The betweenness $C_B(v)$ of a node $v \in V$ is given by:

$$C_B(v) := \sum_{s \neq v \neq t, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{1}$$

where s, $t \in V$ and $\sigma_{st}(v)$ is the number of shortest paths from s to t going through v and σ_{st} the number of all shortest paths between s and t. The higher the betweenness of a given node, the more important this node is for communication within the network.

Note that if a node v is even one single edge away from a node with high betweenness, its betweenness still could be zero. That is because no shortest path goes through v as this automatically increases the path length by one. Still, that node plays an important role for the communication flow. It can be used as a backup in case that the direct connection between the two high betweenness nodes vanishes. To overcome that problem we introduce the following measure:

Definition: The *average betweenness* of a node v is equal to the average betweenness of its own and the betweenness of its neighbors:

$$C_{AB}(v) := \frac{C_B(v) + \sum_{w \in N} C_B(w)}{|N| + 1} \tag{2}$$

where N is the set of neighbors of v and $|N|$ its size.

In other words, not only nodes with high betweenness but also their neighbors have high average betweenness. This more accurately reflects their central position in the network for two reasons. First, these are the nodes which must overtake the information flow if the leader node fails. Second, they can serve as a backup for the leader node before it fails by interconnecting among each other and thus reducing its load. That also significantly reduces the impact of suddenly losing the leader node due to some unexpected reasons.

Another important measure is *closeness centrality* [8], [9], here referred to just as closeness. Using closeness we can identify those nodes within a network which are responsible for fast communication flow. The closeness of a node v is defined as the mean geodesic distance (i.e. the shortest path) between a node v and all other nodes reachable from it:

$$C_c(v) := \frac{1}{n-1} \sum_{t \in M \setminus v} d_G(v, t) \tag{3}$$

where $d_G(v, t)$ is the geodesic distance between v and t in G and n is the size of the connected component M reachable from v. Closeness is a measure of how long it will take for a particular information to spread from a given node to all other reachable nodes.

We also have extended the definition of closeness to allow for a better identification of critical peers. Nodes with small closeness are fast in distributing information through the network. Still, closeness is a linear measure. In large networks there are whole regions of nodes with very similar closeness values. In order to sharpen the precision of closeness, we define *square closeness*. It also computes all shortest distances among all nodes in the network, but squares them before the average is computed:

Definition: The *square closeness* of a node v is the sum of the square distances from v to all other reachable nodes:

$$C_{sqc}(v) := \frac{1}{n-1} \sum_{t \in M \setminus v} d_G(v, t)^2 \tag{4}$$

where d_G, M and n are defined as in (3).

Nodes with best closeness have significantly higher squared closeness than the rest of the nodes. That makes it easier to detect outstanding nodes within clusters of nodes with relatively similar closeness.

Although all these metrics are useful in identifying the globally critical nodes, they are: (i) very expensive to compute; (ii) not applicable without global network knowledge, like in distributed environments.

To overcome that problem is the main contribution of BridgeFinder. Our results show that the nodes BridgeFinder identifies agree to large extent with the nodes one identifies by applying the analytical measures discussed above.

III. The BridgeFinder Algorithm

Gossiping works in the following manner: at an arbitrary position in the network a node starts a *gossip*. It sends the information of interest to all its neighbors. Then each node on its turn shares the information it has just received with its neighbors and so on. Eventually, all nodes become aware of the new *gossip*.

To detect global critical peers, BridgeFinder does extended gossiping. Instead just to spread a piece of information among the nodes, it rather lets a given value *diffuse* through the network. Some node A picks a random floating number d and divides it uniformly among its neighbors and itself. If A has N neighbors, then after the first iteration, all of them as well as A have value of $\frac{d}{N+1}$. In the next iteration, each node that just has received a value (including A) distributes its value among its neighbors as well and so on. Each node also keeps track of the number of iterations it has exchanged values.

To estimate when d has successfully diffused through the network, we introduce a tolerance parameter ϵ. Overall in our simulations we set $\epsilon = 0.02$. When the current exchanged value of a node v does not differ from its previous value with more than 2%, it sets its status to converged. The whole network has converged, when all of its nodes have converged.

The following is a pseudocode illustration of the exchange operation for a node v which just has received a value d_{in} from some of its neighbors:

```
Input: v, d_in, Neighbors(v) ← {w : (v,w) ∈ G}
if hasConverged(v) is false then
    M ← |Neighbors(v)|;
    d_old ← getValue(v);
    d_new ← (d_old + d_in)/(M+1);
    if |d_old − d_new| > ε then
        setValue(v, d_new);
        iterCount(v) ← iterCount(v) + 1;
        foreach w ∈ Neighbors(v) do
            sendValue(w, d_new);
        end
    else
        markConverged(v);
    end
end
```

Algorithm 1: Exchange Operation

The workflow of the the central value exchange operation can be described as follows: If the node v still has not converged, it checks if its old value d_{old} is more than an ϵ away from its new value d_{new}. In the negative case, v marks itself as converged. Otherwise, it posts its new value to all its neighbors. The node v repeats exchanging values with its neighbors until it has converged. The algorithm stops when all nodes have converged.

Each node also keeps a list of the k fastest converging nodes, in our simulations $k = 10$, and the number of exchange steps they required to converge. When a node v has converged, it checks if it has not been faster than some of the nodes in its top list. If this is the case, v sends to its neighbors the information that it has converged and that it qualifies for the top k nodes list. The neighbors of v then adapt their lists of top nodes respectively, inform their neighbors and so on.

After each run of the BridgeFinder algorithm, every node knows the fastest converging nodes. The results of the previous run are distributed in parallel to the value exchange operation of the next run and so on. As we show in Section IV, the fastest converging nodes are also the critical nodes. Once they have been identified, measures can be taken to protect them.

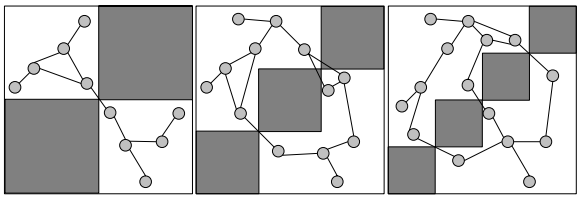

Fig. 1. Three different obstacle scenarios. Examples show 1, 2, and 3 bridges, from left to right.

IV. Evaluation

Wireless multihop networks are usually modeled with unit disk graphs (UDG) [10]. The construction principle is as follows: given N nodes and a distance threshold θ, distribute all N nodes uniformly at random within an area of a fixed size. Then, all nodes among which the geometric distance is less than θ are connected by edges. In other words, all nodes which lay close enough to each other are considered neighbors in the resulting wireless ad-hoc network. This model is valid for wireless networks within a perfect surrounding environment. In practice the entities usually move in groups, have favorite spots and there are areas that they avoid.

We extend the UDG model by inserting obstacles of different sizes within the area where we place the nodes. These obstacles represent unpopular or unaccessible areas. The generated graphs represent real world ad-hoc wireless networks more precisely. The approach for more realistic network models is based on the work of Jardosh [11]. We call the network model ODG (obstacle disk graph), see Figure 1.

We evaluate BridgeFinder using the ODG model for four different network types with none (equivalent to the standard UDG model), one, two and three *bridges*.

Note that nodes are distributed uniformly at random within the areas around the obstacles. Therefore, there is no guarantee how many of the existing bridges are actually used to connect the different clusters, nor how many different paths run through each bridge.

However, networks generated in this manner are very vulnerable. The few nodes lying on the paths among clusters are exactly the few nodes keeping the network together.

There are two important questions that need to be addressed. What topological properties do these few nodes have? More

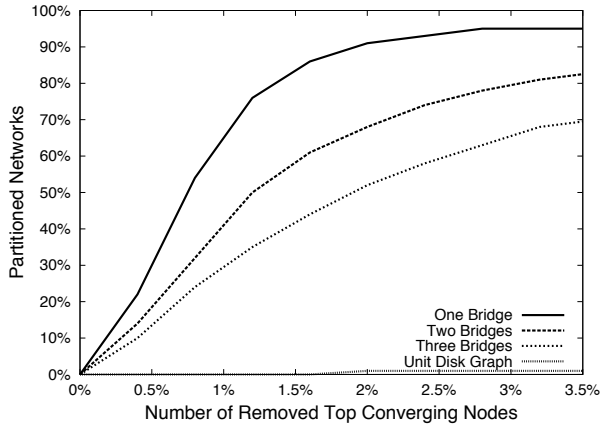

Fig. 2. Destroying networks by removing the fastest converging nodes.

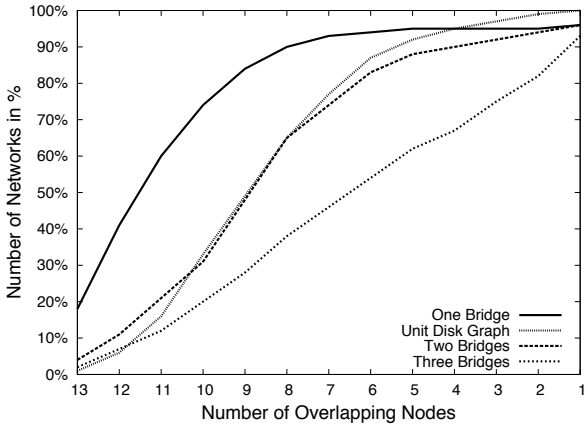

Fig. 3. Intersecting the nodes with best centrality measures with 5% of the fastest converging nodes.

importantly, how effective is indeed BridgeFinder in detecting those very nodes?

We evaluate our approach on 500 networks of each type. Each network consists of 250 nodes placed in a physical area of 100x100 units. The maximum edge length is set to 12 units, i.e., we place edges among all nodes within distance of 12 units from each other. We create multiple obstacles with dimensions of 25x25, 30x30 and 50x50 units (see Figure 1 for an example setup). Because of the obstacles and the random placement of nodes, the resulting networks are not always connected. Therefore, during the generating process we discard non-connected networks and generate new ones until 500 connected instances of each type were acquired.

Now that we have diverse test scenarios, we finally can address the central question of this work: what role do the nodes identified by BridgeFinder play for keeping the network connected. For this purpose, one by one we remove the fastest 3.5% converging nodes. The results are displayed in Figure 2. The x-axis shows the percentage of removed nodes. The y-axis shows the fraction of partitioned networks from the 500 instances of each type. We consider a network partitioned if more than 25% of the remaining nodes are not able to communicate with the rest of the network.

Figure 2 shows averaged results for all four different network types. Removing only 2% (equivalent to just 5) of the fastest converging nodes breaks 90%, 70% and 50% of the one, two and three bridge networks respectively. BridgeFinder finds with very high probability exactly the few nodes that keep the networks together.

Note that the standard UDG graph does not partition. This is not surprising. In such graphs, there are no critical nodes at all. Depending on the density of the nodes, there are two possibilities. Either each node is very strongly interconnected with the rest of the network, or all links are so sparse that each node is critical. Thus, in UDG graphs there are no "special" nodes. They all have very similar convergence rates. Removing fastest converging nodes is equivalent to removing randomly chosen nodes. Figure 2 just confirms the well known fact that UDG graphs are resilient to random failures.

The next question we tackle is whether there is an analytical explanation for the above presented empirical results? Recall the two global measures we defined in Section II: *average betweenness* and *square closeness*. They describe the importance of nodes for distributing information within a network. To answer the above question we test to what extend the top nodes identified by BridgeFinder overlap with the best nodes identified by the two global measures.

For each of our test networks we compute two sets: A and B. Set A consists of the highest 5% (i.e. 13 of all 250 nodes) average betweenness nodes and set B contains the highest 5% square closeness nodes. The nodes in the union of A and B, $A \cup B$, have either the highest average betweenness, or the highest square distance coefficients in the network, or in most of the cases both. Note that A and B are not necessarily disjoint. On the contrary, averaged over all generated networks, A and B overlapped to over 80%. The overlapping factor is almost identical for the four different network types.

The set $A \cup B$ contains the nodes with the "best" topological properties in the network. The question is how many of those nodes are within the nodes identified by BridgeFinder. We intersect $A \cup B$ with 5% (13 nodes) of the fastest converging nodes. Figure 3 displays the results. At least half of the fastest converging nodes lay on key topological positions in their networks. This is the case in more than 80% of the two-bridge networks and in over 90% of the one-bridge networks.

Our empirical results show that with very high probability BridgeFinder identifies the same nodes one would get by using global network measures. In both cases those are the few nodes keeping the network together, see Figure 2.

There is however still a huge difference between the two approaches. We had to use global network knowledge in centralized algorithms to compute the above discussed topological properties of the nodes. Such global knowledge is nevertheless unavailable in distributed environments, rendering that kind of algorithms useless in practice. This is the striking advantage of BridgeFinder. It relies only on information directly exchanged among the participating peers in a fully distributed manner.

Network Type	Push-Sum	BridgeFinder
3 Bridges	469	214
2 Bridges	597	282
1 Bridge	823	367
UDG	94	72

TABLE I
CONVERGENCE SPEED ON DIFFERENT NETWORK TYPES MEASURED IN AVERAGE NUMBER OF EXCHANGE STEPS PER NODE.

V. GOSSIPING CONVERGENCE

The convergence speed of the BridgeFinder algorithm is determined by the diffusion speed of the underlying network. The diffusion speed characterizes how fast a value starting from an arbitrary node diffuses through the whole network.

Another gossiping protocol based on diffusing values through the underlying network is Push-Sum [12]. We use it as a benchmark in our comparison as BridgeFinder similarly relies on a basic value exchange operation.

In graphs with high expansion the convergence speed of Push-Sum has been proven to be $O(\log N + \log \frac{1}{\epsilon})$ [12], where N is the number of nodes and ϵ the relative convergence error. For geometric networks with "roughly" uniform distribution of nodes in the Euclidian space the diffusion speed is shown to be $O(\log^{1+\epsilon} D)$ [13], where D is the diameter of the network.

Many complex networks, have a different topology than the networks described above. They consist of densely connected clusters which are only sparsely interconnected. An O-notation solution for the running time of Push-Sum in such networks is only available for special metric spaces. A general proof is not yet available [13]. Both, their irregular structure and varying number of clusters, make it very unlikely that a general solution for mobile multihop networks could ever be derived.

Therefore, we investigate the convergence of Push-Sum through extensive simulations on all 500 instances of each ODG network type that we investigated in Section IV. The averaged results are displayed in Table I. The result for a single network is measured as the required for the network to converge average number of exchange operations per node. We observe a large discrepancy in the required exchange operations between the ideal case, the UDG graph, and the most demanding case with one bridge.

Note that the gossiping protocol of Push-Sum is indeed information diffusion within the network. Therefore, the faster the node that starts the algorithm distributes information within the network, the better is the convergence time of the protocol. From Section IV we know that those are with high probability exactly the fastest converging nodes of BridgeFinder.

An intuitive solution for optimizing the original Push-Sum protocol arises. After running the algorithm once, the fastest converging node from the last run is responsible for starting the next run. If it is not able to do that, the second fastest converging node becomes responsible for starting the algorithm and so on.

This modification is integrated in BridgeFinder. Recall from Section III that the list of the fastest converging nodes is constantly exchanged among the participating nodes and that at the end of the algorithm each node possesses the list of the k fastest converging nodes. Thus, all nodes within the network are aware which node should start the next run.

We only have to overcome one last problem: starting the algorithm for the first time while avoiding different nodes from starting and running BridgeFinder simultaneously. To achieve that each node which decides to start the algorithm and has not already participated in it, picks a random number and sends it over the network together with its values.

When a node receives values with different random numbers, this means that two simultaneous instances of BridgeFinder are running. The node ignores the gossiping messages with the smaller random number and processes only the larger values. Thus, BridgeFinder instances started with larger random numbers take priority over those started with smaller numbers. This produces a slight overhead during the first run of the algorithm, as initially computed values are being thrown away, but still gives each node the possibility to start the algorithm. This resolves the problem of having to pick a starting node within a distributed environment. Any node can start the algorithm.

The benefit of starting BridgeFinder from the fastest converging nodes from previous runs of the algorithm can be seen in Table I. We ran the algorithm over the 500 instances of each network model and averaged the results. The values of BridgeFinder are always measured at the end of the second run as the first one is used to determine the top converging nodes within the network. Thanks to its modification BridgeFinder performs better than the original Push-Sum algorithm in all four network models. The acquired speed-up ranges from 20% to 220%.

In summary, even in a hostile environments like one bridge network models, our algorithm requires just a few more exchange operations per node than the overall number of nodes in the network.

VI. RELATED WORK

Gossiping is used in many network applications. One of them is the Push-Sum algorithm [1]. Its original epidemic protocol is called anti-entropy and is used for database replication [14]. Another application of the Push-Sum algorithm is counting peers in P2P overlay networks [15]. An overview of actual epidemic protocols is given by Eugster [16].

Our contribution, BridgeFinder, is fully compliant with the Push-Sum based algorithm proposed in [15]. Our modification enables BridgeFinder to detect globally critical peers in a distributed manner. Globally critical peers have significant impact on the connectivity of a network. The absence even of a small number of those nodes may easily render the network completely unusable [17].

There is a large body of recent research dedicated to identifying critical peers. The common approach is to send probe messages from a node, lets say C, to other nodes in the network. When a node receives a probe message, it must send it back to C. C contacts iteratively a chosen set of nodes [18]

or just floods the network [4]. After all probe messages have returned back to C, it computes how fast the single messages have returned. Based on its results, C decides if it is a critical peer or not. Short return times mean C is critical, otherwise it is not. After the procedure is finished the computed results are known solely to C.

Another approach is the detection algorithm based on the Midpoint Coverage Circle [3]. It also has a simple workflow. Assume that the node C wants to estimate whether it is a critical peer. It chooses random pairs of nodes within the network and asks them to contact each other. If the message exchanged by the two nodes on its way goes through C, then C considers itself critical and not otherwise. The computational overhead for C is smaller than the one in the previous approach. However, in order to guarantee reliable results C has to test a very large number of node pairs. This produces a large messaging overhead for testing just a single node. The acquired information is still solely available to C.

BridgeFinder does not test just some nodes, but rather evaluates in parallel all nodes in the network. When the algorithm is finished, the results are known to all nodes. Even with a small state per peer and based only on local computations the results are highly accurate.

In networks with expander topology the overhead for finding all critical nodes with BridgeFinder scales with $O(logN)$, where N is the number of nodes in the network [15]. This is faster than all known approaches, even though they make calculations for only one single node instead of the whole network. Our results show that in wireless multihop networks with arbitrary topologies BridgeFinder is slower with a factor between 2 and 3 compared to expander graphs. Exact O-notation estimate in such cases is a subject to future work.

VII. Conclusion

In this paper we have presented BridgeFinder, a distributed algorithm for identifying critical nodes in complex networks. Critical nodes are nodes whose disappearance would partition the network. Our extensive evaluation shows high consistency between the critical nodes identified with classic graph theory and the ones identified in a distributed manner by BridgeFinder. Detecting and protecting those few nodes is vital for keeping the network operational. The benefit of using BridgeFinder is significant in fast changing distributed environments such as wireless ad-hoc networks, sensor networks and peer-to-peer networks.

Furthermore, BridgeFinder relies only on simple exchange of information among the participating nodes. Being as generic as it is, it can be integrated in any maintenance or other existing communication flow already utilized by the network and hence cause no additional messaging overhead.

Another advantage of our approach is that it does not rely on the actual information exchanged among the participating nodes, but rather only on the convergence speed of that process. This approach is completely transparent to the gossiping process and allows the deployment of efficient guarding mechanism against malicious nodes.

To the best of our knowledge, there is no equivalent distributed approach. BridgeFinder is fast, reliable, produces almost no additional messaging overhead and only small state per peer.

VIII. Acknowledgments

The authors would particularly like to thank Karsten Weihe for many useful comments and suggestions.

This work is funded in parts by the Volkswagen Foundation under grant I/82 718 and by the German Research Foundation under grant FOR 733.

References

[1] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based Computation of Aggregate Information," *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pp. 482–491, 2003.

[2] D. Bradler, L. Krumov, E. Aitenbichler, and J. Kangasharju, "Finding communication bottlenecks in distributed environments," tech. rep., TU Darmstadt, March 2010.

[3] M. Sheng, J. Li, and Y. Shi, "Critical Nodes Detection in Mobile Ad Hoc Network," *Advanced Information Networking and Applications*, vol. 2, pp. 336–340, 2006.

[4] X. Liu, L. Xiao, A. Kreling, and Y. Liu, "Optimizing Overlay Topology by Reducing Cut Vertices," in *Proc. of the 2006 Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, (Newport, Rhode Island), pp. 1–6, ACM, 2006.

[5] R. Wattenhofer and A. Zollinger, "XTC: A Practical Topology Control Algorithm for Ad-Hoc Networks," *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004.

[6] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, pp. 35–41, March 1977.

[7] U. Brandes, "A Faster Algorithm for Betweenness Centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[8] G. Sabidussi, "The Centrality Index of a Graph," *Psychometrika*, vol. 31, pp. 581–603, 1966.

[9] C. Dangalchev, "Residual Closeness in Networks," *Phisica A*, vol. 365, pp. 556–564, 2006.

[10] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "Unit Disk Graph Approximation," *Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing*, pp. 17–23, 2004.

[11] A. Jardosh, E. Belding-Royer, K. Almeroth, and S. Suri, "Towards realistic mobility models for mobile ad hoc networks," in *Proceedings of the 9th annual international conference on Mobile computing and networking*, pp. 217–229, ACM New York, NY, USA, 2003.

[12] W. W. Terpstra, C. Leng, and A. P. Buchmann, "Brief Announcement: Practical Summation via Gossip," in *PODC*, PODC, 2007.

[13] D. Kempe, J. Kleinberg, and A. Demers, "Spatial Gossip and Resource Location Protocols," *Journal of the ACM (JACM)*, vol. 51, no. 6, pp. 943–967, 2004.

[14] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for Replicated Database Maintenance," *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pp. 1–12, 1987.

[15] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "Bubblestorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search," *SIGCOMM*, 2007.

[16] P. T. Eugster, R. Guerraoui, A. M. Kermarrec, and L. Massoulie, "From Epidemics to Distributed Computing," *IEEE Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[17] S. Saroiu, K. Gummadi, and S. Gribble, "Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts," *Multimedia Systems*, vol. 9, no. 2, pp. 170–184, 2003.

[18] T. Qiu, E. Chan, and G. Chen, "Overlay Partition: Iterative Detection and Proactive Recovery," in *IEEE International Conference on Communications 2007 (ICC'07)*, pp. 1854–1859, 2007.