Sense-Share: A Framework for Resilient Collaborative Service Performance Monitoring

Khulan Batbayar*†, Emmanouil Dimogerontakis*, Roc Meseguer*, Leandro Navarro*, Ramin Sadre†
* Universitat Politècnica de Catalunya (UPC)
{batbayar,edimoger,meseguer,leandro}@ac.upc.edu
† Université catholique de Louvain (UCLouvain)
ramin.sadre@uclouvain.be

***Abstract*—Modern large-scale networked services, such as video streaming, are typically deployed at multiple locations in the network to provide redundancy and load balancing. Different techniques are used to provide performance monitoring information so that client nodes can select the best service instance. One of them is collaborative sensing, where clients share measurement results on the observed service performance to build a common ground of knowledge with low overhead. Clients can then use this common ground to select the most suitable service provider. However, collaborative algorithms are susceptible to false measurements sent by malfunctioning or malicious nodes, which decreases the accuracy of the performance sensing process. We propose Sense-Share, a simple light-weight and resilient collaborative sensing framework based on the similarity of the client nodes' perception of service performance. Our experimental evaluation in different topologies shows that service performance sensing using Sense-Share achieves, on average, 94% similarity to non-collaborative brute force performance sensing, tolerating faulty nodes. Furthermore, our approach effectively distributes the service monitoring requests over the service nodes and exploits direct inter-node communication to share measurements, resulting in reduced monitoring overhead.**

I. Introduction

Modern network services used by a large number of clients are typically deployed at multiple geo-locations in the network to avoid the creation of bottlenecks and single points of failure. Different techniques and metrics have been proposed and used in practice to forward client requests to the best service instance. Decentralized approaches have the advantage that they do not depend on the service provider and that they allow clients to make decisions based on their situation. A major disadvantage is a large overhead: To obtain an accurate overview of the performance of all available services, each client would have to regularly probe all of them. This would require a large number of probing requests in scenarios where the service performance perceived by the clients fluctuates frequently [1] in the function of peak hours or the network condition.

In resource-constrained networks, this drawback of decentralized approaches is addressed by crowd-sourcing, collaborative sensing algorithms [2]–[4]. The collaborative sensing refers to each client monitors a subset of the available services and shares its measurement results with other clients. However, the nodes become more dependent on the benevolence of the other participants. The sensing quality depends on the active participation of each node in the measurement effort and the benignity of the nodes. Researchers have proposed anomaly detection techniques to identify faulty or malicious collaborators [5], [6] to improve the effectiveness of collaboration. Often, the security mechanism's computation complexity outweighs the actual sensing algorithm, a potential problem in constrained environments.

In this paper, we propose Sense-Share, a framework for collaborative service performance monitoring with high accuracy and low overhead. We make the following contributions:

- We propose a two-step mechanism to improve the resiliency of collaborative performance monitoring to reduce the impact of faulty or malicious collaborators on the sensing accuracy.
- We reduce the number of message exchanges between more similar close neighbors to reduce sharing of redundant measurement results and to increase measurement accuracy. We further improve the performance monitoring task of each node based on the performance deviation of each service node's measurements.
- We evaluate our framework in a virtual testbed with more than a hundred nodes and different network topologies.

II. Collaborative monitoring

A. Background

Resource-constrained, large-scale collaborative networks face the challenge of distributing and coordinating the measurements among the clients. While sharing measurement information among clients can improve the monitoring process, sharing among clients, it might not be useful to share results between clients located in very different parts of the network or to let multiple clients from the same network probe the same service. Many existing collaborative algorithms [2], [7], [8] use clustering techniques to group network nodes based on their physical location, RTT, topology, signal strength, etc and distribute the measurement tasks to nodes in the same cluster or elect a cluster head node that measures the service performance on behalf of the others. However, clustering comes with large computational overhead, further increased by potentially frequent cluster head selection, excessive inter and intra-cluster communication, etc. Furthermore, clustering makes it difficult for nodes at the cluster border to receive

978-3-903176-24-9 © 2019 IFIP

accurate measurements from others. The ideal collaborative performance monitoring framework has to find a good balance between the accuracy of the sensing, the cost of the monitoring and the management overhead.

Our previous work in [9] proposes replacing complex clustering algorithms by a much more light-weight approach where client nodes identify their close neighbors through RTT measurements. The overhead of the collaborative measurement exchange of the proposal is not optimal, since client nodes could receive redundant measurements from their neighbors. Additionally, [9] assumes that all nodes are benevolent, i.e. no node distributes false information. This assumption can result incorrect for various reasons. Firstly, the node might be simply faulty. Then, a node could be malicious and try to perform a DoS attack by directing other nodes toward the same service instance. Finally, a node could be also unfair and try to maximize its access to a particular service instance by leading other nodes away from it.

B. Design requirements

Based on the above observations, we formulate the following requirements to a general collaborative performance monitoring framework.

Accuracy (R1). Ideally, the collaboratively collected measurements at each node should be as accurate as the node's own measurements. We express the accuracy of a collaborative sensing framework for a node by the cosine similarity index defined as:

$$cos(\boldsymbol{M}, \boldsymbol{A}) = \frac{\boldsymbol{M} \cdot \boldsymbol{A}}{||\boldsymbol{M}|| \cdot ||\boldsymbol{A}||} \tag{1}$$

where $\boldsymbol{A} = [A_0, \dots, A_p]$ is the real performance metric of the p services that the node would measure without collaboration, and $\boldsymbol{M} = [M_0, \dots, M_p]$ is the estimation obtained by sharing measurement results with other nodes. As a result, an index of 1 (or 100%) means that the performance measurement results obtained through collaboration are identical to those that a node would obtain when performing its own (more expensive) measurements without sharing.

Scalability (R2). The service monitoring framework should be able to adapt to large networks with heterogeneous devices. The monitoring process should not influence other network traffic negatively, therefore the cost of monitoring should be kept minimal.

Resilience (R3). The collaborative monitoring framework is subject to faulty or malicious collaborators. We consider a node faulty or malicious if it sends wrong measurements consequently over a certain period. Figure 1 exemplifies the impact of faulty nodes on the collaborative sensing framework proposed in [9]. As observed, for different ratios of faulty nodes in the network, the Empirical Cumulative Distribution Function (ECDF) of the accuracy of the sensing results decreases significantly even if the number of faulty nodes is small. Additionally, the framework should be resilient to selfish (or lazy) nodes that do not promote active participation.

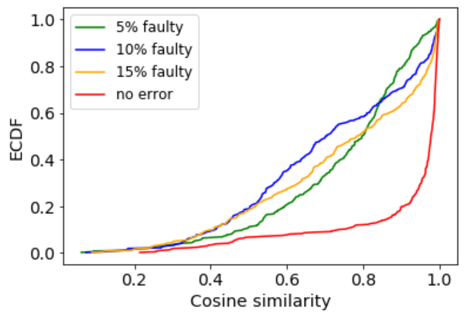

Figure 1: Similarity of perception with faulty nodes for [9]

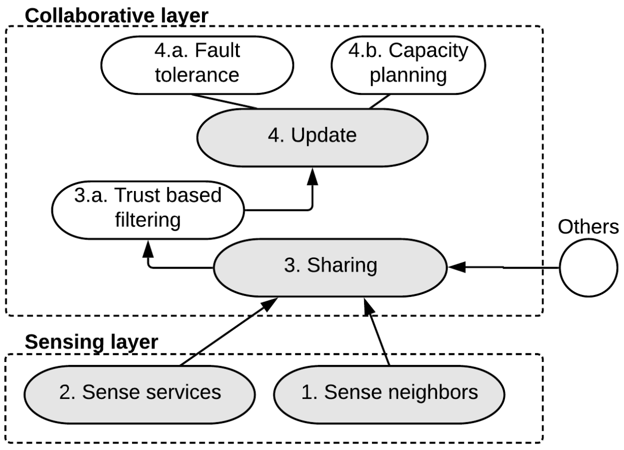

Figure 2: Layered sensing framework

Elasticity (R4). There are many uncertainties associated with the dynamic behavior of networks and services caused by, amongst others, link failure, service downtime, or congestion [1]. Timely reaction to changes is, therefore, crucial. State-of-the-art proposals address such changes with increased computational complexity, e.g caused by cluster head reelections.

III. The Sense-Share framework

Our main goal is to develop a framework for collaborative service performance monitoring meeting the different requirements (R1)–(R4). We propose a generic framework for network service performance monitoring which could be applied to any network scenario, independently from the network service measurement metric or network topology. The realtime network service monitoring results could be further used for different applications e.g, service selection, load balancing, and routing.

The framework consists of the layered architecture shown in Figure 2. Implementations of its components run on every client node participating in the collaborative sensing. The *Sensing layer* provides information related to the service performance sensing and the close neighbors sensing. The *Collaborative layer* implements the measurements' sharing among close neighbors (as received from the *Sensing layer*) and also ensures fault tolerance and optimization of message exchanges. During each round, every client node builds a service performance table from the collaborative sensing process, storing the timestamp of the last measurement, the address of the probed service instance, and the value of the measurement result. The main focus of the paper is on the *Collaborative layer*'s capability to deal with measurement errors and faulty nodes. As far as the *Sensing layer* is concerned, we maintain the approach presented in [9], and provide here a brief overview for completeness. The rest of the section presents the different components in more detail.

A. Sensing layer

The *Sensing layer* contains two components: *Sense neighbors* and *Sense services*. The *Sense neighbors* component determines the initial set of close neighbors by comparing RTT between the nodes with a threshold. Since client nodes are unaware of the underlying network infrastructure, RTT is a simple metric to determine the "proximity" of neighbor nodes. After the initial phase, the *Collaborative layer* actively decides on the selection of collaborator nodes based on their trust score. In every round, a node checks the latest (last 5 by default) history of collaborator nodes, in order to eliminate passive or disappearing collaborators.

The *Sense services* component periodically measures service instances. In each round, the client node randomly selects two services instances to be probed, among all available following the Power of Two Choices principle [10] to distribute the performance measurement requests generated by each client node. This list is updated in every round depending on the consistency of the service performance (see Section III-B3). Since the framework is generic, depending on the service to be monitored, the measurement metric could be any end-to-end metric such as download time, latency, or throughput. The complexity of the service measurement requests is defined as $O(\log \log n)$ in [10] where n is the number of services to measure.

B. Collaborative layer

The *Collaborative layer* is in charge of sharing measurement between nodes, i.e the clients send their own measurements from the *Sensing layer* to their list of trusted neighbors. Initially, when no knowledge of trust is available, all close neighbors identified by the *Sensing layer* are considered as trustworthy and all values are set to 0. This list is updated when measurements are received from them, as explained below.

When a node receives a set of measurements from another node, Algorithm 1 is executed. Each node maintains a list of close neighbors N_1 to N_k, a trust score T_i for each close neighbor i, and the result M_j of the performance measurement for each service instance S_j. The node first checks whether the sender is already known as a close neighbor (line 2). If not, the *Sense neighbors* component checks whether the sender fulfills the requirement for close neighbors (line 3) using the same mechanism used in the *Sensing layer*. If the requirement is fulfilled, the sender is added to the list of close neighbors (line 4), otherwise the measurements are rejected (line 6).

Algorithm 1 Receive

```
Require: $close\_neighbors = \{N_1, N_2, \ldots, N_m\}$
Require: $trust\_score = \{N_1 : T_1, N_2 : T_2, \ldots, N_k : T_k\}$
Require: $service\_table = \{S_1 : M_1, S_2 : M_2, \ldots, S_n : M_n\}$
 1: procedure RECEIVE($sender, measurements$)
 2:     if $sender \notin close\_neighbors$ then
 3:         if If node is close neighbor then
 4:             Add $neighbor$ to $close\_neighbors$
 5:         else
 6:             return
 7:     UPDATE_TRUST($sender$, $measurements$)
 8:     if $sender \in$ TOP-K($trusted\_neighbors$) then
 9:         for all $(service, measurement) \in measurements$ do
10:             Update service performance table for $service$
11:             Update capacity planning policy of $service$
```

The rest of the algorithm, lines 7-11, perform (a) a trust-based filtering of measurements (line 7), (b) the updates on the service performance table (line 10) and (c) updates of the capacity planning policy of the services (line 11). These steps are explained in the rest of the section.

1) Trust based filtering: The main idea of the filtering component is avoiding collaboration with less similar nodes. Each node calculates a trust score for each of its neighbors that indicates how much it trusts their measurements. In the Sense-Share framework, collaboration is limited within the close neighbor nodes, therefore trust computation can be done locally at each node. The trust is non-transitive, since a node considered as a reliable information source by a close neighbor, might not be considered so by another node. Moreover, considering that the trust values fluctuate and degrade over time [11], [12], the trust score is updated every time the node receives a measurement from a collaborator.

Algorithm 2 (called by Algorithm 1 in line 7) calculates the trust score for a $neighbor$ from which the node has received $measurements$. The trust score is the mean absolute difference between the measurements received from that node for the different services and the average of the corresponding service measurements received from other trusted nodes.

Using the trust score, each node can further reduce the number of close collaborating nodes by ranking the close neighbors and choosing to collaborate with the top K trusted ones. We define $K = min(n/2+1, k)$, where n is the number of services to measure and k is the number of close neighbors.

2) Fault tolerance: The objective of the fault-tolerance component is to reduce the impact of false (relatively higher or lower than the real measured value) measurements sent by the trusted nodes. Instead of storing the measurement result for a service directly into the service performance table, we use a weighted moving average of the results $M^{(i)}$ of the last r measurement rounds (with default $r = 2$), with

Algorithm 2 Filter trusty collaborator nodes

```
Require: $trust\_score = \{N_1 : T_1, N_2 : T_2, \dots, N_k : T_k\}$
1: procedure UPDATE_TRUST($neighbor, measurements$)
2:     Score = 0
3:     for all $(service, measurement) \in measurements$ do
4:         $m$ = AVG(all measurement of $service$ neighbors)
5:         Score = Score + $|measurement - m|$
6:     $Score = Score/size(measurements)$
7:     $trust\_score[neighbor] = Score$
```

Performance measurement	Performance deviation	Category
High	Low	Good
High	High	Inconsistent
Low	Low	Inconsistent
Low	High	Bad

Table I: Service node characterization

$\forall w_i <= w_{i+1}, w_i < 1$ and $w_i = \frac{i}{r \cdot (r+1)/2}$:

$$service\_table[service] = \sum_{i=1}^{r} w_i \cdot M_i \tag{2}$$

In case of sudden performance changes, the moving average catches up within a few measurement rounds, depending on the frequency of received measurements from the trusted neighbors for the specific service. On the other hand, intentional false measurements are smoothed out by the moving average until the *Trust based filtering* component detects the deviating behavior of the sending node.

3) Capacity based service monitoring: As mentioned in Section I, the performance of the service nodes can fluctuate significantly over time. In our framework, client nodes learn the behavior of service nodes from past measurements so that nodes can remove unreliable (i.e. bad performing or inconsistently performing) service node candidates, to further lower the monitoring cost. We consider two features to characterize the capacity of a service node to provide satisfactory performance:

1) Current performance: The service measurement M.
2) Performance deviation: The standard deviation of the service performance over the previous measurement rounds. The higher the variance, the worse the network service reliability.

In Table I, the service nodes are categorized into three categories based on the two features. Service nodes in the *Good* category have high and stable performance measurements. The *Inconsistent* service nodes have low performance or high variance. *Bad* service nodes have low performance and high variance. The average service performance value and the average performance deviation, as shaped after every round based on the measurements received from the trusted $TopK$ collaborators, define the threshold value of the *High, Low* category of the service node and performance deviation.

The *Good* category service nodes have a high probability of being a selection candidate for the node and are, therefore, more frequently sensed by the *Sense services* component by adding them into the sensing list of services upon every round. The *Inconsistent* and *Bad* category service nodes are added into the sensing list of services every other measurement round, since those service nodes are probed with less frequency which makes their transition to the *Good* category service more difficult. Depending on the number of *Inconsistent* and *Bad* service nodes, the network service selection list contains a smaller number of candidates resulting in lower collaboration level and fewer measurement requests.

IV. Experimental methodology

In our experiments, we consider the scenario of Internet gateway service monitoring. Under this use case, a large number of client nodes are connected to a set of interconnected access points. Connection to the public Internet is provided through a relatively small set of gateway nodes. Such a scenario is typical for ad-hoc networks, IoT networks and community networks. Previous works [3], [4], [9], [13], [14] have demonstrated that in order to receive the best Quality of Service in a large scale dynamic network, client nodes need to have knowledge of the real-time gateway node performances.

The gateway node performance is sensed by the client nodes by measuring the time to download a 0.1MB file from an external server located in the Internet through the gateway nodes. The download latency is a convenient end-to-end application layer metric which covers delay, RTT, bandwidth, node availability, etc. in one measurement.

We emulate a network with 100 client nodes and 10 gateway nodes in Mininet [15] v2.3 with a balanced tree topology with equal number of nodes at each branch and a custom tree topology with random number of nodes. The RTT between client nodes is randomly distributed between 5 and 20 ms and the RTT between client and gateway nodes ranges from 10 to 20 ms. The latter is extracted from anonymous daily live gateway log files collected over a period of 1 year in the guifi.net [16] community network [1], [17]. The gateways are randomly assigned with 0.3, 0.8, and 1.0 CPU capacity to create performance variations. The duration of a measurement round is set to two minutes and the RTT-threshold to identify close neighbors in the *Sensing layer* is set to 10 ms, in accordance with previous experiments [9]. The resulting average number of close neighbors for each client node is 8-10 depending on the test case.

V. Experimental results

We evaluate our framework studying its effectiveness in terms of resilience, service performance change, network node failure and sensitivity towards the framework parameters. Each experiment is run for 100 measurement rounds (i.e. 200 minutes) and experiments are repeated ten times for each topology and results are shown the average of 2 different topologies.

1) Accuracy: We, first, compare the accuracy of the SenseShare framework's performance monitoring with our previous proposal in [9]. Figure 3 shows the benefit of collaborating with more similar nodes using our trust-based approach instead

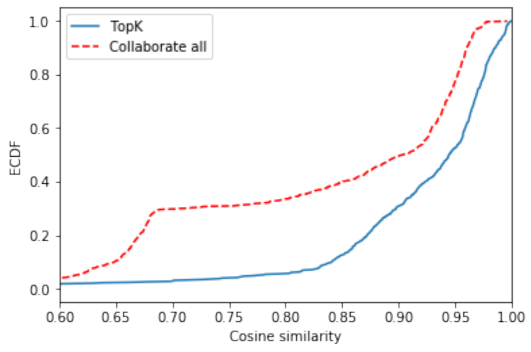

Figure 3: Similarity results for different collaboration schemes

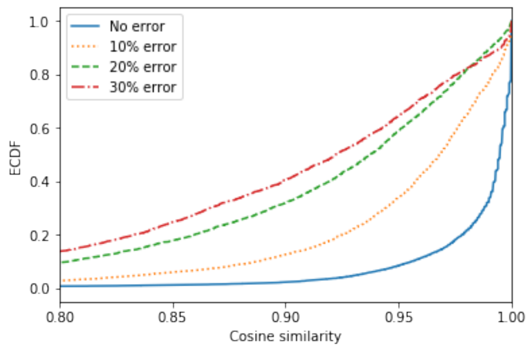

Figure 4: Similarity with the presence of faulty nodes

of collaborating with a fixed number of close neighbors. Collaborating with all close neighbors, as proposed in [9] (red, dashed line), gives an average 88% cosine similarity index of the constructed gateway measurement table. Our proposal based on the $TopK$ trusted collaborators from both the balanced tree topology and custom tree topology result is combined and shown in Figure 3 (blue, straight line) results in an average similarity index of 94%. Based on the results, the Sense-Share framework provides the same quality of collaborative measurements across the different network topologies with consistently high accurate measurements fulfilling the **R1** improved previous proposal [9] similarity by 8%.

2) Fault tolerance analysis: We performed two different experiments to explore the fault tolerance of our proposal. First, we study how the trust-based filtering component can filter the less similar collaborators. We introduce faulty nodes that are programmed to send false measurement values (original value multiplied by 5-10 times randomly) to their collaborators, in random periods. Figure 4 depicts the ECDF of the collaboratively achieved service performance table's cosine similarity, under the presence of 10%, 20%, and 30% faulty network nodes. As observed, the Sense-Share trust-based filtering component can rule out the faulty nodes and continue collaborating with the other similar (trusted) nodes in the neighbor list. As depicted in Figure 4, the average cosine similarity index is reduced from 95% on average to 93%, 92% and 90% as the number of faulty collaborators increases. This small decrease of the similarity index is necessary consequence, since some nodes are forced to collaborate with less similar nodes.

In the second experiment, we study the effect of smoothing on non-faulty peaks and valleys applying a moving average. During the experiment, a node receives from the random trusted neighbor short-bursts of high measurements (5 times more) to a randomly chosen neighbor, measuring the effect on cosine similarity of the constructed service performance table. In the results in Figure 5, we observe approximately 10 points in time during the experiment where the node receives high values. The similarity of the measurements stays relatively stable, maintaining 92% of cosine similarity index on average.

The results of the two experiments presented above show that our proposal accomplishes the requirement **R3** of

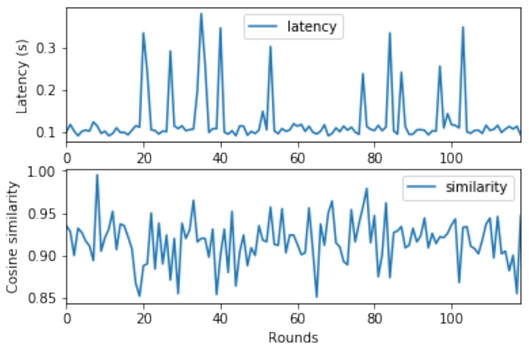

Figure 5: Behavior of the faulty nodes

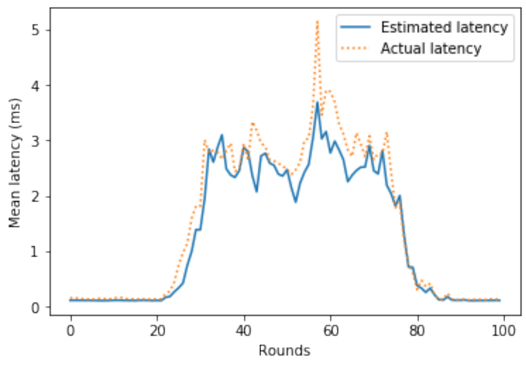

Figure 6: Gateway performance change

providing resilience towards faulty behavior.

3) Performance change analysis: Following the design requirement **R4**, the framework should adapt to any network changes either on the service node side (congestion, link failure, node failure, etc).

During the experiment, we introduce a performance change on the side of the gateway node, by introducing 200ms *netem*[1] delay over 60 rounds starting from round 20. Figure 6 plots the mean latency of the 100 client nodes towards the gateway performance. The introduced performance change on the gateway node is captured within 2-3 rounds and the mean estimated

[1]http://man7.org/linux/man-pages/man8/tc-netem.8.html

Available # of gateways	Collaborating nodes	Inter-node messages
5	3	33
10	6	18
15	8	11

Table II: Sensitivity towards the number of service nodes

	No of samples	Cosine similarity	Standard deviation
RTT<5ms	3	0.95	0.01
RTT<15ms	2	0.91	0.02
RTT<20ms	2	0.875	0.035

Table III: Sensitivity analysis (RTT threshold, no of samples)

latency stays very close and follows the shape of the actual latency curve. When removing the *netem* delay at round 80, the nodes can adjust to the performance change within the same round. Therefore, the Sense-Share framework responds to the service performance changes very fast within 0-3 rounds.

Our proposal adapts to network performance change without reducing the quality of collaborative sensing which fulfills the requirement **R4**.

4) Sensitivity analysis: In this section, we present our study concerning the sensitivity of the proposed framework in terms of the number of available services, the close neighbor sensing threshold and the number of randomized sampling which all of them affects the accuracy of the proposal.

Table II shows the number of measurements sent from each client node and the average number of received measurement requests at 5, 10 and 15 gateway nodes. From Table II, we could easily see that as the number of available services to measure increases, the number of collaborating nodes increases and at the same time the number of messages exchanged between network nodes decreases. The requirement **R2** is achieved since as presented in Table II, our framework scales the number of collaboration with the demand of the increased number of gateway nodes.

On the other hand, when there are not enough neighbors to provide full visibility of performance sensing process, the framework could either 1) increase the close neighbor sensing RTT threshold value to create a wider range of collaboration or 2) increase the number of samples from each client node to sense. Increasing the close neighbor sensing threshold results in receiving less accurate collaborative measurements at each node with the expense of reducing the cost of measurements shown in Table III. As the RTT threshold increases, the similarity of the performance measurements received from the other nodes decreased to 91% and 87.5% shown in Table III. The performance measurements received from the other nodes contain more variations, therefore, increasing the standard deviation value.

As we increase the number of samples, the cost of the measurement requests over the service nodes increases but the accuracy of the measurements also increases. This indicates the trade-off between the accuracy of performance monitoring and the measurement cost.

VI. Related work

Many collaborative service performance monitoring algorithms have been proposed in the state of the art research. We will address works related to our proposal.

The most common collaborative performance sensing algorithms are based on cluster-based collaboration where client nodes are grouped into different clusters. Similarity-based clustering algorithms were first proposed in [18] and improved over time in [2], [7], [19]. Local Negotiated Clustering Algorithm (LCNA) [20] takes account of the similarity of performance perception at the nodes and aims to find the cluster head similar to all the nodes in the cluster. Hierarchical Agglomerative Clustering (HAC) based algorithms [21], [22] propose to cluster network nodes based on the similarity of the distance between the client nodes, which is difficult to adjust to network dynamics (link failure, congestion, etc) and sensitive to outliers. Frequent cluster head elections needed for higher accuracy increase the management overhead and are, therefore, not scalable.

Decentralized algorithms are more suitable in large-scale networks due to their scalability and resiliency. The authors of [14] introduce an additional post-processing step where clients adjust measurements received from the cluster head by an estimation of the difference between them and the head. Non-clustering based collaborative algorithms [3], [23] introduce extensive gossiping and measurement overhead to provide accurate sensing. Despite providing accurate measurements, the cost of data exchange between the network nodes are high due to frequent measurements and a wide range of collaboration within network nodes. The resilient feature of collaborative algorithms has been studied in [24], [25] where they create a collaborator reputation mechanism to rank the collaborating nodes to improve the accuracy of the collaboration. Others propose anomaly detection to detect and eliminate faulty collaborator node [26]–[28]. Both approaches focus more towards detecting the faulty nodes rather than fault mitigation which we address both in the Sense-Share framework.

VII. Conclusion

We proposed a network service monitoring framework with a simple collaborative monitoring algorithm run by each client node. Based on collaboration between similar close neighbors, our framework reduces the service monitoring overhead and the inter-node communication dramatically. Experiments show that the accuracy of collaborative monitoring is very high (94% on average) with different network topology. The Sense-Share framework is resilient towards faulty collaborators by limiting the collaboration to more trusted nodes and adapts to performance changes within 1-3 rounds of sensing without compromising the quality of the collaborative measurement.

Acknowledgment

This work was supported by the EMJD-DC program, by the Spanish Government under contract TIN2016-77836-C2-2-R, and by the Generalitat de Catalunya as Consolidated Research Group 2014-SGR-881.

References

[1] E. Dimogerontakis, R. Meseguer, and L. Navarro, “Internet access for all: Assessing a crowdsourced web proxy service in a community network,” in *International Conference on Passive and Active Network Measurement (PAM).* Springer, 2017, pp. 72–84.

[2] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-efficient communication protocol for wireless microsensor networks,” in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, Jan 2000, pp. 10 pp. vol.2–.

[3] E. Dimogerontakis, J. Neto, R. Meseguer, L. Navarro, and L. Veiga, “Client-side routing-agnostic gateway selection for heterogeneous wireless mesh networks,” in *Integrated Network and Service Management (IM)*, 2017, pp. 377–385.

[4] A. Abujoda, D. Dietrich, P. Papadimitriou, and A. Sathiaseelan, “Software-defined wireless mesh networks for internet access sharing,” *Computer Networks*, vol. 93, pp. 359–372, 2015.

[5] L. G. Jaimes, I. J. Vergara-Laurens, and A. Raij, “A survey of incentive techniques for mobile crowd sensing,” *IEEE Internet of Things Journal*, vol. 2, no. 5, pp. 370–380, Oct 2015.

[6] X. Li and Q. Zhu, “Social incentive mechanism based multi-user sensing time optimization in co-operative spectrum sensing with mobile crowd sensing,” *Sensors*, vol. 18, no. 1, 2018.

[7] O. Younis and S. Fahmy, “Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks,” *IEEE Transactions on Mobile Computing*, vol. 3, no. 4, pp. 366–379, Oct 2004.

[8] Y. Fernandess and D. Malkhi, “K-clustering in wireless ad hoc networks,” in *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*, ser. POMC ’02. New York, NY, USA: ACM, 2002, pp. 31–37.

[9] K. Batbayar, R. Meseguer, E. Dimogerontakis, L. Navarro, and R. Sadre, “Collaborative informed gateway selection in large-scale and heterogeneous networks,” in *IM 2019 ()*, Washington DC, USA, apr 2019.

[10] M. Mitzenmacher, “The power of two choices in randomized load balancing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 10 2001.

[11] J. Cho, A. Swami, and I. Chen, “Modeling and analysis of trust management for cognitive mission-driven group communication systems in mobile ad hoc networks,” in *2009 International Conference on Computational Science and Engineering*, vol. 2, Aug 2009, pp. 641–650.

[12] I. Chen, F. Bao, M. Chang, and J. Cho, “Dynamic trust management for delay tolerant networks and its application to secure routing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1200–1210, May 2014.

[13] Y. Kim, Y. Jeong, M. Seo, and J. Ma, “Load-balanced mesh portal selection in wireless mesh network,” in *MILCOM 2007 - IEEE Military Communications Conference*, Oct 2007, pp. 1–6.

[14] B. J. Ko, S. Liu, M. Zafer, H. Y. S. Wong, and K.-W. Lee, “Gateway selection in hybrid wireless networks through cooperative probing,” in *Integrated Network Management (IM).* IEEE, 2013, pp. 352–360.

[15] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[16] guif.net. Guifi.net, open, free and neutral community network.

[17] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, and L. Navarro, “A technological overview of the guifi.net community network,” *Computer Networks*, vol. 93, pp. 260 – 278, 2015.

[18] R. Jarvis and E. Patrick, “Clustering using a similarity measure based on shared near neighbors,” *IEEE Transactions on Computers*, vol. 22, no. 11, pp. 1025–1034, nov 1973.

[19] Miin-Shen Yang and Kuo-Lung Wu, “A similarity-based robust clustering method,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 4, pp. 434–448, April 2004.

[20] D. Xia and N. Vlajic, “Near-optimal node clustering in wireless sensor networks for environment monitoring,” in *21st International Conference on Advanced Information Networking and Applications (AINA ’07)*, May 2007, pp. 632–641.

[21] I. Okeke and F. Verdicchio, “Shape-based clustering in wireless sensor networks,” in *2017 IEEE SENSORS*, Oct 2017, pp. 1–3.

[22] C.-H. Lung and C. Zhou, “Using hierarchical agglomerative clustering in wireless sensor networks: An energy-efficient and flexible approach,” *Ad Hoc Networks*, vol. 8, no. 3, pp. 328 – 344, 2010.

[23] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: A decentralized network coordinate system,” in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM).* ACM, 2004, pp. 15–26.

[24] H. Lin, J. Hu, J. Ma, L. Xu, and Z. Yu, “A secure collaborative spectrum sensing strategy in cyber-physical systems,” *IEEE Access*, vol. 5, pp. 27 679–27 690, 2017.

[25] T. Zhang, R. Safavi-Naini, and Z. Li, “Redisen: Reputation-based secure cooperative sensing in distributed cognitive radio networks,” in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 2601–2605.

[26] J. Zhao, Q. Liu, X. Wang, and S. Mao, “Scheduling of collaborative sequential compressed sensing over wide spectrum band,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 492–505, Feb 2018.

[27] B. Sun, X. Shan, K. Wu, and Y. Xiao, “Anomaly detection based secure in-network aggregation for wireless sensor networks,” *IEEE Systems Journal*, vol. 7, no. 1, pp. 13–25, March 2013.

[28] L. Lyu, Y. W. Law, S. M. Erfani, C. Leckie, and M. Palaniswami, “An improved scheme for privacy-preserving collaborative anomaly detection,” in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, March 2016, pp. 1–6.