Realising Dynamism in MediaSense Publish/Subscribe Model for Logical-Clustering in Crowdsourcing

Hasibur Rahman[*], Rahim Rahmani, Theo Kanter
Department of Computer and Systems Sciences (DSV)
Stockholm University
Nod Buildning, SE-164 07 Kista, Sweden

***Abstract*—The upsurge of social networks, mobile devices, Internet or Web-enabled services have enabled unprecedented level of human participation in pervasive computing which is coined as crowdsourcing. The pervasiveness of computing devices leads to a fast varying computing where it is imperative to have a model for catering the dynamic environment. The challenge of efficiently distributing context information in logical-clustering in crowdsourcing scenarios can be countered by the scalable MediaSense PubSub model. MeidaSense is a proven scalable PubSub model for static environment. However, the scalability of MediaSense as PubSub model is further challenged by its viability to adjust to the dynamic nature of crowdsourcing. Crowdsourcing does not only involve fast varying pervasive devices but also dynamic distributed and heterogeneous context information. In light of this, the paper extends the current MediaSense PubSub model which can handle dynamic logical-clustering in crowdsourcing. The results suggest that the extended MediaSense is viable for catering the dynamism nature of crowdsourcing, moreover, it is possible to predict the near-optimal subscription matching time and predict the time it takes to update (insert or delete) context-IDs along with existing published context-IDs. Furthermore, it is possible to foretell the memory usage in MediaSense PubSub model.**

Keywords*—Internet; crowdsourcing; pervasive computing; context information; dynamism; context-ID; logical-clustering; Publish/Subscribe; MediaSense

I. Introduction

The penetration of pervasive devices is escalating and the rate of proliferation is always on the rise. This pervasiveness of computing devices has paved the way where any situation can be sensed and analyzed anywhere for anything. This directly corresponds to the distributed dissemination and acquisition of context information from physical objects. This has become possible by and large due to spontaneous human participation from online community which is most popularly known as *crowdsourcing*. This trend of crowdsourcing has been facilitated by the deployment of pervasive devices along with increasing popularity of Internet-enabled services and the trend is expected to upsurge. For instance, billions of mobile devices are already in use today and Ericsson predicts that 50-500 billion mobile devices will be in use by 2020 [1]. This coupled with increased deployment of sensors in the Internet-of-Things (IoT) will empower human to spontaneously participate in the crowdsourcing. Social-networks are anticipated to contribute to this cause as well, for example, a tweet feed can be considered as sensor data [11]. This surge of social networks, mobile devices, Internet or Web-enabled services have enabled unprecedented level of human participation in crowdsourcing which has been branded as “human-in-the-loop-sensing” or citizen sensor networks [12, 13]. This phenomenon has allowed us to encounter vast amount of real-time crowd-sourced data from distributed context sources. Ericsson envisions a world which is connected in real-time with people using things around us to create new innovative ideas- which is known as the Networked Society [1]. This Networked Society can be viewed as another way of defining the crowdsourcing. In a nut-shell, the followings are the properties and requirements for crowdsourcing:

- People
- Pervasive devices
- Internet or Web-enabled services
- Surrounding things
- Context Information

Although crowdsourcing is gaining popularity very fast and this, however, brings forth many challenges in the real-time distributed systems communication. Sharing heterogeneous context information obtained from distributed sources is one of them [4, 5, 11]. Publish/Subscribe (*PubSub*) model has perhaps emerged as most popular and efficient form of communication system to sharing ubiquitous context information. PubSub is an enabler for real-time context information sharing and providing means of notification for distributed devices [4, 5, 6, 7, 11]. By leveraging the PubSub in the crowdsourcing model can unravel the challenge of sharing context information in real-time [18].

Research in pervasive computing has resulted in MediaSense and was originally developed by the research group called *Immersive Networking* as context sharing platform in the Internet-of-Things domain based on peer-to-peer (p2p) technologies [2, 3]. MediaSense can run on any platform that runs JAVA. However, the promise and potential of

MediaSense makes it a good candidate to utilize it beyond the mentioned scope. It has the potential to be utilized in crowdsourcing domain. MediaSense is an open source platform which can be used for real-time and scalable seamless context sharing [2, 3].

In response to the challenge of sharing context information in crowdsourcing, our previous paper presented the scalable MediaSense platform as the PubSub model [18]. Results suggested that MediaSense platform is very fast, efficient and capable of supporting large-scale system. However, as crowdsourcing evolve around pervasive devices and pervasive computing is always changing and this dynamic nature of pervasive computing further challenges the scalability of PubSub model. A PubSub model must cope with the fast varying anytime, anywhere computing i.e. crowdsourcing. The distributed objects with heterogeneous context sources demand scalable computing when detecting changes and adjusting accordingly. The changes could be anything such as network connectivity, bandwidth, insertion and deletion of PubSub items, etc. Moreover, since logical-clustering involves physically distributed but logically synchronized sinks, hence it is mandated that we investigate its stability in case of failure of one of the sinks. The natural question arises what happens one of the sinks is down? Will the system be stable? Can MediaSense still be able to synchronize without failed sink(s)? Therefore, this mandates that we further examine MediaSense's scalability in dynamic environment. The aim is to enable real-time response to the fast varying nature of crowdsourcing. The massive scale of context information in crowdsourcing requires adjusting to the dynamic environment along with efficient and scalable acquisition, dissemination, and management. This paper particularly enlightens MediaSense's impact as PubSub model for dynamic crowdsourcing environment.

The rest of the paper is organized as follows: section II shows the related work, section III outlines the motivation of the work, section IV draws the approach while section V demonstrates the evaluation of the work, finally section VI concludes the paper and briefly hints at the future work.

II. Related Work

Related work in the aforementioned scenario focused on feasibility of using Publish/Subscribe model for mobile systems [4] where they focused on scalability and mobility issues; for mobile crowdsensing which focused on real-time data delivery and saving energy [5]. And others have proposed different methods to implement PubSub, for example, Le Subscribe proposed web based publish/subscribe system [6, 7], the Toronto Publish/Subscribe System (ToPSS) utilized DBMS-based matching algorithm [8] and PARDES implemented rule-based matching algorithm [9] for PubSub model. None of the above mentioned model alone offers the advantages that MediaSense offers as highlighted before.

Franco in [10] portrayed that spontaneous human participation i.e. crowdsourcing is pivotal for future pervasive computing. The human engagement in distributed collaboration would enrich the urban networks which will implement the idea of sensing, actuating and computing anything anywhere and anytime. Human participation in real-time crowdsourcing is further highlighted in [12, 13]. Demirbas et al. in [11] also illustrated crowd-sourced sensing and they showed Twitter as an example of achieving this. Ericsson [1] predicts that in future people will be connected along with things and will produce innovative ideas through the Networked Society. All these researches show that heterogeneous context will be generated from distributed sources in real-time. In light of this, one of our previous papers proposed the idea of logical-clustering based on context similarity [14] and we further demonstrated its performance in [15]. The definition of context by Dey AK (2001, [17]) is widely accepted, based on this our definition of context is: "*Sensor's flow packets that describe the current situation of the sensor*". Although our initial proposal concentrated generally on wireless sensor networks scenario and flow-sensors, however, our approach has the ingredients to suit the crowdsourcing platform as well. Similar context is the basis for logical-clustering. Context similarity is calculated based on similar flow of context of flow- sensors [14, 15]. Our proposal implies that heterogeneous context generated from distributed sources would be logically clustered based on context-similarity. The main goal of our research was to provide a mean for managing huge context information in a proficient manner. The challenge of sharing clustering identification has been addressed in our previous paper [18] by employing a PubSub model in MediaSense. This opens up the floodgate for sharing the clustering identification. This PubSub would act like a driving wheel for logical-clustering concept.

Zaslavsky in [19] portrayed that key to efficient pervasive computing i.e. crowdsourcing is to adjust applications' behavior and functionality. This underpins the need for applications' capability to cope with the dynamic environments. An application cannot be called scalable if it fails to address the aforementioned scenario. This was further discussed in [20] that it is inconvenient if pervasive system is static i.e. if not dynamic.

III. Motivation

The unprecedented power and promise of pervasive devices capitalized by human will lead the future pervasive environment. Huge amount of heterogeneous data i.e. context information generated from crowdsourcing necessitates proper management; and logical-clustering of context is one of the techniques to manage context information proficiently and share resources remotely thus enabling heterogeneous interoperability [14]. This approach can even be applied to the Networked Society concept where similar ideas from connected people can be categorized into a cluster meaning that clustering will be done based on similar context i.e. ideas. However, solution to the PubSub of context-IDs was missing in the existing proposal. Therefore, the primary motivation of this work is to address the PubSub issue of the proposed logical-clustering concept. In logical-clustering, each cluster is identified as context-ID and published on the Internet so that other interested entities can subscribe to the context-ID. The idea of logical-sink was utilized to control the enormous number of entities in a small-scale network. Logical-sink implies that sinks will be physically distributed but logically synchronized. PubSub is the enabler for accomplishing logical-sink. In our previous paper [18], we adopted MediaSense as PubSub enabler in logical-clustering. This approach solved the PubSub issue for both fronts i.e. for dissemination (publishing) of context-IDs in the Internet and for logical-sink synchronization. Fig. 1 (elaborated further in next section) shows the incorporation of MediaSense into the logical-clustering concept. Diversity and heterogeneity are not only related to the context information but also to the environment itself. Our previous paper dealt with the static scenario where only regular publish/subscribe items have been addressed. The paper did not take into consideration of dynamic situation where it might require to alter or update the context-IDs along with regular publish/subscribe. This motivated us to investigate further the MediaSense credibility whether it can match the demand of crowdsourcing dynamism. In addition, it has been observed that MediaSense initially takes some time to match a subscription compared with other distributed system such as PARDES system, therefore, another goal of this paper is to identify the reason behind this delay and propose a potential solution to the problem. With the ever increasing smartdevices and increasing popularity of intelligent systems, it is desirable to have a model which can predict the outcome in some capacity. And this paper will also explore if it is possible to predict the PubSub messages per second and the memory consumption for which the MediaSense was evaluated in the previous paper. Finally, it is unknown what happens when one of the physical sinks down and logical-sink synchronization, stability will further be evaluated.

IV. APPROACH

Firstly, this section briefly discusses how MediaSense works and follows by modifications made to the current

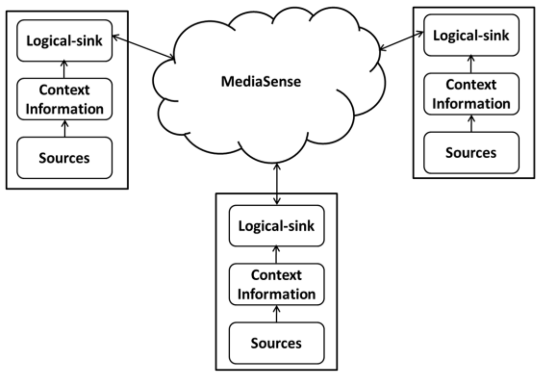

Fig. 1. MediaSense as PubSub model in logical-clustering

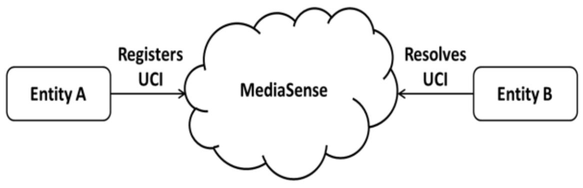

Fig. 2. MediaSense registering and resolving UCI

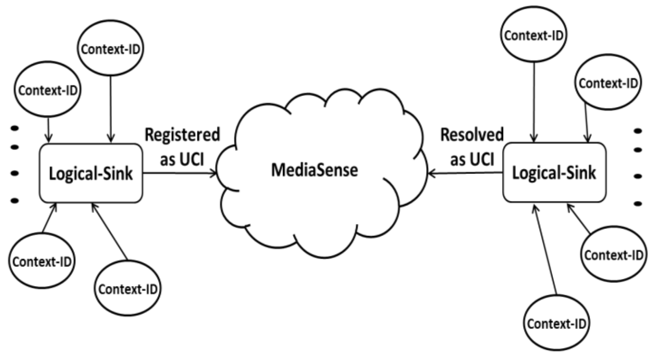

Fig. 3. Our approach to utilize MediaSense

MediaSense implementation to adjust to the approach i.e. using MediaSense as PubSub model is highlighted.

A. *MediaSense*

MediaSense uses a p2p infrastructure and implemented in JAVA. Distributed Context eXchange Protocol (DCXP) is used to disseminate information between all the entities that are using the platform. MediaSense can offer real-time context sharing, and context entity is referred to as Universal Context Identifier (UCI) in MediaSense. An entity requires to resolving this UCI in order to fetch context information, but before an entity can fetch context information the entity that holds the context information needs to be registered. Fig. 2 gives an idea of how this mechanism works. Entity A registers a UCI in MediaSense using the Registrator class and entity B resolves the UCI by using the Resolver class to fetch context information associated with the resolved UCI. An entity can register more than one UCI. However, the only drawback with MediaSense is that an entity needs to know the UCI prior to resolving.

B. *MediaSense as PubSub in logical-clustering*

The contribution of this paper begins with adoption of MediaSense into logical-clustering concept. This sub-section describes the approach and modifications made to the MediaSense platform to fit into the proposal. Currently, an entity registers the host ID and hash key along with the UCIs. Host ID and hash key remain unchanged for a particular entity. The idea is that a logical-sink registers itself as UCI and the context-IDs associated with the logical-sink as UCI's data. Other logical-sink resided remotely resolves the UCI and fetches the context-IDs. This is shown in fig. 3. Logical-sink collects data i.e. context information from distributed sources e.g. sensors, mobile devices and other physical objects that produce context information, and is responsible for creating the context-IDs based on the context similarity (see fig. 1). Logical-sink needs to be synchronized as well i.e. changes in a physical sink should be synchronized with other physical sink(s). This synchronization could be achieved by the MediaSense PubSub model too. Fig. 4 illustrates this. In this later case, a physical sink would be registered as UCI and changes inside the sink would be shared with other physical sinks over MediaSense. Therefore, our approach would be evaluated for both these purposes.

However, the current MediaSense implementation does not support the registration of context information along with the UCI at the same time. Rather it collects context information and this is sent over MediaSense as a message. This method would incur delay in our approach as there might be millions of context-IDs to be published and subscribed. Hence, the MediaSense platform has been modified in a manner that the context information can be registered at the same time as UCI. Therefore, whenever a logical-sink is registered, its context information is also registered in parallel. This will further enable faster and real-time synchronization of context information. And, changes in the logical-sink can be updated using the MediaSense Updater class. Fig. 5 & 6 show the algorithms for UCI and context information registration and resolve. Algorithm for registration first begins with initializing MediaSense platform and starting the MediaSense bootstrap. MediaSense bootstrap needs to be initiated only once inside a network. As we assume that MediaSense entities are already up and running, so time to set MediaSense up is not included in the evaluation. The algorithm next checks if the UCI is registered. UCI is updated with new and old context information- if UCI is already registered. Otherwise, UCI is registered along with its context information. The registered UCI can be deleted and a logical-sink in essence can register multiple UCIs at the same time. This gives us flexibility; for example, an entity acting as both physical sink (part of logical-sink) and logical-sink (while communicating other logical-sinks) can communicate with other entities using different UCIs. The registered UCIs are saved on the MediaSense platform which means the context information is never lost, as long as the UCI is not deleted, when an entity dies or fails. This guarantees no central point of failure.

Fig. 6 shows the algorithm for resolving UCI. The algorithm first resolves the context information from the UCI if it exists. The algorithm then fetches context-IDs until the list is empty. The context-ID that is to be subscribed is then checked against the fetched context-IDs and a notification message can be sent to the subscription requestor when match is found. If the UCI is being requested to be resolved is nonexistent then a message notifies that UCI does not exist.

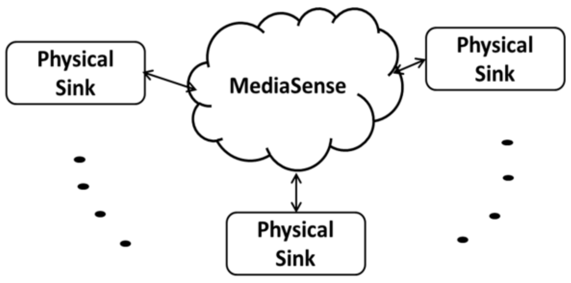

Fig. 4. MediaSense as PubSub for logical-sink synchronization

```
Algorithm UCIRegistration

Initialize MediaSense platform
Run the MediaSense bootstrap
// measurement starts from here
if UCI is not registered
        Invoke Registrator class
        Initialize registration and add UCI invoking MediaSensePlatform's registerUCI method
        Add context information
else if
        Invoke Update class
        Initialize Updating and update UCI invoking MediaSensePlatform's update method
        Update context information
end if

end UCIRegistration
```

Fig. 5. Algorithm for UCI and context information registration

```
Algorithm UCIResolve

Initialize MediaSense platform
// measurement starts from here
if UCI exists
        Invoke Resolver class
        Initialize resolve and resolve UCI invoking MediaSensePlatform's resolveUCI method
        Resolve context information
        while context-ID list is not empty
              get context-ID
              if list contains context-ID
                     subscription matched
              end if
        end while
else if
        UCI does not exist
end if

end UCIResolve
```

Fig. 6. Algorithm for UCI and context information resolve

V. EVALUATION

This section first begins with highlighting the need for modification and then exhibits the evaluation of MediaSense as a PubSub model.

The evaluation can be divided into three parts: (i) PubSub for the context-IDs sharing in logical-clustering for which each published context-ID is matched for subscription, and (ii) PubSub for logical-sink synchronization for which all the changes are published to the other physical-sinks, and (iii) dynamic behavior of MediaSense.

TABLE I. REQUIRED TIME FOR PUBLISHING

# of published context-IDs	Current MediaSense	Modified MediaSense	% improvement
1000	7.34 ms	4.17 ms	76
10000	8.93 ms	5.37 ms	66
100000	10.74 ms	6.23 ms	72
200000	11.65 ms	6.69 ms	74

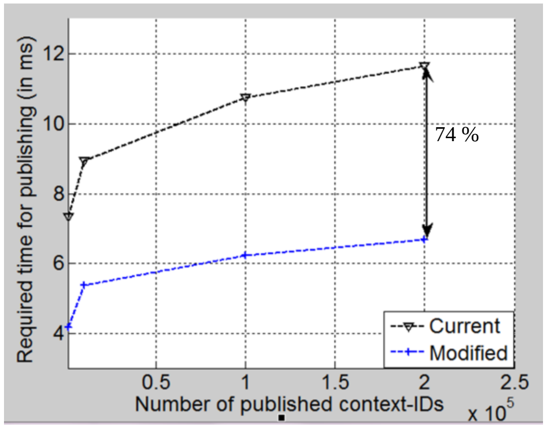

Fig. 7. Publishing time difference in MediaSense (current vs. modified)

A. *Current vs. Modified MediaSense*

In current MediaSense, if we want to share context-IDs then each context-ID would need to be registered as UCI. This will sustain delay. Table II summarizes the time required to publish items i.e. context-IDs on current and modified MediaSense platform. It can be clearly seen that current MediaSense takes longer time compared to the modified MediaSense- if we publish context-IDs as UCIs. Hence, it is efficient to register context-IDs as context information and sink as UCI. This way we can achieve nearly 74 % improvement. Fig. 7 further illustrates this.

B. *MediaSense for logical-clustering*

The PubSub model that we proposed initially for logical-clustering could send maximum 1000 messages/sec for PubSub events. However, we have achieved better result with MediaSense. It can support as high as 3537 messages/sec. This result has been obtained by running the PubSub for 1 second and result is the average for multiple simulations. This gives an increase of 254 % which outperforms our former idea. It clearly shows that MediaSense can be an efficient PubSub model. The rest of this sub-section will demonstrate performance of MediaSense for various scenarios and under assumption that all the MediaSense entities are already up and running. In order to evaluate its performance we have used three PCs with one PC acting as host sink and remaining two as recipient sinks. All three PCs have similar RAM size but the recipient sinks have different processors. The results have been obtained by simulating multiple times and the average results have been presented. Subscription matching time is shown in logarithmic scale and in milliseconds (ms).

Fig. 8 shows MediaSense's performance for different number of published context-IDs. This result is obtained for both published and subscribed duration. Context-IDs have been generated randomly using UUID in JAVA. For this particular scenario, each of the published context-ID is matched for subscription on the recipient sinks. It can be seen that both sinks give almost similar results. No significant fluctuation in terms of performance. MediaSense provides PubSub messages per second of around 2911, 1789, and 931 for context-IDs size of 10K, 50K, and 100K respectively. Although it is apparent that the performance reduces with the increase size of context-ID, but PubSub lowers only by one-third while the magnitude of the context-ID increased by ten-fold. This is due to the fact that time for resolving UCI increases when we want to publish and subscribe larger size. Moreover, subscription matching always vitiates when published item increases as can be seen from previous examples of PubSub [6, 7, 8, 9]. This can be understood from the fact that with the increased size of published item, the matching takes longer time.

Fig. 9 shows the subscription matching for context-IDs in MediaSense. Again almost identical performance for both sinks. Subscription matching duration understandably increases with the size of context-IDs. The result suggests that for hundred-fold increase in the context-ID size, matching duration increases only by 86 %.

Fig. 10 shows subscription matching time for a single context-ID. The ^{i}th context-ID is matched from i-size of the context-ID. Surprisingly, sinks have slightly different result for this scenario. The difference largely can be seen at the beginning (for 100K) and for 1 million. The one-millionth context-ID took 8.76 ms to match with the published context-IDs. While most of the PubSub systems are centralized and do not scale well in the distributed computing, the PARDES large-scale PubSub system in [9] is a distributed PubSub system which showed that one publication can be matched in 4.25 ms for 200K subscriptions, although for our approach we are matching subscription against published items and result illustrates that it takes 7.71 ms to match 200,000th item for 200,000 published items in real-time. This increase perhaps due to time required to resolve UCI with large context-IDs (see further fig. 14).

However, if we analyze fig. 11 it can be observed that the increase rate for subscription matching is much higher in PARDES compared to MediaSense. The matching rate increases nominally for MediaSense. It increases by merely 7% when context-IDs increase from 500K to 1 million and from 1 million to 2 million. As for PARDES, we see that it increases by 54%, 89%, and 125% when subscriptions increase from 25K to 50K, 50K to 100K, and 100K to 200K respectively. Since PARDES did not show its results beyond 200K and if we take the minimum increase rate which is 54% and plot them, then we see that PARDES overtakes MediaSense from 500K and beyond. MediaSense shows 99% improvement compared to PARDES for 2 billion context-IDs matching. This result signifies that our approach is easily suitable for large-scale PubSub scenarios and scales very efficiently with nominal increase in matching duration in a distributed large-scale scenario. The scalability efficiency can further be seen from table II and III. It is mentioned earlier that for all the PubSub systems, PubSub messages/sec decreases with the increase in published items. Le Subscribe system is a very efficient and fast PubSub system as outlined in [6, 7], but our approach has outperformed its counting algorithm as table II and III confirm. MediaSense achieves as high as 2058% increase in subscription matching and 1200% increase in PubSub messages/sec. Although Le Subscribe has other algorithms which performs better compared to its counting algorithm, but the other algorithms eliminate a portion of subscriptions to achieve this. This contradicts our approach and we do not eliminate any context-ID (i.e. subscription), hence other algorithms were not considered for comparison. And we have shown that our approach performs better compared to other approaches.

The above scenarios have been evaluated on the same network and with same Internet speed. In order to verify whether Internet speed plays a significant role in the MediaSense performance, we have tested our approach in a different network with one-third slower Internet speed. Fig. 12 illustrates this case. The result demonstrates that Internet does play a role in determining the performance. Interestingly, the fluctuation mostly varies between 5K and 20K. As for 50K and 100K, the fluctuation is insignificant. For example, for the size of 10K, network-2 (with low speed) shows 31 % performance reductions while for the 100K size, the decrease is merely 3 %. This indicates that although with low Internet speed MediaSense demonstrates slight performance reduction, however, the decrease rate is marginal.

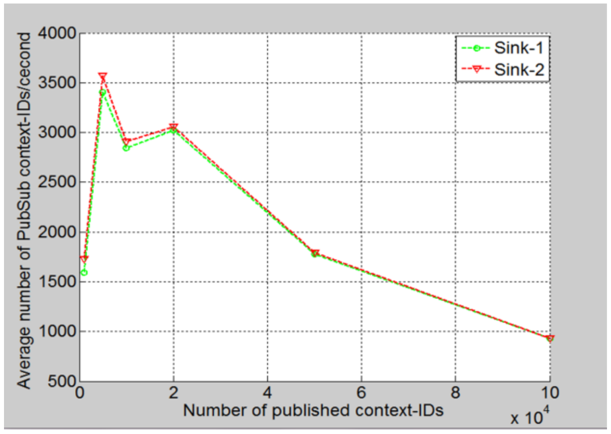

Fig. 8. MediaSense PubSub messages per second

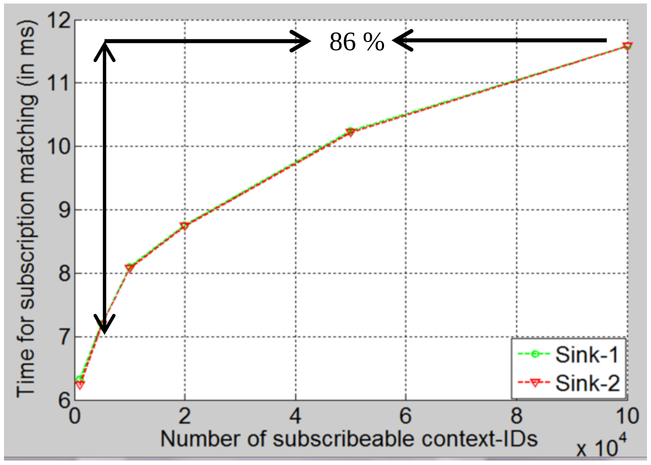

Fig. 9. MediaSense subscription matching

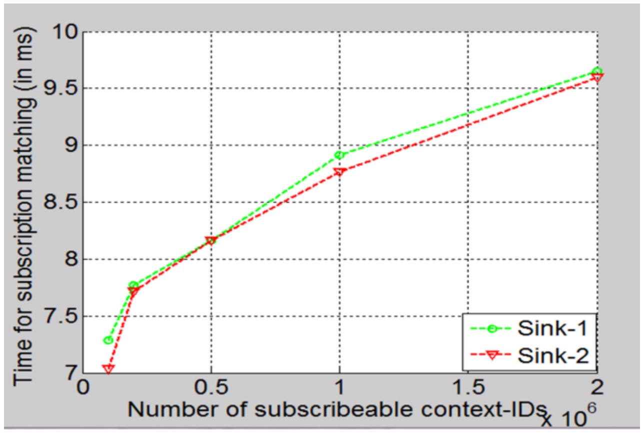

Fig. 10. MediaSense subscription matching for ith item

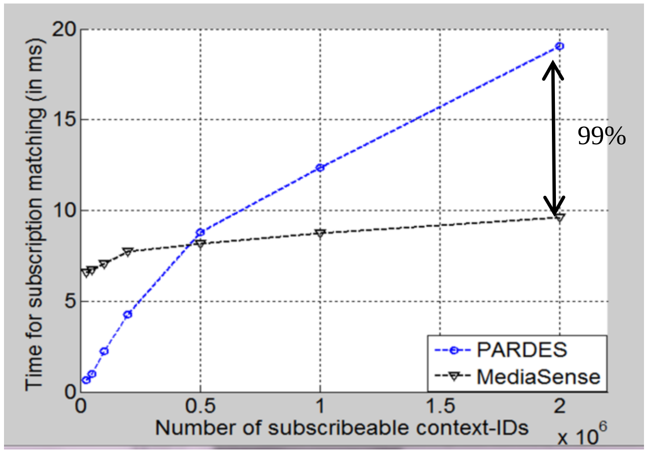

Fig. 11. Subscription macthing time comparisons

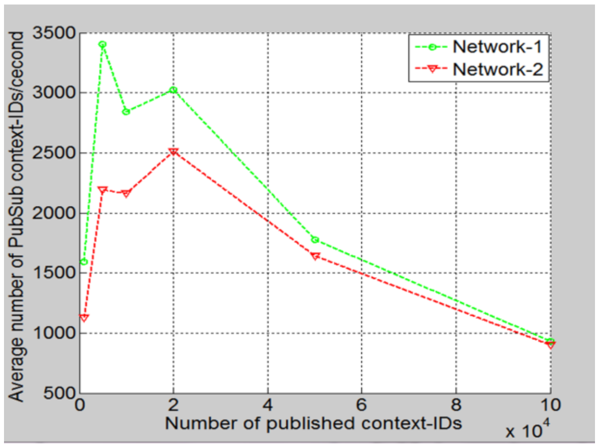

Fig. 12. MediaSense PubSub messages per second in different Internet speed

TABLE II. SUBSCRIPTION MATCHING

# of context-IDs	Le Subscribe (Counting)	MediaSense	% improvement
500 K	85 ms	14.76 ms	476
1 million	350 ms	16.22 ms	2058

TABLE III. PUBSUB MESSAGES/SEC

# of context-IDs	Le Subscribe (Counting)	MediaSense	% improvement
15 K	621	3151	407
1 million	7	91	1200

C. *MediaSense for logical-sink*

As for logical-sink i.e. synchronization of physical sinks, matching for published items is not required. In order to synchronize each physical sink, only the changes need to be retrieved in other sinks. And, depending on the nature of changes and need, each physical sink would decide whether to save the changes in a file or as UCI on the MediaSense. And, since no matching operation required in this case, MediaSense can provide as high as 9032 event changes per second. This is a further improvement by factor of nearly 3 compared to PubSub messages per second. This overwhelming number makes MediaSense a very competent and efficient tool for PubSub model in crowdsourcing- especially for the purpose of logical-clustering.

D. *MediaSense memory usage*

Memory usage plays an important part in the PubSub model evaluation as highlighted by earlier researches [7, 8, 9]. MediaSense is very efficient in terms of memory usage as well. Fig. 13 confirms this. Memory usage grows linearly. 37 MB of memory is required in order to store 1 million context-IDs. ToPSS PubSub prototype in [8] and Le Subscribe prototype (the counting algorithm was described in [6] and its memory usage was shown in [7]) required very large memory sizes, for example, ToPSS occupied minimum of 4400 KB memory to store 1000 subscriptions, and in our approach it is possible to store 1000 subscription with 39 KB of memory. This gives an 11216 % improvement in terms of memory usage for this particular scenario. However, this is not always the case as illustrated in table IV. The table further shows the comparison between these three PubSub models. MediaSense and Le Subscribe grow linearly. Table IV also reflects this where MediaSense's % improvement compared to ToPSS varies and the comparison is stable with Le Subscribe in terms of memory requirements. MediaSense betters Le Subscribe and ToPSS respectively by 163% and minimum by 451%.

TABLE IV. MEMORY USAGE

# of context-IDs	MediaSense	ToPSS (Kdb)	Le Subscribe (Counting)	% improveme-nt
1000	0.038 MB	4.3 MB	-	11216 / -
1 million	37.1 MB	381.46 MB	97.66 MB	928 / 163
2 million	74.38 MB	762.94 MB	195.31 MB	926 / 163
5 million	185.97 MB	1024 MB	488.28 MB	451 / 163

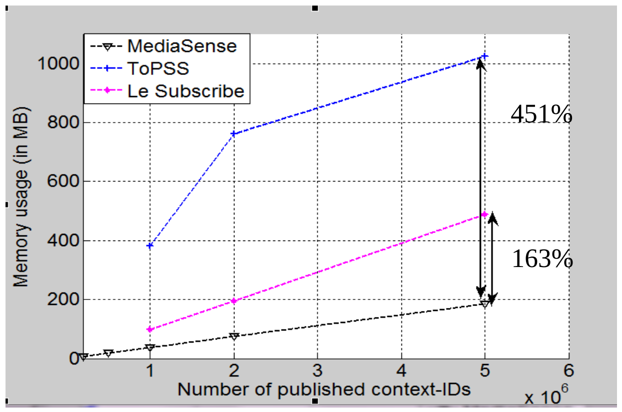

Fig. 13. MediaSense memory usage

E. *UCI resolved delay analysis*

We have seen in fig. 10 & 11 that context-ID matching takes bit long time initially and we further assumed that this could be due to the time that sink takes to resolves UCI. We have seen from fig. 10 & 11 that subscription matching grows linearly but initially takes some time. If the time required to resolve UCI can be ignored then this could result in faster subscription matching which is desirable in real-time computing. The following figures (fig. 14 & 15) further discuss the issue. First, fig. 14 shows the comparison for subscription matching between UCI resolved and without UCI resolved. The result in this particular figure has been simulated for context-ID matching for every published context-ID. The result is out of the blue for us, we did not expect this result. Our assumption was that without UCI resolved would result in faster context-ID subscription matching. However, MediaSense demonstrated almost identical performance for both scenarios.

For example, MediaSense demonstrated only 23% increased subscription matching time for UCI resolved compared to without UCI resolved for 5K published context-IDs. Moreover, this subscription matching time reduces to almost 0% if the published context-ID is increased to 100K. This could be understood from the fact that as we are matching for each published context-ID and time for subscription is matching is short (measured in ms) as well as for UCI resolving. Therefore, with the increase of published context-ID, the resulting subscription matching is independent of time required for UCI resolving. Nonetheless, if we now examine fig. 15 we can see the significance of discarding required time for UCI resolving.

Fig. 15 shows the subscription matching required for ^{i}th context-ID from i-size of the context-ID. Fig. 11 also showed the result for this scenario. Fig. 15 clearly shows the difference. Since pervasive computing is a dynamic environment and more often than not it is desirable to match a context-ID as fast as possible with minimal delay. This motivated us to look into a solution for finding a faster approach for context-ID matching.

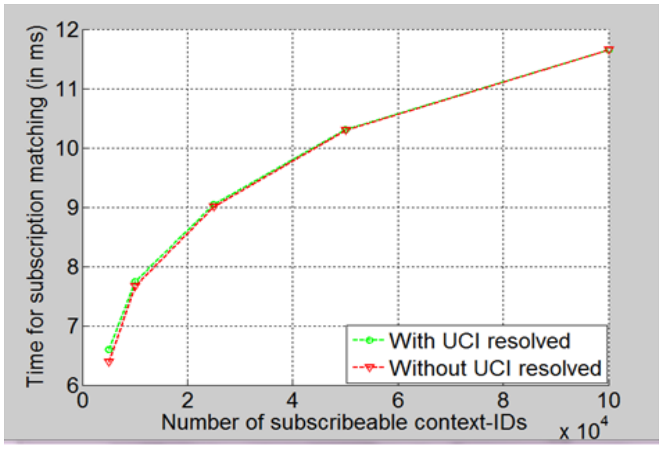

Fig. 14. Subscription matching with and without resolved UCI

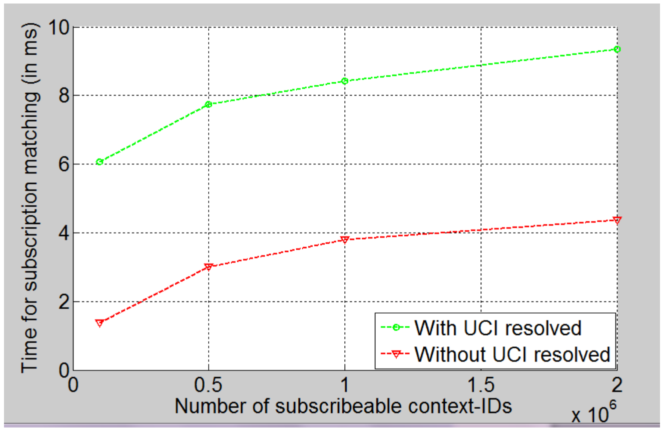

Fig. 15. ^{i}th Subscription matching with and without resolved UCI

Fig. 15 exhibits this. The figure shows the subscription matching from 100K to 2 million. Both results i.e. for both with and without UCI resolved qualitatively reveals similar performance.

However, without UCI resolved clearly outperforms other approach. The improvement percentage is significant. It betters the UCI resolving by 338% and 114% respectively for 100K and 2 million context-IDs. However, it leads to another research question if we ignore the UCI resolving then how do other sinks resolve the context-IDs? This could be done by employing adaptability and awareness in MediaSense which is part of our future work.

F. Dynamic MediaSense PubSub

The previous evaluations have been explored for static scenario which means it did not consider the dynamic environment. This sub-section will examine if MediaSense can fulfill the demand of crowdsourcing dynamism. The current MediaSense allows a UCI to be updated and deleted, however, since the MediaSense had been modified to fit into logical-clustering concept, therefore, the MediaSense has been further extended to adapt to crowdsourcing dynamism. The extended MediaSense now can be used to insert and delete any context-ID anytime. The remainder of this sub-section examines the MediaSense platform's performance for context-IDs insertion and deletion scenarios.

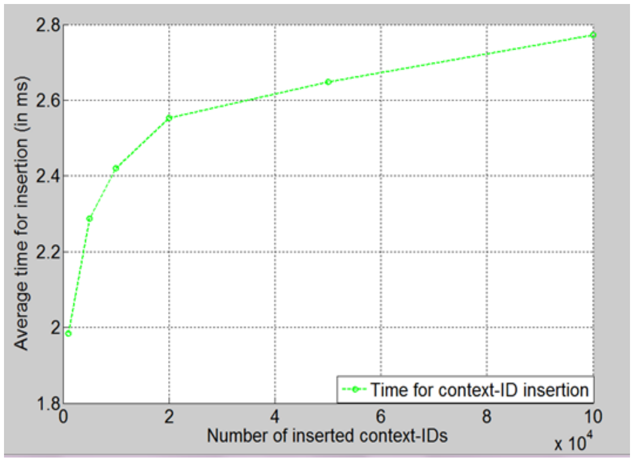

Fig. 16. Average time for context-ID insertion (I)

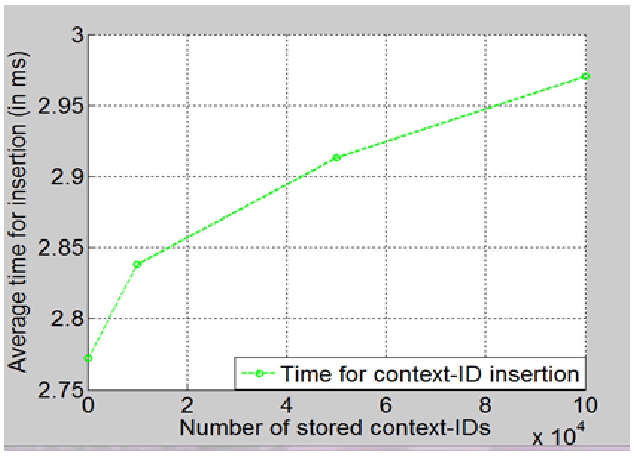

Fig. 17. Average time for context-ID insertion (II)

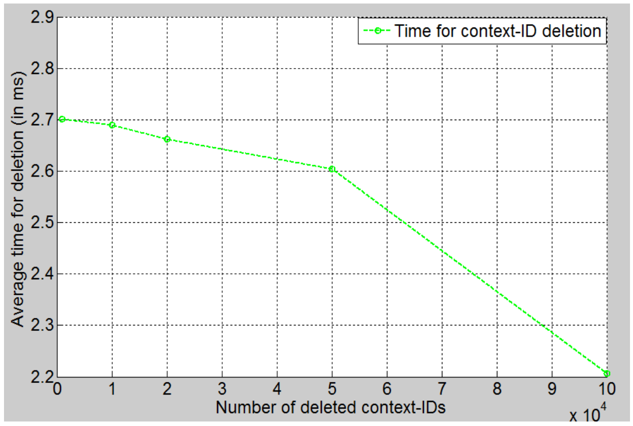

Fig. 18. Average time for context-ID deletion

Fig. 16 shows the context-ID insertion scenario for an already resolved UCI. As expected, the time for insertion increases with the increased number of context-ID. When the number of context-ID is increased from 1K to, average time for context-ID insertion is increased by 40%. The increase is not substantial compared to increase in number which is a 9900% upsurge. More importantly and perhaps significantly, this context-ID insertion follows a specific pattern for most cases.

For example: when the number of context-ID is increased from 5K to 10K the average time for insertion increases by 6%. The same goes true for 10K to 20K increases and for 50K to 100K. Therefore, we can conclude that a 100% increase in context-ID insertion would employ about 5% increases in time (see table V). This phenomenon could be very significant given that in dynamic real-time crowdsourcing it is always of great advantageous to predict the outcome beforehand. Therefore, with this pattern we can always predict the time required for context-ID insertions. Fig. 16 has been evaluated with very small stored context-ID, and in fig. 17 we further investigate if already stored context-ID for a UCI has any impact on average context-ID insertion. Thus we increase the number of stored context-ID in a UCI from 1 to 100K and the average time for context-ID insertion varies merely by around 3% and varies by just 7% when number of stored context-ID in a UCI increased from 1 to 100K. These numbers are very minimal compared to the increase in stored context-ID and does not offer a bottleneck for context-ID insertion.

Fig. 18 shows the context-ID deletion. This result is very surprising for us and it was totally unexpected. Our assumption was that average time for deletion of context-ID would grow with the increase of number of context-ID. Surprisingly, the average time decreases when number of context-ID to be deleted increases. However, if we closely investigate and look at the fig. 18 then we find out the time decrease is very minimal. The decrease is almost negligible when context-ID to be deleted increased from 1K to 50K (only 4%) and the rate is just 22% when context-ID to be deleted increased from 1K to 100K. This assures that MediaSense does not consume too much time to delete context-IDs.

This result is indeed beneficial for dynamic crowdsourcing as we want to acquire outcome faster in real-time.

TABLE V. INSERTION TIME % INCREASE

# of context-IDs increase	1K to 5K	10K to 20K	20K to 50K	50K to 100K	1K to 100K
% increase in average time for insertion	15	6	4	5	40

G. Prediction in MediaSense evaluation

In the above results, it has been observed in many scenarios that the results tend to follow a specific pattern. For example, it has been revealed by fig. 11, 14 & 15 that subscription matching grows linearly and so does the memory growth as observed by fig. 13 and table IV.

Therefore, the objective of this sub-section is to examine and propose some formulas where it can be possible to predict the outcome of the result. Since the real-time crowdsourcing is dynamic and it is imperative that the system is able to pre-determine the outcome. This intelligence in the MediaSense system would give us flexibility in terms of predicting such as time for subscription matching, memory occupation, etc. Table VI portrays the published time percentage increase when the number of context-IDs is increased. The observation indicates that published time increases between 4% – 6% for a 100% increase in the context-IDs size. And if we further analyze table VII we observe that this increase for published time follow a specific pattern. For example, for each 100% increase published time increases by about 5±1%. Even when we have 400% increases then MediaSense demonstrates around 16% - 18% increase. Hence, analyzing the above results the following formula for MediaSense published time increase can be written:

$$P\_T_i = \left((5 \pm 1)\ .\ P\_I_f\right)\% \ldots \ldots \ldots (1)$$

Where P_T_i is the published time increase and P_I_f is the percentage increase factor (for example, for a 100% increase P_I_f would be 1 and for a 400% increase P_I_f would be 4). Although by using eq. 1, it might not be always possible to predict exact published time increase, however, we can at least predict nearest value. As for subscription matching table VIII indicates that it varies always. This is understandable from the fact that while subscribing for a context-ID, MediaSense battles with bandwidth while resolving UCI, and it might not provide any stable equation. Nevertheless, we can at least provide an equation which can provide us a near optimal value for subscription matching. The equation can be written as:

$$S\_M_i = (15 \pm 5)\% \qquad \ldots \ldots \ldots \quad (2)$$

S_M_i is the subscription matching increase. Eqn. 2 is true only when each published context-ID is matched, but as for ^{i}th context-ID subscription macthing from i-size of the context-ID, the subscription matching increases by about 10% in most cases as indicated by table IX.

TABLE VI. PUBLISHED TIME % INCREASE (I)

# of context-IDs increase	**1K to 2K**	**5K to 10K**	**10K to 20K**	**20K to 30K**	**25K to 50K**	**50K to 100K**	**100K to 200K**
% increase in published time	6	6	5	4	4	4	6

TABLE VII. PUBLISHED TIME % INCREASE (II)

# of context-IDs increase	**1K to 5K**	**2K to 10K**	**10K to 50K**	**25K to 100K**
% increase in published time	18	17	16	12

TABLE VIII. SUBSCRIPTION MATCHING % INCREASE (I)

# of context-IDs increase	**1K to 5K**	**2K to 5K**	**5K to 10K**	**10K to 25K**	**25K to 50K**	**50K to 100K**
% increase in subscription matching	15	20	18	19	15	14

TABLE IX. SUBSCRIPTION MATCHING % INCREASE (II)

# of context-IDs increase	**100K to 200K**	**250K to 500K**	**500K to 1 m**	**1 m to 2 m**
% increase in subscription matching	10	9	9	11

It is also possible to predict the memory usage in MediaSense. This can be seen from the fig. 13 and table IV. The memory usage grows linearly and minimally. MediaSense memory usage corresponds to the following equation:

$$M_u = 0.0381 \,.\, N_{C\_id} \text{ (KB) where } N_{C\_id} \geq 5000 \ldots\ldots\ldots (3)$$

Where, M_u is the memory usage and N_{C_id} is the total number of context-ID to be published.

As mentioned earlier that one of the objectives of this paper is to examine if MediaSense remains stable when one of the physical sinks down, according to our finding it does remain stable (the results are not shown here due to page limitation).

From the above results, it is clear MediaSense can adjust to the dynamic nature of crowdsourcing environment and fulfill the mentioned demand without any performance degradation. Moreover, it is also possible to predict the outcome of MediaSense PubSub result which makes MediaSense more attractive as a PubSub model.

VI. CONCLUSION

The growing popularity of crowdsourcing in pervasive computing gives rise to many challenges. Sharing context information in real-time is one of them for example in logical-clustering scenario. The challenge of sharing context information is unraveled by employing MediaSense as PubSub model. MediaSense demonstrated very efficient performance for the PubSub purpose and it performs better than existing PubSub models and requires only 9.59 ms to match two-millionth published context-ID, furthermore the memory requirement is very low. However, the results are analyzed only for static environment. The contribution of this extended paper begins with extending MediaSense to counter the dynamic nature of logical-clustering for crowdsourcing. The paper first proposes a solution for reducing the delay to subscription matching. The solution works very well for ^{i}th item subscription matching, however, when each published item is subscribed then the solution does not offer any improvement. Nevertheless, for the ^{i}th item case the new solution improves by 114% for two-millionth published context-ID which could be hugely significant in dynamic crowdsourcing. However, this solution brings forth a new research question: if we ignore the UCI resolving then how do other sinks resolve the context-IDs? This could be countered by employing adaptability and awareness in MediaSense which is part of our future work.

As for updating published context-IDs i.e. inserting or deleting context-IDs from an existing UCI. The result shows average time for insertion is just 5% for 100% increase in context-IDs. The deletion of context-ID demonstrated a surprising behavior, while deletion time was expected to rise with the escalation of context-ID but the result indicated the opposite. In addition, based on the acquired results few formulas have been presented to predict the outcome for publish and subscribe context-IDs time and for memory usage. The formulas could be very significant in dynamic logical-clustering since it would help to regulate the outcome beforehand.

Although MediaSense did live up to its expectation as scalable PubSub model for both static and dynamic environments but its viability can be further examined. For example: adaptability and awareness in MediaSense; to have prior knowledge of UCI before resolving; and how it will perform on devices with limited computational capabilities. Crowdsourcing heavily involves mobile devices; therefore MediaSense's performance on mobile devices will also be explored. Thus the mobility, energy (e.g. on android devices) issues of MediaSense along with performance in devices with limited computational capabilities (such as on raspberry pi) can be examined.

ACKNOWLEDGMENT

The work is partially supported by funding from the European Union FP7 MobiS project. We would also like to thank Mr. Jamie Walters and Mr. Johan Eliasson for their feedback about extending MediaSense.

REFERENCES

[1] [online] 5G Radio Access, Research and Vision: http://www.ericsson.com/res/docs/whitepapers/wp-5g.pdf [Last Accessed: 08-February-2014]

[2] T. Kanter, S. Forsström, V. Kardeby, J. Walters, U. Jennehag and P. Österberg., “MediaSense – an Internet of Things Platform for Scalable and Decentralized Context Sharing and Control,” In: ICDT 2012,, The Seventh International Conference on Digital Telecommunications, pp. 27-32, April 2012.

[3] [online] MediaSense | The Internet of Things Platform, http://www.mediasense.se/ [Last Accessed: 08-February-2014]

[4] G. Cugola and H. Jacobsen, “Using Publish/Subscribe Middleware for Mobile Systems”, In ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, pp.25–33, October 2002.

[5] I. P. Zarko, A. Antonic and K. Pripužic, “Publish/subscribe middleware for energy-efficient mobile crowdsensing". In Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication (UbiComp '13 Adjunct). Zurich, pp. 1099-1110, September 2013.

[6] J. Pereira, F. Fabret, _F. Llirbat, and D. Shasha, “Efficient matching for web-based publish/subscribe systems,” 7th International Conference, CoopIS 2000, Eilat, Israel, September 6-8, 2000.

[7] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K A. Ross, and D. Shasha, “Filtering algorithms and implementation for very fast publish/subscribe systems”, In Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD '01), 2001

[8] Ashayer, G.; Leung, H.K.Y.; Jacobsen, H.-A., "Predicate matching and subscription matching in Publish/Subscribe systems," Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on , vol., no., pp.539,546, 2002

[9] E. Fidler. PADRES: A Distributed Content-Based Publish/Subscribe System. PhD thesis, University of Toronto, 2006.

[10] F. Zambonelli, “Pervasive urban crowdsourcing: Visions and Challenges”, 2011 IEEE (PERCOM Workshops), pp. 578-583, March 2011.

[11] M. Demirbas, M.A. Bayir, C.G. Akcora, Y.S. Yilmaz, H. Ferhatosmanoglu, "Crowd-sourced sensing and collaboration using twitter," World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a , vol., no., pp.1-9, June 2010.

[12] M.K. Boulos, B. Resch, D.N. Crowley, J.G. Breslin, G. Sohn, R. Burtner, W.A, Pike, E. Jezierski, K.Y.S Chuang, “Crowdsourcing, citizen sensing and sensor web technologies for public and environmental health surveillance and crisis management: trends, OGC standards and application examples”. International journal of health geographics, 10(1), 67, 2011.

[13] Sheth, A., "Citizen Sensing, Social Signals, and Enriching Human Experience," Internet Computing, IEEE , vol.13, no.4, pp.87,92, July-Aug. 2009.

[14] R. Rahmani, H. Rahman, and T. Kanter, “Context-Based Logical Clustering of Flow-Sensors - Exploiting HyperFlow and Hierarchical DHTs”, In Proceeding(s) of 4th International Conference on Next Generation Information Technology, 2013 ICNIT, June 2013.

[15] R. Rahmani, H. Rahman, and T. Kanter, “On Performance of Logical-Clustering of Flow-Sensors”, The International Journal of Computer Science Issues (IJCSI), Vol. 10, Issue 5, No 2, September 2013.

[16] A. Tootoonchian, Y. Ganjali, “HyperFlow: A Distributed Control Plane for OpenFlow”, Proceedings of the 2010 Internet Network, 2010.

[17] Dey AK (2001), “Understanding and using context. Personal and Ubiquitous Computing", 5: 20–24.

[18] H. Rahman , R. Rahmani, and T. Kanter, “Enabling Scalable Publish/Subscribe for Logical-Clustering in Crowdsourcing via MediaSense”, IEEE Science and Information Conference 2014, August 27-29, 2014, London, UK

[19] A. Zaslavsky, “Adaptibility and Interfaces: Key to Efficient Pervasive Computing”, NSF Workshop series on Context-AwareMobile Database Management, Brown University, Providence, 24-25 January, 2002

[20] M. Miraoui, C. Tadj and C. b. Amar, “Dynamic Context-Aware Service Adaptation in a Pervasive Computing System”, IEEE Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2009