Supporting Self-Organization with Logical-Clustering Towards Autonomic Management of Internet-of-Things

Hasibur Rahman*, Theo Kanter, Rahim Rahmani
Department of Computer and Systems Sciences (DSV)
Stockholm University
Nod Building, SE-164 55 Kista, Sweden

***Abstract*—One of the challenges for autonomic management in Future Internet is to bring about self-organization in a rapidly changing environment and enable participating nodes to be aware and respond to changes. The massive number of participating nodes in Internet-of-Things calls for a new approach in regard of autonomic management with dynamic self-organization and enabling awareness to context information changes in the nodes themselves. To this end, we present new algorithms to enable self-organization with logical-clustering, the goal of which is to ensure that logical-clustering evolves correctly in the dynamic environment. The focus of these algorithms is to structure logical-clustering topology in an organized way with minimal intervention from outside sources. The correctness of the proposed algorithm is demonstrated on a scalable IoT platform, MediaSense. Our algorithms sanction 10 nodes to organize themselves per second and afford high accuracy of nodes discovery. Finally, we outline future research challenges towards autonomic management of IoT.**

Keywords—autonomic management; Future Internet; Internet-of-Things; self-organization; logical-clustering; MediaSense

I. Introduction

Research towards Future Internet mandates exploring new challenges. Autonomic management has been around for over a decade since IBM coined this concept [1]. One of the requirements of autonomic management in Future Internet is to bring about self-organization [2, 3]. Moreover, we are about to see a paradigm shift from Internet-of-Things (IoT) to Internet-of-Everything (IoE) [4, 5]. The goal of which is to integrate people, process, data (context information) and things [5] in the Connected Society. Autonomic management was not part of early IoT, however, recently there is a shift in research activities which aims to tie these two [4]. This corresponds to massive participation of nodes in IoT, for example, 212 billion things are expected to be connected to IoT by 2020 [5]. The number of connected devices may even upsurge to 500 billion [6]. This massive immersion within IoT networking mandates to comprehend dynamism [7, 8]. One of the challenges in IoT would be to adapt to fast varying environment and be aware of any changes. The key to unravel these challenges is to organize a system such that it can respond to changing environment and stabilizes the system in situations uncalled for. For example, network connectivity, bandwidth, insertion and deletion of information, joining and leaving of a node/device, etc. are some of the changes that are expected in IoT [5, 8]. Any distributed system requires countering the changes in an organized way; however, most of these changes are not predictable. Therefore, it is imperative that the system organizes itself in such cases. This infers that a system (part of IoT) is desirable to be self-organized leaving outside sources mostly out of the loop.

The self-organization phenomenon exists in wide range of disciplines extending from physics to biology [9, 10]. It has also attracted attention from computer science and is now a very active research area [10]. Some case studies for self-organization in computer science have been presented in [9] which are inspired from the nature. This reflects the vision of autonomic computing that was coined by IBM [1]. They envisioned automatic computing as a grand challenge and outlined four aspects to be the core of automatic computing such as *self-configuration, self-optimization, self-healing, and self-protection* [1]. The requirement of these *self-** capabilities in massively distributed systems have been further discussed in [11]. The definition of self-organization varies in different disciplines befitting the respective goals and criteria. In general, self-organization can be considered as a system which organizes itself automatically i.e. without any intervention from outside sources [9, 12]. However, keeping outside source completely out of the loop is still a research challenge. Even the vision of autonomic computing states: "*a system should organize itself according to high-level objectives, and will collect and aggregate information to support decisions by human administrators*" [1]. This has further been outlined as "*Put simply, the autonomic paradigm seeks to reduce the requirement for human intervention in the management process through the use of one or more control loops that continuously reconfigure the system to keep its behavior within desired bounds*" [13].

Franco and Omer in "*IEEE Transaction special issue: Self-Organization in Distributed Systems Engineering*" have further stressed that self-organizing systems shall leave human mostly out of the loop [14]. They have wisely used the word "*mostly*", because although it is desirable but currently it is impractical to leave outside sources (e.g. human as an administrator- these will be used interchangeably in rest of the paper) completely out of the loop. This implies that a system should execute its tasks even when there is minimal or no support at all from outside source. But the system should be able to interact with outside source and run a periodic algorithm to rectify errors and make system aware and learned about faults. Thus, the system will be able to evolve itself next time it encounters similar error and, thereby, reducing the outside intervention as much as possible. Therefore, self-organization can be defined as a system that "*evolves correctly, adapts to dynamic situations, stabilizes itself in unpredicted situations, and pre-protects itself against probable attacks*".

In light of above, this paper aims to design and develop new algorithms to empower self-organization. The algorithms will be specifically targeted at logical-clustering approach. Logical-clustering efficiently filters out similar context information from distributed sources [15]. MediaSense, a context sharing Internet-of-Things (IoT) platform [16], has been employed to disseminate the clustering identity (context-ID) as a PubSub model for logical-clustering [17]. Results showed that MediaSense is viable for scalable context-ID distribution. It has been further demonstrated in [8] that MediaSense can counter the fast varying environment i.e. dynamism efficiently as well. One of the focuses of this paper is to address "*how can logical-clustering organize itself and evolves according to the requirement and manages in dynamic unpredictable circumstances?".* This paper is particularly interested in automatic and periodic insertion and deletion of context-IDs; self-configuration (automatic seamless integration of participating nodes) and self- healing of sinks (nodes and sinks are used interchangeably throughout this paper); self-optimization of both context-IDs and sinks etc. The correctness of newly designed and developed algorithms will be proven on MediaSense. As it has been already proven that MediaSense can easily counter scalable distribution of context information and it can support dynamism, therefore, it is worthy that we develop self-organization algorithms using MediaSense. This will also contribute in structuring logical-clustering topology in an organized way.

The remainder of the paper is structured as follows: section II presents the motivation, section III revisits the state of the art autonomic computing, section IV demonstrates our approach while section V demonstrates the evaluation of the work, section VI outlines future research challenges, and finally section VI concludes the paper.

II. MOTIVATION

The idea of autonomic computing has been around over a decade now. Since then, there have been several proposals for self-organizing systems. Our goals in this paper are: to support self-organization with the logical-clustering and to propose algorithms which should contribute towards an autonomic IoT management architecture. The aim is to bring about self-organization to the logical-clustering approach. The overall objectives are to ensure that logical-clustering evolves correctly in dynamic and unpredictable situations, and structures itself in an organized manner. As logical-clustering implies that context-ID (identification of cluster) is created as soon as a cluster is formed and depending on the requirement (policy) context-IDs should be deleted after a specific time. The system needs to be aware of this i.e. *self-optimized*. Each sink in logical-clustering needs to fetch context-IDs from other sink(s). The system should be configured in such a way that sink(s) can fetch CI from other sinks automatically and periodically. This also implies that sink should discover other available sinks and advertises itself so that seamless integration to the system is ensured. This mandates establishing *self-configuration* as outlined by autonomic computing vision. Sink in logical-clustering topology is considered to be fixed. And fixed sinks are considered not removable; however, sinks can be down for example due to no Internet connectivity, power failure, etc. Hence, it is imperative the system should re-configure whenever it is up again. Re-configuration also includes retrieving old data and synchronizing immediately with other sinks automatically. This is branded as *self-healing*. Therefore, our particular focus is limited to design and develop algorithms for *self-configuration, self-optimization and self-healing*. The algorithms will be evaluated on a proven, scalable, and adaptable IoT platform MediaSense.

III. AUTONOMIC MANAGEMENT

This section briefly introduces the state of the art autonomic computing and further reviews each of the fundamental aspects of self-organization system.

A. Revisting the state-of-the-art

A self-organized system deemed to be dynamic and each organized element constitutes overall system, thus, the resulting overall system becomes complex and its behavior becomes unpredictable [11]. According to many research papers, as mentioned earlier, a self-organized system should acquire its organized characteristics without intervention from outside sources [9, 12]; however, some other researchers mention that outside sources should be kept out of the loop as much as possible [13, 14]. The vision of autonomic computing would only be complete through acquired knowledge i.e. awareness from each organized system [1, 13]. Therefore, the overall system should evolve gradually and eventually keep outside sources out of the loop mostly- if not completely.

Self-organized system can have several self-* capabilities [11]. All these self-* capabilities can be summed into the four aspects of autonomic computing i.e. self-configuration, self-optimization, self-healing, and self-protection. Table I further illustrates this [1].

These aspects need to be managed by a manager which will enable interaction with other organized elements and/or outside source (e.g. human administrator). Manager will enable analyzing, planning and executing the high-level objectives (policies) set by outside sources and it will further allow an organized element to interact with another organized element inside the system. Fig. 1 shows how a manager can achieve this (the figure is redrawn from [1]).

TABLE I. SELF-* ASPECTS

Aspects	Capabilities
Self-Configuration	A new node joins; advertises itself and discovers other nodes Adjusts and integrates automatically and seamlessly according to the high-level policies
Self-Optimization	A node should be able to optimize the local operation parameters according to global objectives Learning and altering objectives adpated by others Should be able to adjust in case of policy conflict
Self-Healing	Re-configurations of the nodes in case of failures Healing for configuration and optimization
Self-Protection	A node should be able to protect itself from outside and undesirable attack

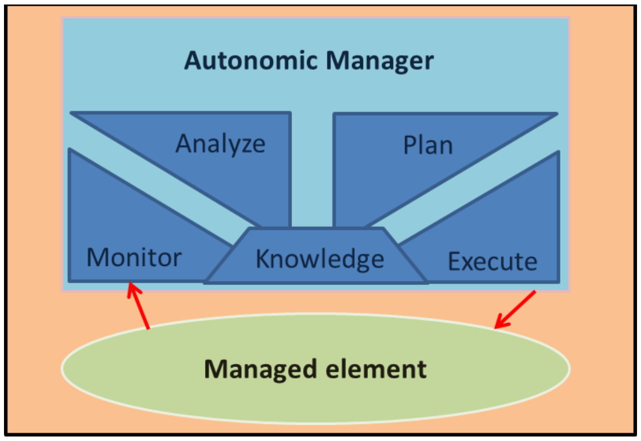

Fig. 1. Structure of an autonomic element [1]

An element joins the system and the autonomic manager, first of all, analyzes the element, then plans and finally executes based on the objectives required by the overall system. This behavior of joining and execution follows the trend of a control loop (the red arrows correspond to this). That means autonomic computing relates to a control loop which is executed based on the specified policies. Fig. 2 further demonstrates this. The policies (objectives) are responsible for implementing the self-* capabilities. At the beginning, these policies are integrated with the system; and as system evolves and encounters new problems, new polices are added. However, this requires learning i.e. awareness (through acquiring knowledge) from each organized element.

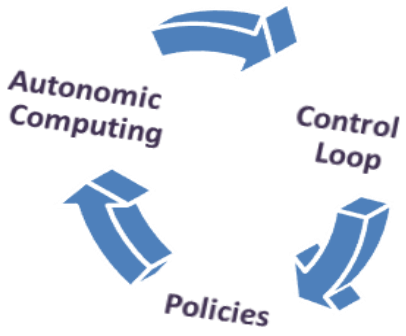

Fig. 2. Life cycle of autonomic computing

IV. OUR APPROACH

Our focus in this paper is not to redefine the self-* aspects outlined in previous section; rather we focus to design and develop algorithms that would implement these aspects. A limitation of this paper is that it will not explore the self-protection aspect. In this section, we will explain the proposed algorithms for other three self-* aspects targeted at logical-clustering. Moreover, the correctness of the algorithms will be demonstrated on a p2p based IoT platform- MediaSense.

MediaSense disseminates context information using Distributed Context eXchange Protocol (DCXP) where each entity is registered as Universal Context Identifier (UCI) and other entities can resolve the UCI [16]. It utilizes rendezvous host i.e. a bootstrapping node to initiate communication between entities. Any entity plans to use MediaSense must use the primitive functionalities defined in the MediaSense Platform. Our proposal is to utilize MediaSense Platform as controller i.e. autonomic manager and each of the designed three self-* aspects has been added as extended primitive functions for MediaSense. Fig. 3 shows how MediaSense Platform can be utilized as an autonomic manager. In the following sub-sections, we will describe how self-organization can be supported in logical-clustering by employing the autonomic computing concept. We have shown in [17], how MediaSense can be utilized for logical-clustering concept.

A. *Self-Organization Support for Logical-Clustering*

Logical-clustering consists of logical-sinks [15] and sinks are responsible for controlling (creation, insertion, deletion) the clusters. However, sinks require discovering, co-operating with other available sinks, and it should also maintain itself. In other words, it should be organized and organization should be done with minimum support from outside sources. Since logical-clustering involves real-time communication, it is imperative that it evolves correctly, automatically in real-time.

Fig. 3 shows how we want to employ autonomic computing concept i.e. self-organization in logical-clustering. As in any other system, a sink first needs to join which is the first operation in the control loop.

An autonomic manager (i.e. MediaSense Platform in our case) will analyze the policy associated with the join request. During this operation, MediaSense platform will implement the self-* algorithms. The sink will be adapted based on the outcome of the algorithms after which sink would be ready for execution. The platform should be aware all of these operations as indicated in fig. 3. Actions showed in Fig. 3 can be summed up as following:

- *A sink joins*
- *Platform analyzes the policies i.e. evaluates the self-* algorithms*
- *Platform further adapts the sink based on the policies' outcome*
- *Sink is ready for execution (after this stage sink is said to be an organized sink)*
- *Platform has the awareness of all these actions*

Fig. 4 shows how the concept can contribute towards autonomic management of IoT. An entity i.e. a thing in IoT will be connected to the scalable MediaSense Platform and will be managed by the self-* algorithms automatically.

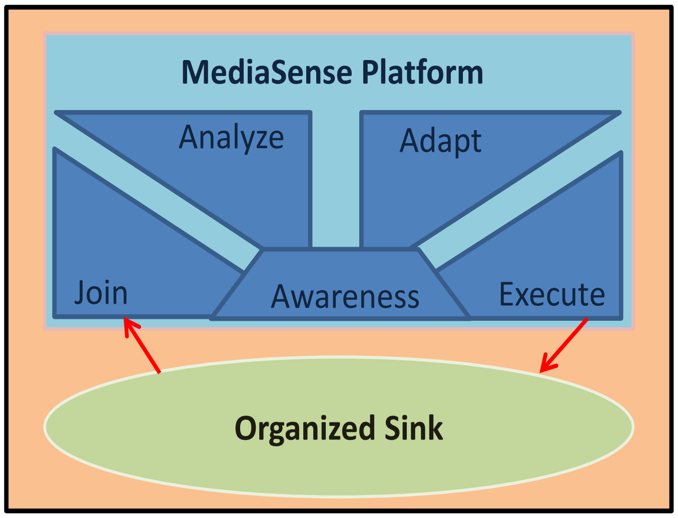

Fig. 3. Supporting self-organization with logical-clustering

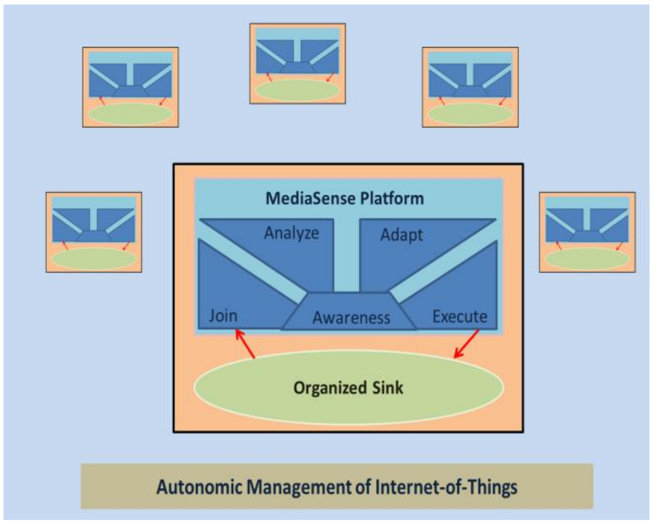

Fig. 4. Autonomic management of IoT

B. *Self-Configuration*

Self-configuration implies that whenever a new element/node joins a system; it should be able to advertise itself and discover other available nodes or let other discover inside the system. The goal of this self-* aspect is to ensure automatic and seamless integration to the joined system.

To achieve the aforementioned goal, we have employed publish/subscribe (PubSub) model that was shown in [17]. MediaSense utilizes a bootstrapping node, as the case with other distributed p2p systems, which needs to be initiated before any communication can take place. Our idea is to define a global publisher e.g. global_uci in MediaSense and each node whenever tries to join the MediaSense Platform automatically subscribes to the publisher without knowing which node holds the global_uci. This newly joined node is added to the global_uci and existing nodes are automatically discovered after a certain time. Self-configuration involves several steps. The steps are listed below:

- *Sink join*
- *Configure global_uci (at beginning on MediaSense bootstrap node)*
- *Sink configuration*
- *Check for self-healing i.e. reconfiguration*
- *Discover other sinks*

Each of the algorithms is depicted in the following figures.

```
Algorithm registering global_uci

begin
  initiate bootstrap_MediaSense
  //the followings are added as extension to current MediaSense
  if global_uci exists
    renew global_uci
  elseif
    register global_uci (based on PubSub's publisher algorithm)
  endif
  //the above is extension
  run bootstrap_MediaSense
  start P2P Communication on the bootstrap IP address and bootport
end
```

Fig. 5. Algorithm for configuring a global_uci as publisher

Fig. 5 depicts the algorithm for configuring global_uci which is exploited as a publisher. Whenever MediaSense is bootstrapped, global_uci is either merged or renewed based on the requirement (different situation might require different policy). As for this approach, global_uci is renewed meaning that a fresh copy of global_uci is guaranteed. Global_uci stores all the available sinks as published items and each node - that joins a MediaSense platform and invokes the MediaSense Platform's selfConfig primitive function (fig. 7)- is automatically subscribes to the global_uci.

```
Algorithm sink_join

begin
  create an instance of MediaSensePlatform
  initialize platform with network settings (Bootstrap IP address, bootport, local port)
  while MediaSense is bootstrapped
    declare UCI (i.e. identity of the node)
    invoke MediaSensePlatform's joinUCI
     if MediaSensePlatform's selfHealing is true
      if the UCI is not listed on global_uci
       publish on global_uci by invoking MediaSensePlatform's config method
       return the UCI's current configuration status
      endif
     elseif
      register the UCI on MediaSense platform
      publish on global_uci by invoking MediaSensePlatform's config function
     endif
    invoke MediaSensePlatform's selfConfiguration
    synchronizes with the existing UCIs after every T seconds
  endwhile
end
```

Fig. 6. Algorithm for sink joining

As mentioned earlier, each sink that joins the MediaSense Platform is identified as UCI. Therefore, every time a new sink joins, it fetches all the available UCIs i.e. sinks. Fig. 6 shows the algorithm for sink joining. In this procedure, first a sink must initiate the MedaSense Platform along with the network settings to use its functionalities.

An identity for this sink is declared which is then used to join the platform. When the joinUCI function is called- it first checks if UCI already exists (i.e. registered with MediaSense) with this particular sink- if that check returns true then sink is reconfigured with previous configuration (see fig. 8), if that check returns false meaning no identity duplication from this sink then the UCI gets registered on MediaSense platform and gets published on the global_uci which allows other sinks to discover node. The joined sink then invokes the selfConfig function of MediaSense Platform which enables automatic discovery of other available nodes on MediaSense every *T* seconds.

The interval is implementation dependent- by default set to 20 seconds. The interval is an open research issue for battery powered devices, rechargeable devices; but not an issue for AC powered sources. Discovery of sinks involves the idea that of MediaSense's subscription [17]. In this procedure, first the global_uci is resolved based on MediaSense's subscription algorithm. When the global_uci is resolved, each of the stored UCIs is read and current UCI is added to the global_uci. After adding the current UCI, global_uci is updated. This implies that global_uci is either merged or renewed based on the configuration requirement. Subscription is matched whenever this function- implementing this algorithm- is invoked.

```
Algorithm sink_discovery

begin
   resolve global_uci (based on PubSub's subscription algorithm)
   read the subscribeable UCIs
   store the UCI to the global_uci
   update global_uci
   merge or renew depending on the configuration
    subscription is synchronized whenever this method is invoked
end
```

Fig. 7. Algorithm for sink discovery

```
I. Algorithm sink_duplication_check

begin
  resolves the UCI
  fetches associated information
  if the sink id is found with the fetched
information
   returns true
  elseif
   returns false
  endif
end


II. Algorithm sink_reconfigure

begin
  resolves the UCI
  fetches associated information
  if uci exists in the current MediaSense
   reconfigure the node and fetch previously
existing data
  elseif
   start uci registration
   while register
     invoke selfConfiguration
   endwhile
  endif
end
```

Fig. 8. Algorithm for sink reconfiguration

C. Self-Healing

Self-healing refers to the reconfiguration of a node in case of failure. This also corresponds to the healing for the other two self-* algorithms. The self-healing function that is called when a node joins checks if UCI is already exists. Moreover, this algorithm should ensure when a sink is down for some reason, it should be able to reconfigure with previous configuration whenever it is up again. Fig. 8 shows the algorithm. The first part of algorithm requires two parameters i.e. the UCI and sink-ID; and returns a boolean value. Sink ID is obtained by calling MediaSense Platform's getHostID function. This procedure begins by resolving the UCI and it fetches the associated information with this UCI. If the host ID is found in the fetched information then this function returns true otherwise a false value is returned. Second part of self-healing algorithm also requires resolving the UCI and its associated information are fetched. After that, if UCI exists on MediaSense then the algorithm reconfigures the sink with previously existing data. But if the UCI is nonexistent then UCI is registered on the Platform and selfConfiguration is invoked so that the UCI being registered can execute the self-configuration algorithm as discussed in previous sub-section.

```
Algorithm sink_optimization

I. Insert context-ID

begin
  resolves the UCI
  checks for new context-IDs
  if new context-IDs are found
    insert new context-IDs in the UCI and
adjusts seamlessly with existing context-IDs
    invoke Insert ContextID Policies
  endif
 end

II. Delete context-ID

begin
  resolves the UCI
  checks for context-IDs to be deleted
  if context-IDs need to deleted
    delete existing context-IDs in the UCI
and adjusts seamlessly with existing context-
IDs
    invoke Delete ContextID Policies
     if single context-ID with a TTL
      delete context-ID
     eleseif
       delete context-IDs
     endif
  endif
end
```

Fig. 9. Algorithm for sink optimization

D. Self-Optimization

Self-optimization implies that a node should optimize itself according to the policies set by manager (in our case MediaSense Platform). This should further ensure that each node performs to its best capability and efficiency. In this paper, we looked into optimizing context-IDs associated with logical-sink. The goal of this algorithm is to support autonomic dynamism in logical-clustering. This means automatic insertion, deletion of context-ID is made possible which enables awareness in the sinks. Each sink creates new context-ID in real-time; therefore, it is imperative that sink optimizes itself by inserting new context-IDs with an interval of *T* seconds. Each sink should also be able to delete context-IDs automatically whenever needed. Moreover, each sink should be aware if a particular context-ID is needed to be deleted after a specified time. Fig. 9 depicts the algorithm. This algorithms also has two main parts i.e. insertion and deletion of context-IDs. In the first part of this procedure, the algorithm first resolves the UCI and checks if there are any new context-IDs to be inserted.

When there are next context-IDs found for insertion, the class that implements the Insert ContextID Policies gets invoked. Depending on the requirement, different policies may be executed. In our case study i.e. logical-clustering, it automatically and periodically inserts the new context-IDs, integrates seamlessly with existing context-IDs and synchronizes with other sinks. As for deleting context-IDs, the procedure checks if a particular context-ID (with a timestamp for time to live (TTL) is attached) is to be deleted or it should delete context-IDs. The procedure executes these and optimizes the sink.

V. Performance Evaluation

This section presents the performance evaluation of supporting self-organization with logical-clustering. Self-* capabilities algorithms have been developed on the MediaSense platform. Each of the self-* aspects has been developed and included as separate package, this can be found under the new package called autonomic in current MediaSense platform. We start first with reporting the effect of incorporating self-organization on MediaSense Platform. Table II and II report this. We have evaluated the performance of joining node on two different networks with different Internet speeds. Measurements are shown in logarithmic scale and in milliseconds; the mean μ and standard deviation σ values are depicted in tables. Results suggest that on both networks, we get almost similar result. In terms of self-organization, we have divided the evaluation into two. In the first part, we do not consider time required for self-healing i.e. duplication check and reconfiguration; and only time required for self-configuration is considered, and the second part includes time required for self-healing (see IV-B). MediaSense incurs a delay if self-* algorithms are employed. This, however, is what is expected of self-* algorithms, reason being a node goes through the life cycle of autonomic computing (see fig. 2 & 3) before completing the joining operation i.e. becoming organized (a managed node). The algorithms demonstrated almost identical performance on both networks.

TABLE II. Node Joining (Network I)

	MediaSense (Current)	**MediaSense (Self-Configuration)**	**MediaSense (Self-Organization)**
μ	1.59	1.69	1.88
σ	0.0522	0.0459	0.0338

TABLE III. Node Joining (Network Ii)

	MediaSense (current)	**MediaSense (Self-Configuration)**	**MediaSense (Self-Organization)**
μ	1.61	1.69	1.91
σ	0.0593	0.0261	0.0260

First operation in the self-organization starts with node(s) joining the autonomic management platform. For this particular evaluation, we consider that nodes are not competing for MediaSense Platform's resources. This means that nodes are executed one after another. The issue of concurrent node(s) joining leads to load-balancing and scheduling issue which is not covered in this paper. Table VII elaborates the necessity for load-balancing and scheduling. Fig. 10 shows the performance of nodes joining the platform. X-axis represents the number of nodes and y-axis represents the processing time. Processing time is shown in logarithmic scale and in seconds here. If we analyze the figure and the processing time in normal scale, we find that each second 10 nodes can join the MediaSense Platform. This could be a starting point for exploring the load-balancing and scheduling issue. However, this result should not be confused with table II and II. Table II and II only reported the operation time require for a single sink joining operation, these did not consider the inside mechanism requires for fully organizing with other nodes (time was shown in logarithmic scale there too).

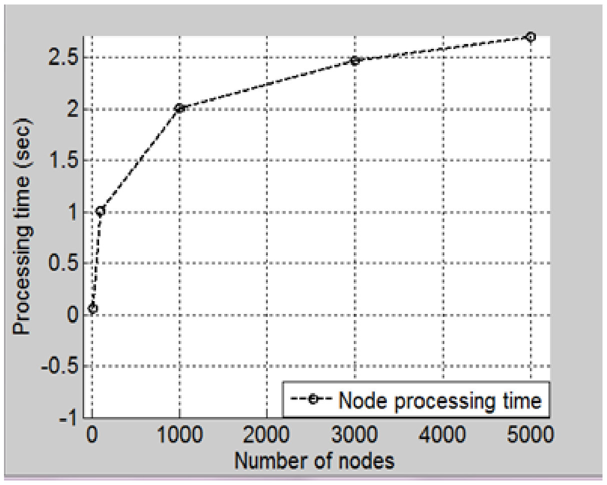

Fig. 10. Processing time for nodes joining

Next we evaluate the self-organization algorithms in terms of discovery of nodes and accuracy of discovery. Fig. 11 shows the discovery of nodes. This implies the time required for a node to discover i.e. synchronize with other nodes. This has been evaluated for both dynamic and stable scenarios. Dynamic means discovery of nodes measured while other nodes are joining simultaneously and stable implies that currently no more nodes are joining the system. This measurement was done when evaluating fig. 10. This algorithm is run after every 20 seconds. Table VI illustrates the algorithm depicted in fig. 7. The stable scenario corresponds to this. The results are different from dynamic scenario where each node goes through the cycle of self-organization and competes for resources; thus incurs delay. The result portrayed in table VI suggest that discovery of nodes while system is stable is very fast. This also corresponds to the MediaSense's PubSub model which also demonstrated fast and efficient result [17]. Hence, we do not discuss this in detail in the paper.

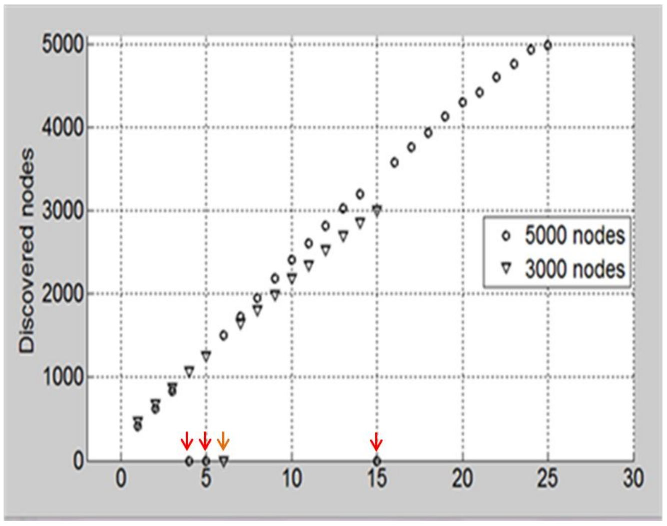

Fig. 11. Discovery of nodes

The above figure depicts nodes' discovery for 5000 and 3000 nodes joining the platform. The mean *μ* for discovery is 197, 184; and most frequent number (*F*) is 205 and 202 respectively for 5000 and 3000 nodes. These values have been calculated based on the differences of the vector elements excluding the first and last element that makes the discovery stable i.e. no more nodes are joining. This reflects that each second (in normal scale) 10 nodes join the platform. Since the interval period *T* was chosen to be 20 seconds, so the mean values are justified. This could further be seen from the fig. 11 that node discovery became stable (all available nodes were discovered) on the 15th (3000 nodes case) and 25th (5000 nodes case) attempt. However, 4990 and 2997 nodes out of 5000 and 3000 nodes could be discovered respectively. This gives a high discovery accuracy of over 99 %. Discovery accuracy can further be seen from table V. Also on the 4th, 5th and 15th attempt (5000 nodes case- red arrows) and on 6th attempt (3000 node case- orange arrow) of self-configuration, the algorithm failed to discover any nodes. This could be because currently the algorithm tries to renew (see fig. 7) the global_uci, this means the old global_uci is deleted and a fresh copy of global_uci is inserted on nodes. And, during the call and renew, some UCIs might not have been synchronized. But the system was able to fetch information immediately in next attempts meaning stability is ensured (further discussed below).

TABLE IV. NODE DISCOVERY (DYNAMIC)

	5000 nodes	**3000 nodes**
μ	197	184
σ	29.79	14.8613
F	205	202

TABLE V. DISCOVERY ACCURACY (SEQUENTIAL)

	5000 nodes	**3000 nodes**	**1000 nodes**
Nodes discovered	4990	2997	1000
Accuracy	99.8 %	99.9 %	100 %

Table V indicates that discovery accuracy is very high and near to 100 % for all three simulated cases, however, these measurements were done when nodes were joining sequentially. We did not consider concurrent nodes joining where nodes would compete to access MediaSense Platform resources. Table VII shows the discovery accuracy for 3000 and 2000 concurrent nodes joining. Arithmetic mean is 1699, 1195 and standard deviation is 110.4513, 97.4664; and the discovery accuracy drops to 56.63 % and 59.75 % respectively. This drop in discovery accuracy further highlights necessity for designing and developing load-balancing and scheduling algorithm.

TABLE VI. DISCOVERY DURATION (STABLE)

	5000 nodes	**3000 nodes**	**1000 nodes**
μ	54.67 ms	47 ms	44 ms
σ	9.76 ms	9.18 ms	9.57 ms

TABLE VII. DISCOVERY ACCURACY (CONCURRENT)

	μ	***σ***	***Accuracy***
3000 nodes	1699	110.4513	56.63 %
2000 nodes	1195	97.4664	59.75 %

Self-healing algorithm implies that each node should be able to heal itself and compensate for failure. The developed algorithms offer duplication check and compensate when a failed (e.g. was down or offline) node rejoins the system. This algorithm was evaluated with 2000 and 3000 already existing nodes trying to rejoin the system concurrently (this was evaluated while measuring for table VII) and each node was successfully checked for duplication. This means duplication check accuracy was 100 % for both cases. This was further confirmed with 1000 existing nodes trying to join one after another i.e. sequentially, this too offered 100 % success rate.

One of the goals of this paper was to make sure logical-clustering evolves correctly and automatically i.e. optimize itself.

Optimization still needs improvement (should be developed upon more policies being explored); as of now this algorithm (along with other two algorithms) allows us to structure the logical-clustering topology. A sink in logical-sink requires synchronizing with other sink(s). Thus far, this was done manually as shown in [8, 17]. Now it is possible to synchronize the logical-sink automatically and periodically. According to the experiments done, the algorithm has successfully achieved automatic dynamism (insertion, deletion) of context-IDs.

A self-organized system is considered complex and its outcome is often unpredictable [11]. Results demonstrated in this paper also reflect this. This can be easily perceived from the fig. 11, table IV, V and VI. Discovery nodes per attempt are not always same (see fig. 11); even the μ and σ do not exhibit similarity for both 3000 and 5000 nodes. However, by perceiving the standard deviation, the deviations are not too fluctuated and remain within reasonable limit.

Network stability and resilience are two of the main components of a system. Our self-organized algorithms were able to structure the logical-clustering topology and the topology was able to evolve correctly without any central point of failure. We have observed in fig. 11 that node failed to discover nodes in few cases, but it was able to stabilize itself immediately. Since global_uci holds the information about each existing sink and this information is accessible to each node subscribed to this global_uci. This makes the system resilient to failure i.e. no central point of failure. When a sink failed or left the network and attempted to rejoin the system, all previous information was fetched immediately. Fig. 11 further confirms this when after failing to discover in few cases, it was able to fetch old and new information simultaneously. Self-healing's 100 % success rate also reflects to such claim of stability and resilience.

VI. Future Research Challenges

The idea of logical-clustering was proposed in [15] and the research reported in this paper is a step forward to fulfilling the based vision. However, the vision of fully functional self-organized logical-clustering still requires some considerable research work. So far, we have not discussed the policies in organizing logical-clustering. In this paper, we have presented a template for achieving self-organization. A fully self-optimized system requires adapting new policies, and optimizing the system accordingly. Moreover, autonomic management of IoT would only be possible through exploring further policies. This mandates exploring the policies that would enable autonomic management of IoT and thereby IoE. Furthermore, integrating polices into the manager mandates a more flexible and concrete manager. This could be achieved by deploying Software-Defined Networking (SDN) concept. The intelligence of SDN would enable to counter the challenges of efficient traffic management, and data and services delivery.

Another fundamental aspect of self-organization i.e. self-protection also need to explored and implemented. Each node should be protected against possible attacks and from getting removed by other node(s). Protections of context-IDs also need to be ensured.

Our focus in this paper was limited to designing and developing a template for self-* aspects, and these self-* aspects need to be adapted based on awareness. This awareness should come from all the stages as depicted in fig. 3. Learning plays an integral part in building the awareness. The managers should be able to learn new policies and create awareness of the learned policies to other nodes. However, we have not addressed the issue of learning in this paper. An approach for learned-manager is still an open issue which can be implemented along with SDN and learned-management system.

Unique identification of each node is another important research issue needs to be addressed. However, uniquely identifying billions nodes is not something easy to implement. Moreover, these identifications should also be easy for humans to remember. For example, context-IDs in logical-clustering should be unique and human should be able to access these context-IDs easily. Therefore, it mandates to define a naming scheme which will ensure unique identification of node in IoT landscape.

Load-balancing and scheduling of autonomic manager is another feature that needs attention.

Heterogeneous interoperability of the system also remains a challenge. IoT heavily involves cross-platform communication, therefore, it is mandated that we look into cross-platform behavior of the system too.

VII. Conclusion

The contribution of this paper by and large lies with designing and developing the self-* aspect capabilities inspired by the autonomic computing concept. In particular, *self-configuration, self-optimization and self-healing* algorithms were designed and developed; and correctness of these algorithms was proven on a scalable and versatile IoT platform MediaSense. MediaSense Platform was employed as what is autonomic manager to autonomic computing. This new algorithms enable logical-clustering to organize automatically and periodically. Each sink in logical-sink now said to be organized and it evolves correctly.

The algorithms sanction 10 nodes to self-organize themselves in each second on the MediaSense Platform, and discovery accuracy is over 99 % when there is no competition for MediaSense Platform's resources. While nodes compete for MediaSense Platform's resources, discovery accuracy is around 60 %. Stable system allows discovering nodes very fast; and duplication check always succeeded. This enables logical-clustering topology to evolve correctly and structure itself. There is no central point of failure, even if bootstrap node fails, other node takes over and stabilizes the system.

Our next step is to explore the challenges mentioned in the future research challenges. SDN would, perhaps, enable us to see fully operational autonomic management of IoT. Autonomic management itself is a grand challenge and it will go through many transitions. IoT would also need to see-off many transitions and finally embrace an operational autonomic IoT- thereby IoE.

We hope that the algorithms depicted in this paper are step forward towards the autonomic management of IoT and could be used as template for further development.

ACKNOWLEDGMENT

The work is partially supported by funding from the European Union FP7 MobiS project. We would also like to thank our colleagues at Immersive Networking Research Group for their feedback that helped in finalizing paper.

REFERENCES

[1] J. 0. Kephart and D. M. Chess, "The vision of autonomic computing",

IEEE Computer Society, 36(1):41-52, 2003.

[2] J. Strassner, S.S. Kim, J. Won-Ki Hong, "The Design of an Autonomic Communication Element to Manage Future Internet Services", Management Enabling the Future Internet for Changing Business and New Computing Services Lecture Notes in Computer Science Volume 5787, 2009, pp 122-132

[3] Siekkinen et al., "Beyond the Future Internet – Requirements of Autonomic Networking Architectures to Address Long Term Future Networking Challenges", IEEE computer society

[4] Wikipedia, "Internet of Things" http://en.wikipedia.org/wiki/Internet_of_Things [Last Accessed: 04-February-2015]

[5] Ruthbea Yesner Clarke, "Smart Cities and the Internet of Everything: The Foundation for Delivering Next-Generation Citizen Services", white paper sponsored by Cisco, October 2013

[6] Ericsson White Paper, "5G Radio Access: Research and Vision", 2014

[7] 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

[8] H. Rahman , R. Rahmani, and T. Kanter, "Realising Dynamism in MediaSense Publish/Subscribe Model for Logical-Clustering in Crowdsourcing", International Journal of Advanced Research in Artificial Intelligence (IJARAI), Vol. 3, No. 11, pp. 49-59, 2014

[9] M. Mamei, R. Menzes, R. Tolksdorf, F. Zambonelli, " Case studies for self-organization in computer science", Journal of System Architecture, pp. 443-460, Vol. 52 (2006), Issues 8-9, Elsevier

[10] Wikipedia, "Self-Organization", http://en.wikipedia.org/wiki/Self-organization [Last Accessed: February 2015]

[11] F. Dressler, "Self-Organization in Massively Distributed Systems – Methods and Techniques"

[12] M. A. Razzaque, S. Dobson, and P. Nixon, "Enhancement of Self-organization in Wireless Networking through a Cross-layer Approach," First Int'l Conf. ADHOCNETS, 2009.

[13] Kephart, J. O. (2005, May). Research challenges of autonomic computing. In Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on (pp. 15-22). IEEE.

[14] F. Zambonelli, O.F. Rana, "Self-Organization in Distributed Systems Engineering: Introduction to Special Issue", IEEE Transactions on Systems, Man, and Cybernetics,Vol. 35, No. 3, 2005

[15] 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

[16] T. Kanter et al., "MediaSense | The Internet of Things Platform", http://www.mediasense.se/ [Last Accessed: 08-February-2014]

[17] 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