Towards an Information Management Overlay for Emerging Networks

L. Mamatas, S. Clayman, M. Charalambides, A. Galis and G. Pavlou

Department of Electronic & Electrical Engineering
University College London
{L.Mamatas, S.Clayman, M.Charalambides, A.Galis, G.Pavlou}@ee.ucl.ac.uk

Abstract— There has recently been an increasing research interest in network management infrastructures that autonomously adapt to the dynamics of the environment. In this paper, we present an information management platform, called the Information Management Overlay (IMO), which is an infrastructure that regulates information flow based on the properties and the state of the network environment. We discuss the design, implementation and optimization issues of this platform and we investigate experimentally how the IMO can adapt to different settings and optimization requirements.

I. INTRODUCTION

A key issue in the management of emerging networks is the development of a common control space, which has autonomic characteristics and enables heterogeneous network technologies, applications, and network elements to interoperate efficiently. Management applications need to be adaptive to a rapidly changing environment with respect to specific network properties, and service or user requirements, as examples. This implies that management applications and network entities should be supported by a platform that collects, processes, and disseminates information characterizing the underlying network. An increased awareness regarding the properties and state of the network can bridge the gap between high-level management goals and the configuration that achieves them. In this paper we consider an infrastructure that manages both information flow and processing within the network as an important stepping-stone towards this objective.

Key design requirements of an information management infrastructure are: (i) information collection from the sources (e.g., network devices), (ii) information processing that produces different information abstractions, (iii) information dissemination to the entities that exploit that information, (iv) information flow optimization with respect to specific requirements of the underlying network (e.g., topology properties) and the high-level management objectives. The design of such a system is challenging because it should exhibit the following properties while imposing a minimum management overhead:

- *self-adaptation to different environment conditions and network properties*, e.g., regulate information flow, information accuracy, or monitoring configuration, accordingly.
- *self-management functionalities*, e.g., have autonomous decision-making capabilities based on high-level objectives.
- *scale efficiently*, at least, up to the level of the administrative domains that are exploiting such information.

It is common that such design approaches (e.g., [1], [2]) have a hierarchical structure and aggregate information using aggregate functions. Consequently, real-time monitoring of network parameters may introduce significant communication overhead, especially for the root-level nodes of the aggregation trees. We argue that information flow should adapt to both the information management requirements and the constraints of the network environment. As a result, nodes of an IMO may dynamically change position in order to adapt to changes in the network environment, one example being: changes in the information collection configuration. This paper investigates the self-adaptation behavior of the IMO infrastructure under different conditions.

In this paper, we pay particular attention to the control requirements of the IMO infrastructure, highlighting that the IMO incorporates policy-based management functionality. A special component of control is the IMO Controller, which is responsible for performing adjustments of the information flow whenever the network environment requires it. We provide examples of performance requirements and show how different optimization strategies can be combined towards the same goal: to fulfill these requirements. A requirement for minimum network overhead may, for example, lead to a different aggregation graph in comparison to a requirement to minimize CPU utilization. We discuss algorithms that are able to self-adapt to the environment. In this context, we evaluate different optimum IMO-node placement algorithms with respect to the optimization requirements.

We have implemented and evaluated the proposed IMO infrastructure under different scenarios, using both real experiments and simulations. Our experiments cover different topology scales, different information collection/aggregation configurations, and information flow adjustments. We investigate how different performance trade-offs (e.g., processing cost vs. communication overhead) are related to the configuration of the IMO. We show results associating information flow optimization algorithms with example performance requirements. The results presented are very promising and call for further investigation.

II.   RELATED WORK

The Information Management Overlay is a basic component of the autonomic management architectural model proposed in the Autonomic Internet (AutoI) project [3]. The AutoI project aims to develop a self-managing resource overlay that spans across heterogeneous networks and supports service mobility, security, and quality of service. The IMO realizes the core functionalities of the Knowledge Plane in an architecture developed by AutoI [4] as part of the Future Internet initiative (www. future-internet.eu). The IMO is a focused-functionality Knowledge Plane that provides management applications with the necessary information for realizing self-management capabilities. It is different from the Knowledge Plane proposed in [5], which is a unified solution that includes cognitive techniques and knowledge management. The IMO decouples information management from other network management functionalities. The latter are considered being part of either the Management plane or the Orchestration plane within, for example, the AutoI architecture [4].

In [6] the authors present specific implementation details of a knowledge plane architecture that consists of a network knowledge plane (NetKP) in the network layer, and of multiple specialized KPs (spec-KPs) on top of that. One of the key differences with our approach is that their architecture is built using static agents, whereas the proposed IMO uses dynamic nodes that may change position dynamically for the purpose of information flow optimization.

Most autonomic computing platforms address fixed networks with sufficient resources [7], [8]. There are a limited number of approaches that focus on dynamic environments, such as MANETS. In [9], the authors propose a context-aware system that performs protocol deployment. The reconfigurable context-sensitive middleware (RCSM) [10] deals with context awareness in mobile devices assuming reliable underlying ad hoc transport protocols. In [11] the authors introduced a collaborative context determination approach optimized for MANETs. In that approach, a mobile node acquires context information from its neighboring peers. These studies are focusing on infrastructureless environments while the IMO can potentially apply to different environments.

The Aura project [12] associates accuracy and confidence values to context information and is optimized for ubiquitous environments. Other approaches that address the fundamental trade-off between information accuracy and management overhead have been published in [1], [2], [13], and [14]. The information collected is usually aggregated using functions such as SUM, AVERAGE, MIN, and MAX. GAP [1] and AGAP [2] are generic aggregation protocols with controllable accuracy, which are based on the constraints of the network environment. Resource-limited sensors may, for example, trade information accuracy for extended network lifetime. In our proposal, information accuracy is complemented with other parameters that adapt the information flow to the network environment, namely, the number of aggregation points and the structure of the overlay topology.

III.   THE INFORMATION MANAGEMENT OVERLAY

The Information Management Overlay is a management infrastructure that collects, processes, and disseminates information from/to the network entities and management applications, acting as an enabler for self-management functionality. It consists of the IMO Controller and a number of IMO-nodes placed at different points in the network, forming a hierarchy. A Management Application can interact with the IMO Controller (see Fig. 1) by specifying its requirements in terms of the information sources, the type of information, the monitoring rate, the aggregation function, etc. Furthermore, it can specify performance optimization requirements with respect to network overhead or processing cost. The IMO Controller performs overlay-wide control of the platform and enforces decisions by communicating with the appropriate IMO-nodes, in order to satisfy the above requirements.
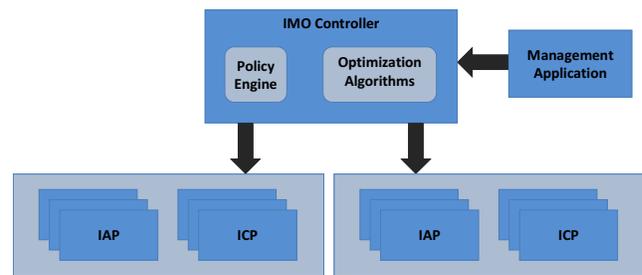

Fig. 1. The IMO structure

The IMO-nodes may have one of the following roles: (i) to collect information, acting as Information Collection Points (ICPs), (ii) to aggregate information, acting as Information Aggregation Points (IAPs), or (iii) both, acting as Information Collection & Aggregation Points (ICAPs). Our infrastructure supports both optimum placement of IMO-nodes and information filtering based on accuracy objectives, in order to adjust the performance-related trade-offs.

The key factors that allow the IMO to be scalable, efficient, and robust are the quantity and the placement of the IMO nodes. Since the IMO-nodes are a subset of the physical nodes, each overlay topology should be carefully created, based on algorithms that enable the optimum deployment of the IMO-nodes in an automated manner. Below we detail the different IMO components, namely: the Information Collection Points, the Information Aggregation Points, and the IMO Controller.

A.   Information Collection Points (ICPs)

The Information Collection Points act as sources of information: they monitor hardware and software for their state, present their capabilities, or collect configuration parameters. The IMO supports three types of monitoring queries to an ICP: (i) 1-time queries, which collect information that can be considered static, e.g., the number of CPUs, (ii) N-time queries, which collect information periodically, and (iii) continuous queries that monitor information in an on-going manner.
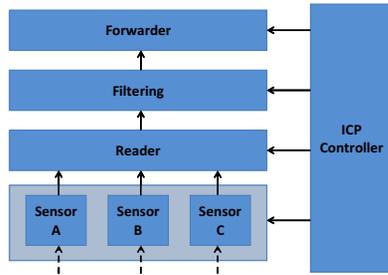
Fig. 2. Structure of Information Collection Points

ICPs should be located near the corresponding sources of information in order to reduce management overhead. Filtering rules based on accuracy objectives should be applied at the ICPs, especially for the N-time and continuous queries, for the same reason. Furthermore, the ICPs should not be many hops away from the corresponding IAPs.

Fig. 2 shows the structure of an ICP, which we have designed and implemented, consisting of 5 main components: the *sensors*, a *reader*, a *filter*, a *forwarder* and an *ICP controller*. These are described below.

The *sensors* can retrieve any information required. This can include common operations such as getting the state of a server with its CPU or memory usage, getting the state of a network interface by collecting the number of packets and number of bytes coming in and out, or getting the state of disks on a system presenting the total volume, free space, and used space. In our implementation, each sensor runs in its own thread allowing each one to collect data at different rates and also having the ability to turn them on and off if they are not needed.

The *reader* collects the raw measurement data from all of the sensors of an ICP. The collection can be done at a regular interval or as an event from the sensor itself. The reader collects data from many sensors and converts the raw data into a common measurement object used in the IMO framework. The format contains meta-data about the sensor and the time of day, and it contains the retrieved data from the sensor.

The *filter* takes measurements from the reader and can filter them out before they are sent on to the forwarder. Using this mechanism it is possible to reduce the volume of measurements from the ICP by only sending values that are significantly different from previous measurements. By using filtering [15] in this way, the ICP produces less load on the network. In our case, the filtering percentage matches the accuracy objective of the management application requesting the information.

The *forwarder* sends the measurements onto the network. The common measurement object is encoded into a network amenable measurement format. The measurements are encoded using XDR [16] as a way to minimize the size of the transmitted data. The XDR format is commonly used in monitoring systems [17] in order to reduce network loading.

The *ICP Controller* controls and manages the other ICP components. It controls (i) the lifecycle of the sensors, being able to turn them on and off, and to set the rate at which they collect data; (ii) the filtering process, by changing the filter or adapting an existing filter; (iii) the forwarder, by changing the
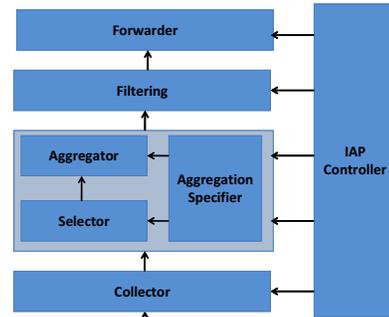

Fig. 3. Structure of Information Aggregation Points

attributes of the network (such as IP address and port) that the ICP is connected to.

For the IMO platform, we have developed various sensors, which can measure attributes from CPU, memory, and network components of a server host. We can also measure the same attributes of virtualized hosts by interacting with a hypervisor to collect these values. Finally, we have sensors that can send emulated measurements. These are useful for testing and evaluation purposes, with one example being an emulated response time, which we use in our experiments.

B. Information Aggregation Points (IAPs)

IAPs apply aggregation functions to the collected measurement information. The aggregation process increases the level of information abstraction, thereby transforming the data into a structured form, but at the same time reducing the load on the network. Aggregation works in situations where information consumers do not need a continuous stream of data from an ICP, but can get by with an approximation of the data. For example, getting an occasional measurement with the average of the volume of traffic on a network link may be enough for some applications. Some common aggregation functions include SUM, AVG, STDDEV, MIN and MAX.

Fig. 3 shows the structure of an IAP, which we have designed and built, consisting of 7 main components: a *collector*, an *aggregation specifier*, a *selector*, an *aggregator*, a *filter*, a *forwarder* and an *IAP Controller*. All of these components are described below.

The *collector* collects measurement data from the network and converts the XDR encoded measurement format into a measurement object. After this the measurement objects are saved in a data store for later aggregation processing. The data store used in the IMO is the Timeindexing Framework [18], [19] which allows any kind of data to be stored and retrieved using timestamps or time intervals. The Timeindexing Framework provides the mechanism by which arbitrary sequences of measurements can be selected and aggregated. It creates an index into the data, called a timeindex, and provides an API for accessing the data.

The *aggregation specifier*, the *selector*, and the *aggregator* are actually combined into an aggregation engine. The *aggregation specifier* specifies *when* the aggregator executes, *what* it aggregates, and *how* it aggregates. These three specifications are similar to those used in SLA compliance

systems [20], because the process of analyzing the data is similar.

The *when* specification is of the form: wake up every *N* seconds, which will cause the aggregation engine to wake up regularly to provide an aggregation. The *what* specification takes the form of a time interval, such as "from now, back 30 seconds". The *how* specification is the name of a function to aggregate the data, such as AVERAGE, SUM, etc.

The *selector* selects the required measurements to aggregate using the *what* specification. It determines what data is eventually chosen by applying the time interval, such as "from now, back 30 seconds", to the timeindex and selecting the relevant measurements. In this case, it will cause the selector to select the most recent 30 seconds worth of data. As the data store uses timeindexing, the time interval can be changed arbitrarily. Once the selection is complete the selected data is passed to the aggregator.

The *aggregator* aggregates the selected measurements presented by the *selector*. It uses the *how* specification to aggregate data. Although it is most common to use aggregation functions, such as SUM, AVERAGE, STDEV, MIN and MAX, the IAP Controller can pass in an arbitrary function into the aggregator in order to do the aggregation. This gives considerable power and flexibility when determining aggregations. Once the aggregation is calculated, the aggregated measurement data is passed to the *filter*.

The *filter* takes measurements from the aggregator and can filter them out before they are sent on to the forwarder. Again, this reduces the volume of measurements by only sending values that are significantly different from previous measurements. Using filtering in this way in the IAP, like filtering in the ICP, less load is produced on the network.

The *forwarder* sends the aggregated measurements onto the network. The common measurement object is encoded into the same network amenable format as in the ICP, the XDR. By having the same network format, the consumers of the measurement data do not need to know if data has come directly from and ICP or has come from an IAP. This allows hierarchies of elements to be composed as an IAP can further aggregate data from other IAPs, if this is required.

The *IAP Controller* controls and manages the other IAP components. It controls (i) the *collector*, by changing the attributes of the network that the IAP listens to, (ii) the aggregation process, by managing the *aggregation engine* and by passing in the *aggregation specifier*, (iii) the filtering process, by changing the *filter* or adapting an existing *filter*, (iv) the *forwarder*, by changing the attributes of the network (e.g. IP address and port) that the IAP sends to.

The aggregation engine itself is flexible enough to be given different aggregation specifications by the IAP Controller in order to process the data in varying way. For example, it can be configured to wake up once an hour and select data for the last day, and then apply an aggregation function. This is achieved using a mechanism that relies on plugins. These plugins represent code blocks, which can be pre-defined, such as an average aggregator, or can be defined to suit the need.

Information dissemination is done through the IAPs. As an application may request a specific piece of information from an IAP, the deployment location of the IAPs should also consider the locations and the traffic requirements of the nodes retrieving information. As well as requesting information, an application has the option to subscribe to an event-based notification service by setting an appropriate threshold to a specific type of information. Whenever this threshold is exceeded, the application is notified.

C. IMO Controller

The IMO Controller has centralized functionality and is responsible for the setup and optimization of the overlay. As shown in Fig. 1, it takes input from a management application regarding the optimization requirements (e.g. cpu/memory, network resources, or response time) and then configures the ICPs and the IAPs via their respective controllers. The functionality of the IMO Controller is realized by two components: (i) the optimization algorithms, and (ii) the policy engine. The optimization algorithms can potentially implement a variety of optimization tasks that involve performance related tradeoffs, but in its current form it focuses on algorithms for the placement of IAPs. The policy engine provides the necessary logic for the optimal configuration of the overlay according to the optimization requirements and the properties of the environment. These two components are discussed in more detail below.

1) Optimization Algorithms. The IMO Controller can implement various optimization algorithms. Example algorithms include (i) optimizing with respect to the network protocol deployed, or (ii) trading processing cost for communication cost by using compression techniques. The optimization considered in this work, however, concerns the placement of the IAPs in the network. This is a process, which is carried out when the overlay is initially deployed, but can also be triggered at run-time to react to changes in the network, or to emerging requirements from the management application.

We now present different IAP placement algorithms, namely the: Random, Hotspot and two variations of the Greedy Algorithm. Similar algorithms have been proposed in the area of Content Distribution Networks [21].

Random Algorithm

The Random algorithm randomly chooses *k* nodes from the set of *N* available nodes, based on a uniform distribution. These *k* nodes become the location for the IAPs. It is a very simple algorithm and thus produces insignificant processing overhead. It is suitable for performance requirements to improve resource efficiency in terms of CPU utilization or memory consumption.

Hotspot Algorithm

The Hotspot algorithm places the *k* IAPs near the nodes that have more chances to produce significant communication overhead. A node is considered as a "hotspot" when it is connected to a high number of nodes. In practice, node density is the important factor. For each potential IAP, a cost function *F*, that quantifies node density, is calculated. The cost function is elaborated as follows:

$$F(N_i) = a^3 + b^2 + c^1 \qquad (1)$$

where *a*, *b*, *c* are the number of nodes that have one hop, two hops and three hops distance from the node N_i, respectively. We note that the number of hops may not be the only parameter that expresses the network distance between two nodes. For example, a similar cost function may also include parameters such as network latency, available energy, free memory, CPU power, etc. The hotspot algorithm is suitable for performance requirements that focus on the network overhead optimization.

Greedy Algorithm

The Greedy algorithm places one IAP at a time. Every new iteration considers all previous selections in place, choosing the best node using a cost function that is based on the number of hops. The selected cost function should ideally reflect the performance requirements. For example, a requirement to maximize network lifetime may change this cost function from the number of hops to one that is a function of the available energy. An adaptive cost function that reflects the diverse node requirements in terms of resource availability and the performance requirements is a subject of a future work.

The *Greedy A* algorithm assumes that every node N_i is a potential source of data. Realistically however, only a subset of *N* nodes are data sources for the IMO. We call this subset *S*. Based on the assumption that the IMO is aware of the locations of the information sources, the *Greedy B* algorithm considers only the *S* nodes as information sources rather than every potential node (i.e., *N*). In our case, this information is passed from the Management Application to the IMO through the IMO Controller. In environments where all nodes are potential information sources, the *Greedy A* algorithm may be used instead.

It should be noted that the Random and Greedy algorithms are suitable for both the initial and the dynamic placement of IAPs because they can produce a result from a partial input. The Hotspot algorithm may be used only for the initial placement, because it requires a cost function calculation for each node in the network.

2) Policy Engine. It is evident that there are a number of parameters to be considered when configuring the information management overlay, such as the size of the network, the number of IAPs to be used, and the tolerance level of information accuracy. Furthermore, some of these parameters may change during the course of the system's operation and reconfiguration may be required to maintain optimized collection of information. For this reason, our approach incorporates policy-based management [22] as the mechanism to achieve adaptation in a flexible and simplified manner.

We have used policies to drive the functional behavior of the overlay, based on the requirements of the management application and the properties of the environment. The policies are executed by the IMO Controller and have an overlay-wide scope. Our implementation is based on the Ponder2 framework [23], which uses Event-Condition-Action (ECA) rules to encode strategies. It considers two policy types that: (i) configure and execute the appropriate IAP placement algorithm, and, (ii) set the accuracy level at which the information is collected. The decision on the placement

algorithm in the first policy type is based on a set of constraints that represent the network properties and the optimization requirements sent from the management application. An instance of such a policy is specified below in Ponder2 format, where aspects such as the topology size (medium – Med), its type (dynamic – Dyn), and the optimization requirement (response time – Resp) are encoded in the conditional part. Given an *event* to deploy the IMO (deplIMO), and all the conditions being true, then the *action* (execGreedy) executes a script that implements the Greedy algorithm, passing the number of IAPs to be deployed (3 in this case) as a parameter.

```
policy := root/fa    tory/ecapolicy cre    te.
policy event: root/event/    eplIMO.
policy condition: [ :topol    ize :topolType :opt    Type |
        (topolS    ze=="Med") & (topolType    ∋ "Dyn"
        (optzTy    te=="Resp") ].
policy action: [ root/plAlg    xecGreedy: 3 ].
root/policy at: "gr    edyAlg" put: policy.
policy active: true.
```

As mentioned in Section IIIA, the accuracy with which information is transmitted has an impact on the communication overhead. The IMO Controller can regulate this parameter on both ICPs and IAPs by configuring their filter. The following policy is enforced only if the management application requires the optimization of network resources, and sets the accuracy as a deviation percentage (5%).

```
policy := root/fa    tory/ecapolicy cre  3 te.
policy event: root/event/    3eplIMO.
policy condition: [ :optzT    pe | optzType=="Net" ].
policy action: [ root/fltr    setAcc: 5 ].
root/policy at: "f    3lter" put: policy.
policy active: true.
```

The advantage of introducing interpreted logic in the form of policies is that the behavior of the managed overlay can be dynamically changed without modifying the underlying implementation. The filtering value can, for example, be adjusted without interrupting the information collection process. In the current implementation, and for evaluation purposes, policies are manually triggered and the constraint parameter values are represented by static variables. Part of the future work in this area involves acquiring dynamic environment properties and optimization requirements, and then executing the policies at run-time.

IV. EXPERIMENTAL ANALYSIS

In this section, we evaluate different aspects of the IMO infrastructure using both real experiments and simulations. We implemented all basic functionalities of the IMO infrastructure, including Information Collection Points that monitor network and system resources, Information Aggregation Points that perform dynamic placement, a sample Management Application requesting information and the policy-driven IMO Controller. We carried out experiments to evaluate the impact of the following:

- The *filtering mechanism*, where we show how information accuracy can be traded for communication cost.
- The *information source*, where we show the impact of different sources of information, including CPU and

memory sensors, and an emulated source (i.e., the response time emulator).

- The *placement algorithms*, where we show their impact on the communication overhead and processing cost in small-scale and large-scale networks.

We complemented the real experiments with simulations in order to investigate the impact of the placement algorithms in large-scale topologies.

A. Experimental Setup

We deployed the IMO infrastructure on a test-bed that consists of two SUN servers with 4 CPU cores (2.5 GHz each) and 8GB of memory and one with 8 CPU cores (1.9 GHz each) and 32GB of memory. Each server hosts 10 virtual machines using the XEN virtualization platform [24]. With such a setup, we emulated a network that consists of 33 machines. In our experiments, each physical machine acts as an emulated Autonomous System (AS).

In the real scenarios, we gradually increased the number of physical machines from one to three, in order to show the behavior of our proposal in different topology scales. In these experiments, all the virtual machines act as information sources. One Aggregation Point is deployed dynamically using the AP placement algorithms and the Management Application requesting information is hosted on the last physical machine of each experimental run. *Greedy B* is the default placement algorithm used.

Here, we measure *throughput* at the IAPs and the Management Application and *transmission time* from the information collectors to the Information Aggregation Points. The Information Collection Points and Information Aggregation Points transmit a measurement every 30 seconds, unless it is otherwise stated. It should be noted that the experimental scenarios were defined as inputs to the IMO Controller and the corresponding policies.

In the simulation scenarios, we evaluated the proposed IAP placement algorithms using the linear topology shown in Fig. 4, matching the emulated topology in the test-bed. We implemented the simulator using TCL and executed the experiments on an Acer TravelMate 3022WTMi laptop with 1GB of memory. In our implementation, we used the Dijkstra's shortest path algorithm for any node distance calculation in hops.

For the simulations, we measure the average data transmitted per hop (measured in Bytes / sec / hop) and the average processing time for the IAP placement algorithm. The former metric quantifies the communication and the latter the processing cost.

In all experiments, we assume that the IMO is aware of the network topology graph. This is a reasonable assumption, since topology is one of the most important information aspects that need to be handled from such infrastructures.
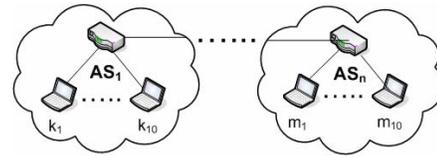

Fig. 4. Simulation topology

B. Real Measurements

In the following scenarios, we tested the IMO infrastructure under different conditions, focusing on different aspects each time. We evaluated the statistical accuracy of our results by performing 10 runs and calculating the standard deviation. We did not observe any large deviations. For example, in a scenario with 2 ASs and 20 VMs acting as information sources, the standard deviation in the *throughput* measurements was just 3.4% of the average *throughput* value. The evaluation results are presented below.

1) Impact of Information Filtering: As we can see in Fig. 5, the *throughput* at the AP is reduced significantly due to the filtering algorithm. So, filtering can adjust the trade-off between information accuracy and communication overhead. The impact increases with more information sources and larger topologies. In Fig. 6, we show the impact of filtering on the *throughput* that reaches the Management Application, for the case where the aggregation point transmits one measurement every 5 sec. Again, filtering improves communication overhead. It should be noted that in certain cases, the communication overhead may be zero (e.g., for 5% filtering with 3 ASs, as shown in Fig. 6); this happens when the deviation of the monitoring value never exceeds the filtering specification.

2) Impact of Information Sources: In this scenario, we show the impact of different information sources on the *throughput* and *transmission time*. We use different information sensors that monitor CPU utilization and memory usage. Additionally, we use a sensor that collects measurements from a response-time emulator. As shown in Fig. 7, the type of information has an impact on the communication cost. In our case, the emulated source associates with much less *throughput* than the CPU and memory sources. This is not reflected to the *transmission time* (see Fig. 8), in the case of 3 ASs.

3) Impact of IAP placement algorithms: In this scenario, we evaluate the impact of the placement algorithms on the communication cost. Here, we used the second version of the Greedy algorithm. In this case (Fig. 9), the random placement of the IAP often results to a location many hops away from the information sources, which increases significantly the communication overhead. On the other hand, an aggressive placement algorithm, such as the hotspot, can determine the most optimal location for the IAP. The greedy algorithm is a good choice too, because it performs well and is associated with less processing cost compared to the hotspot, especially if the number of sources is less than the available nodes. Larger topologies and numbers of IAPs are evaluated using simulations. The results are presented in the next section.
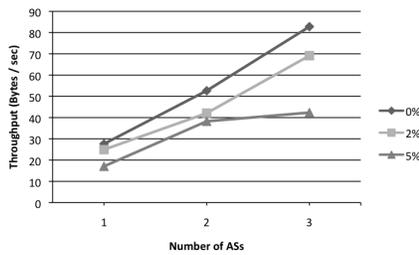
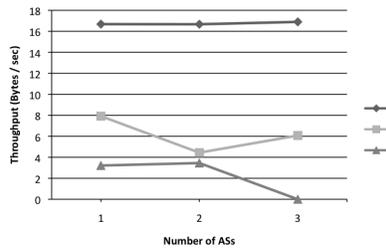Fig. 5. Impact of Filtering at the Aggregation Point

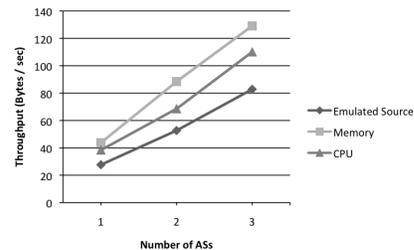Fig. 6. Impact of Filtering at the Management Application (rate 5 sec)

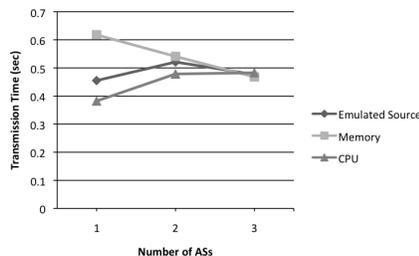Fig. 7. Impact of Information Source on the Aggregation Point

Fig. 8. Impact of Information Source on the Aggregation Point
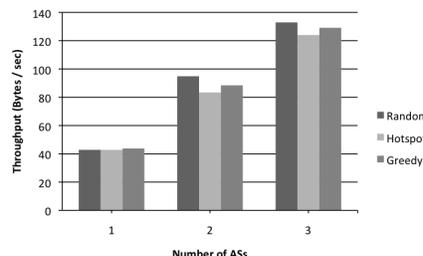

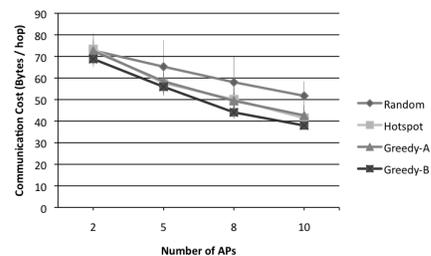Fig. 9. Impact of AP Placement Algorithm

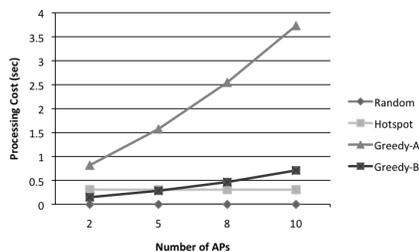Fig. 10. Communication Cost (5 ASs)

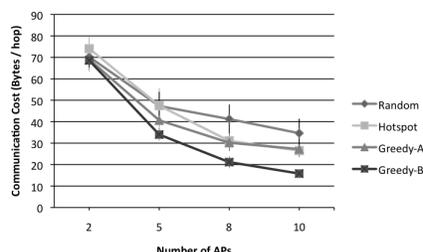Fig. 11. Processing Cost (5 ASs)

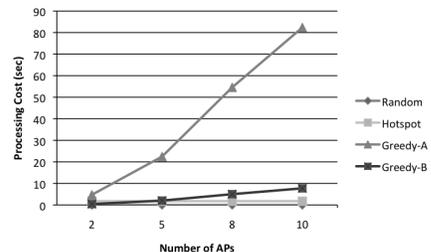Fig. 12. Communication Cost (10 ASs)

Fig. 13. Processing Cost (10 ASs)

C. Simulation Results

We ran two sets of simulations, consisting of 5 and 10 networks in the backbone line (i.e., Autonomous Systems - ASs), respectively. As illustrated in Fig. 4, 10 nodes were connected to each AS. All of our simulations have a non-deterministic nature. Consequently, we executed each one 20 times using different initiation values. Each of our figures depicts the average values of the particular metric. In the figures, we indicate the standard deviation of our measurements, every time it is not insignificant.

We randomly placed 10 ICPs in the topology, each of which periodically transmits data (i.e., with a period of 1 sec) to the nearest IAP (in number of hops). The data is collected from the IAP, processed, and transmitted again to a node that exploits this information (i.e., a Management Application). The node that resolves information is also placed randomly in the topology. In our scenario, we assume that each data packet has the size of 100 bytes. The data transmission between the IAP and the Information Consumer has a period of 10 seconds,

assuming that the aggregated information requires lower data transmission rates. In both scenarios, we ranged the number of IAPs from 2 to 10.

Figs. 11 and 13 show that the Random placement algorithm is associated with the lower processing cost. This result is consistent with the complexity of the algorithm that is *O(MN)* [21]. However, the lower complexity comes with a high communication cost (Figs. 10 and 12). The Hotspot and Greedy B algorithms have similar complexity (see Figs. 11 and 13) but the Greedy B algorithm performs better in terms of communication cost (as in Fig. 10 and 12). The Hotspot and Greedy A algorithms produce almost the same communication cost. The associated processing cost however, is significantly higher for the Greedy A algorithm.

In general, the Greedy B algorithm performs better in terms of communication and processing cost. In this example, the Greedy B algorithm produces almost the same communication cost with the Greedy A algorithm (Fig. 12). However, the latter does use 3 less IAPs.

V. Discussion & Open Issues

By having our results and analysis as a starting point, we can now map different optimization strategies to example performance requirements.

In the case when a *performance requirement for minimum network overhead* is set, the communication cost can be traded for resource consumption by: (i) using more efficient but resource expensive algorithms (like Hotspot or Greedy); (ii) using a large number of Information Aggregation Points; or (iii) using information compression techniques. Furthermore, filtering can significantly reduce communication cost, in case this satisfies the accuracy objectives of the management application.

When a *performance requirement to improve CPU utilization & memory consumption* is set, placement algorithms with less processing overhead should be preferred, e.g., Random or Greedy B. The number of IAPs should be reasonable because resource consumption increases proportionally.

Topology properties (e.g., topology size, frequency of topology changes etc) may call for further adjustments. For example, in case of dynamic environments, the Greedy and Random algorithms can perform dynamic adjustments to the positions of the IAPs. In case of static environments, the Hotspot algorithm is suitable for the initial placement of IAPs.

We plan to extend this work and address the following issues:

- to evaluate the IMO robustness with dynamic scenarios, e.g., dynamic topologies or information flow requirements.
- to enrich the IMO system with additional policy-driven functionality, e.g., consider additional programmable parameters of individual placement algorithms.
- to consider additional optimization algorithms that focus on other aspects of the information management, e.g., information distribution, network protocol used etc.

Finally, to aid researchers in this area it is a goal to release the IMO platform as an open-source software.

VI. Conclusions

This paper presents the design, the implementation, and the optimization issues of an Information Management Overlay. This IMO is a key component of the Autonomic Management Architecture [4] proposed in the Autonomic Internet project [3]. The IMO supports network management applications with information, thus acting as an enabler for self-management functionality. We discuss different strategies to regulate the information flow with respect to existing conditions and performance requirements. Finally, we validate our proposal both with real experiments and simulations.

References

[1] M. Dam, R. Stadler, "A generic protocol for network state aggregation," in *Proceedings of Radiovetenskap och Kommunikation (RVK)*, 2005.

[2] R. A. Gonzalez Prieto, "A-gap: An adaptive protocol for continuous network monitoring with accuracy objectives," *IEEE Transactions on Network and Service Management*, vol. 4, no. 1, pp. 2–12, June 2007.

[3] "Autonomic Internet (AutoI) Project," http://www.ist-autoi.eu/.

[4] A. Galis, S. Denazis, A. Bassi, P. Giacomin et al., *Management Architecture and Systems for Future Internet Networks*. IOS Press, http://www.iospress.nl, ISBN 978-1-60750-007-0, April 2009.

[5] D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, USA: ACM, 2003.

[6] J. Li, "Agent organization in the knowledge plane," Massachusetts Inst. of Technology, Tech. Rep. MIT-CSAIL-TR-2008-034, June 2008.

[7] C. H. Crawford and A. Dan, "emodel: Addressing the need for a flexible modeling framework in autonomic computing," *International Symposium on Modeling, Analysis, and Simulation of Computer Systems*, 2002.

[8] X. Dong, S. Hariri, L. Xue, H. Chen et al., "Autonomia: an autonomic computing environment," 2003, pp. 61–68. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs all.jsp?arnumber=1203684.

[9] A. Malatras, G. Pavlou, and S. Sivavakeesar, "A programmable framework for the deployment of services and protocols in mobile ad hoc networks," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 12–24, Dec. 2007.

[10] S. S. Yau et al., "Reconfigurable context-sensitive middleware for pervasive computing," *IEEE Pervasive Computing*, vol. 1, no. 3, 2002.

[11] J. Mantyjarvi, P. Huuskonen, and J. Himberg, "Collaborative context determination to support mobile terminal applications," *IEEE Wireless Communications*, vol. 9, pp. 39–45, 2002.

[12] G. Judd and P. Steenkiste, "Providing contextual information to pervasive computing applications," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA, 2003.

[13] G. Cormode, M. Garofalakis, S. Muthukrishnan, R. Rastogi, "Holistic aggregates in a networked world: distributed tracking of approximate quantiles," in *Proceedings of the ACM SIGMOD international conference on Management of data*. New York, USA, 2005.

[14] A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *The VLDB Journal*, vol. 13, no. 4, pp. 384–403, 2004.

[15] A. Cooke, A. J. G. Gray, L. Ma, W. Nutt et al., "R-gma: An information integration system for grid monitoring," in *Proceedings of the 11th International Conference on Cooperative Information Systems*, 2003.

[16] R. Srinivasan, "Xdr: External data representation standard," 1995.

[17] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation and experience," *Parallel Computing*, vol. 30, p. 2004, 2003.

[18] S. Clayman, "Time indexing - an introduction," May 2003, http://www.timeindexing.com/documentation/papers.html.

[19] S. Clayman, "Timeindexing repository," http://xircles.codehaus.org/projects/timeindexing.

[20] A. Sahai, et al., "Specifying and monitoring guarantees in commercial grids through sla," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid'03), 2002.

[21] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proceedings of IEEE INFOCOM*, 2001.

[22] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, 1994.

[23] "Ponder2 policy framework," http://www.ponder2.net/.

[24] P. Barham, B. Dragovic, K. Fraser, S. Hand et al., "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, 2003.

[25] B. Rochwerger, D. Breitgand, E. Levy, A. Galis et al., "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, 2009.