Controlling Performance Trade-offs in Adaptive Network Monitoring

Alberto Gonzalez Prieto and Rolf Stadler

School of Electrical Engineering
KTH, Royal Institute of Technology
{gonzalez, stadler}@ee.kth.se

***Abstract*— A key requirement for autonomic (i.e., self-*) management systems is a short adaptation time to changes in the networking conditions. In this paper, we show that the adaptation time of a distributed monitoring protocol can be controlled. We show this for A-GAP, a protocol for continuous monitoring of global metrics with controllable accuracy. We demonstrate through simulations that, for the case of A-GAP, the choice of the topology of the aggregation tree controls the trade-off between adaptation time and protocol overhead in steady-state. Generally, allowing a larger adaptation time permits reducing the protocol overhead. Our results suggest that the adaptation time primarily depends on the height of the aggregation tree and that the protocol overhead is strongly influenced by the number of internal nodes. We outline how A-GAP can be extended to dynamically self-configure and to continuously adapt its configuration to changing conditions, in order to meet a set of performance objectives, including adaptation time, protocol overhead, and estimation accuracy.**

***Index Terms*— Adaptive management, distributed management, real-time monitoring.**

I. Introduction

As networks have increased in size and complexity, the necessity for simplifying their management and reducing their operational costs has become evident. In response to this need, the vision of autonomic management, or more generally, autonomic computing [11] has emerged.

Following the autonomic computing vision, a human manager specifies her goals or high-level objectives for the network, which are expressed in form of behavioral rules, utility functions, and so forth. That is, the manager does not specify what has to be done, but what has to be achieved. These goals are the input to the autonomic system. Then, it is the responsibility of the system to take the necessary actions to achieve them. The goals must be met despite changes in the networking conditions caused by load changes, failures, etc. This requires autonomic systems to adapt to such changes. This adaptation is achieved by continuously checking the system status and reconfiguring the system appropriately. In other words, an autonomic system continuously takes actions to reach (and maintain) the desired operating point.

A key requirement to autonomic systems is a short adaptation time to changes in the networking conditions. In a dynamic environment, where networking conditions change fast, a slow-adapting system lags behind the system state. In such scenarios, a slow-adapting system provides outdated configurations, thus never reaching a configuration that permits to meet the manager's goals.

In this paper, we focus on decentralized real-time monitoring, a key building block for realizing large-scale autonomic systems. Real-time monitoring provides the necessary input to decision-making processes, enabling systems to perform self-configuration and self-healing tasks. We argue that, with the autonomic computing vision in mind, the monitoring protocol itself must be autonomic. As a consequence, given a set of objectives, such as the accuracy of the metric estimation, the monitoring protocol must self-configure in a way that these objectives can be met, and it must dynamically adapt to changing conditions in a timely manner.

Specifically, we focus on the adaptation time of such a monitoring system to changes in the performance objectives, network topology or networking conditions. We investigate this topic using (as an example) A-GAP, a tree-based distributed protocol (we developed in our earlier work [2]) for continuous monitoring of global metrics, designed to achieve a given monitoring accuracy with minimal overhead.

Global metrics describe the state of the entire network, in contrast to local metrics that describe the state of a device. Global metrics are often computed from device counters using aggregation functions, such as SUM, AVERAGE and MAX. Examples of global metrics in the context of an ISP include the total number of VoIP flows in the domain and the 50 subscribers with the longest end-to-end delay. Providing such metrics is key for achieving effective network supervision, quality assurance and proactive fault management.

Our paper makes three main contributions. First, we show that *the adaptation time of a distributed monitoring protocol can be controlled.* This result extends our earlier work centered on controlling the performance of such protocols in steady-state [2][3].

Second, we demonstrate (through simulation experiments) that there is a *trade-off between the adaptation time of the protocol and the protocol overhead in steady-state*. This trade-off is controlled by the topology of the tree for the case

of A-GAP, and we show that the choice of the topology can be used to achieve performance objectives.

While it is well known that the topology of the tree strongly influences the performance of monitoring protocols [5][6][7][8], there is no literature on how to effectively control the protocol performance through the tree topology.

Third, we outline *how the A-GAP protocol can be extended to dynamically self-configure*, in order to meet a set of performance objectives, including adaptation time, protocol overhead, and estimation accuracy.

The paper is organized as follows. Section II describes A-GAP. Section III demonstrates, through simulation, the dependencies between the adaptation time, the topology of the tree, and the protocol overhead. Section IV discusses our work towards extending A-GAP so that it can support several performance objectives simultaneously. Section V reviews related work. Section VI concludes the paper.

II. Overview of the A-GAP Protocol

A-GAP [2] is a protocol for continuous monitoring of global metrics, which aims at achieving a given monitoring accuracy with minimal overhead.

A-GAP assumes a distributed management architecture, whereby each network device participates in the computation by running management processes, either internally or on an external, associated device.

A-GAP is decentralized and asynchronous. It executes on an overlay that interconnects management processes associated with the managed devices. On this overlay, the protocol maintains a breadth first search spanning tree and updates the global metrics through incremental aggregation. Based on a stochastic model, it dynamically configures local filters that control whether an update is sent towards the root of the tree. A-GAP has proved effective in controlling the trade-off between accuracy and protocol overhead.

In A-GAP, each node holds information about its children in the tree, in order to compute the partial aggregate, i.e., the aggregate value of the local management variables from all nodes of the subtree where this node is the root.

In A-GAP, the mechanism we use for trading accuracy and overhead is a set of filters, one for each node in the system. When the partial aggregate of a node n changes, n sends an update to its parent if the difference between the value reported in its last update and the current value exceeds the local filter width F^n.

Estimating the network variable at the root node with minimal overhead for a given accuracy can be formalized as an optimization problem.

Let n be a node in the network graph, ω^n the rate of updates received by node n from its children, F^n the filter width of node n, E^{root} the distribution of the estimation error at the root node, and ε the accuracy objective. We formulate the problem as:

$$\text{Minimize } \max_{n}\{\omega^n\} \quad \text{s.t. } \mathrm{E}[|E^{root}|] \le \varepsilon \qquad (1)$$

whereby ω^n and E^{root} depend on the filter widths $(F^n)_n$, which are the decision variables.

We have developed a stochastic model for the monitoring process. The model is based on discrete-time Markov chains and describes individual nodes in their steady-state. The model permits us to compute the distribution E^{root} of the estimation error at the root node and the rate of updates ω^n processed by each node.

An optimal solution to (eq. 1) can be computed using a centralized algorithm. Such an approach, however, is not feasible for large networks, since the computational cost of this algorithm grows exponentially with the number of nodes. A-GAP realizes a distributed heuristic, which attempts to minimize the maximum processing load on all nodes by minimizing the load within each node's neighborhood. A-GAP maps (eq. 1) onto a local problem for each node n as follows:

$$\text{Minimize } \max_{\pi}\{\omega^\pi\} \quad \text{s.t. } \mathrm{E}[|E^n_{out}|] \le \varepsilon^n, \qquad (2)$$

where $\pi \in \{n\} \cup \gamma^n$ (γ^n is the set of children of node n). This means that node n attempts to minimize the maximum load in a neighborhood $\{n\} \cup \gamma^n$ for a given accuracy objective ε^n of its partial aggregate.

A node attempts to solve (eq. 2) by periodically re-computing the filter widths and accuracy objectives of its children, based on our model.

A-GAP computes the local filter widths and accuracy objectives in a decentralized and asynchronous fashion.

A. Initialization

A-GAP initializes by constructing a spanning tree on the management overlay. On all nodes of the aggregation tree, the accuracy objectives are set to zero and the filters are set to the minimal width, so that changes to the local variable are reported up the aggregation tree to the root node. During a warm-up period, all model variables are initialized on all nodes.

B. Estimating the evolution of the local variables

The evolution of a local variable is modeled as a random walk. A-GAP continuously estimates the step sizes for the random walk model on each node. This estimation is based on the maximum likelihood estimator (MLE). In other words, we compute the frequency of each step size.

C. Re-computing local filters and accuracy objectives

Each node asynchronously and periodically executes a control cycle. It starts with the node polling each child for the model variables. Then, it selects a subset of its children for filter re-computation. It computes the new filters that minimize its processing load, while observing its accuracy objective. The determination of the filters is based on a grid search, in which the search space is limited to $[F^n\text{-}1, F^n\text{+}1]$. Next, the new accuracy objectives are computed. Finally, the node computes its own model variables.

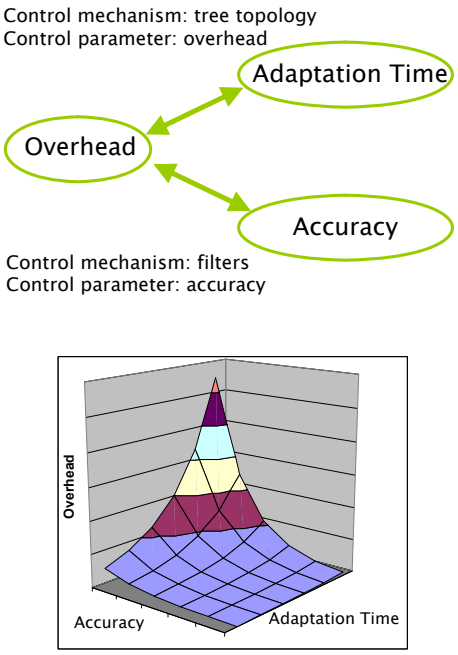

Fig. 1. Performance trade-offs in decentralized monitoring. The figure on the bottom shows the border of the feasible region for a protocol such as A-GAP.

D. *The two planes of A-GAP*

A-GAP can be understood as operating in two different planes. On the aggregation plane, updates of the local variables are propagated towards the root and filtering occurs. In this plane, information flows bottom-up, from the leaves towards the root.

In the control plane the filters are computed. In this plane, information flows in both directions, bottom-up and top-down. The model variables (step sizes and errors) flow from the leaves towards the root and are incrementally aggregated. The filters are distributed top-down, from the parents to their respective children.

E. *Controlling performance trade-offs*

The configuration of A-GAP is characterized by the topology of the aggregation tree and the filter widths in all the nodes. This configuration determines the performance of the protocol and involves some trade-offs (see Fig. 1). In our previous work [2][3], we focused on the trade-off between overhead and accuracy. This trade-off is controlled by the filter widths. In this paper, we have identified a second trade-off, adaptation time vs. overhead. As we will show, this trade-off can be controlled with the topology of the aggregation tree.

III. SIMULATION-BASED ANALYSIS

In this section, our goal is to understand how to control the performance of A-GAP through the topology of the aggregation tree and the performance trade-offs involved. We demonstrate the impact of the topology of the aggregation tree in the protocol performance (i.e., in its adaptation time and overhead in steady-state.)

A. *Simulation setup and scenario*

We have analyzed the performance of A-GAP through extensive simulations using the SIMPSON simulator [1].

Performance metrics. We consider two performance metrics. First, the adaptation time in terms of the duration of the transient period (i.e., the time it takes the overhead of A-GAP to reach the steady-state, see Fig. 11 for an example) after setting a new accuracy objective.

Second, the protocol overhead in terms of the number of processed updates by the nodes in steady-state. We consider the maximum overhead across all nodes, the overall overhead, and its distribution across nodes.

Local variables. The local management variables in the simulation experiments are updated asynchronously, once every second, following a random walk. These random walks are statistically independent and identically-distributed (iid), both in time and space. There are two main reasons for using a random walk model. First, our results in [2] support the use of this model for real traces of HTTP flows. Second, local variables have been modeled as random walks in similar contexts to ours [9][10].

Network sizes. The results presented in this paper are based on simulations for networks ranging from 16 to 200 nodes.

Aggregation tree topologies. The number of different aggregation tree topologies that can be created, even for small systems, is very large. In order to make this study tractable, we have limited it to a set of trees that we build as follows. First, with a subset of the nodes in the system, (the dark nodes in Fig. 2) we create a balanced tree of height h and constant tree degree d (shown in bold in Fig. 2). Second, the rest of nodes in the system are evenly assigned to this subset as leaves (the light nodes in Fig. 2). Note that h and d determine the number of internal nodes in the tree (we also refer to them as *aggregating nodes*). Fig. 2 shows two examples of such trees for a network size of 35 nodes. The top tree is created using values h=2 and d=2. As a consequence, the tree has 7 aggregating nodes and 28 leaf nodes. The bottom tree is created using values h=2 and d=3 (13 aggregating nodes and 22 leaf nodes). We have explored the configuration space along these two dimensions, h and d.

A key concept related to the topology of the aggregation tree is the *degree of decentralization*, which is the fraction of nodes in the system acting as aggregating nodes (i.e., cooperating in computing the global metric and the local filters). The more aggregating nodes, the higher the degree of decentralization.

Other simulation parameters. All evaluation scenarios share the following settings. Link speeds are 100 Mbps. The communication delay is 4 ms, and the time to process a message at a node is 1 ms.

For all simulations, the total computational resources devoted to the control plane are 200 control cycles per second. This means that for a tree topology with 10 aggregating nodes, each of them executes 20 control cycles per second. While, for

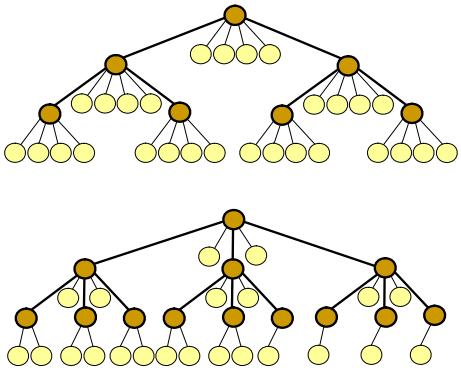

Fig. 2. Two different aggregation trees for a network of 35 nodes. In bold, we depict a balanced tree of aggregating nodes (dark nodes) with constant tree degree. The rest of nodes in the system are evenly assigned to these aggregating nodes as leaves (light nodes).

a tree topology with 20 aggregating nodes, each of them executes 10 cycles per second. For all experiments, in each control cycle, two filters are recomputed.

The accuracy objective (in this case, expressed in terms of the average error) is set to 5 units.

B. *The trade-off between adaptation time and overhead*

We have measured the protocol adaptation time in function of the experienced overhead (the maximum overhead across all nodes) for different aggregation tree topologies. Fig. 3 shows the measurement results for a network of 200 nodes. Every point in Fig. 2 corresponds to a simulation run.

As can be seen, the adaptation time decreases monotonically, as the overhead increases. For smaller overheads, the adaptation time decreases faster than for larger overheads.

Consequently, the adaptation time can be reduced by allowing a larger overhead. For example, compared to an overhead of 18,3 updates/sec, allowing an overhead of 20,5 updates/sec reduces the adaptation time by 55%. An overhead of 29,1 updates/sec reduces the adaptation time by 97%.

Next, we study in more detail the impact of the aggregation tree topology on each of the performance metrics individually.

C. *Impact of the aggregation tree topology in the protocol adaptation time.*

For the simulations described above, we have analyzed the adaptation time. That is, the time it takes the overhead of A-GAP to reach the steady-state after setting a new accuracy objective (see Fig. 11 for an example).

Fig. 4 shows the results for a network size of 200 nodes. We observe that the adaptation time grows with the height of the tree. First, the growth is slow. Then, it increases sharply. For all evaluated aggregation trees and network sizes, this rise happens for tree heights of 3 or 4 levels.

Note that the adaptation time depends on the computational resources assigned to the control plane, responsible for the computation of the filters. The cost of computing filters is controlled by two A-GAP parameters: the control cycle period τ, and the number of filters re-computed in each control cycle

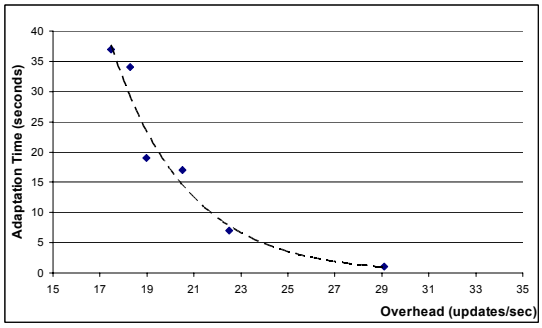

Fig. 3. Adaptation time as a function of the incurred overhead for a network of 200 nodes.

$|\Omega|$. This cost is inversely proportional to τ and grows exponentially with $|\Omega|$.

Increasing τ (or reducing $|\Omega|$) reduces this cost. However, this comes at the cost of reducing the adaptability of the system. Specifically, it would produce longer adaptation times.

D. *Impact of the aggregation tree topology in the management overhead incurred in steady-state*

We have analyzed the overhead in steady-state for different aggregation trees for networks of different sizes. Fig. 5 shows the measurement results. The y axis represents the maximum number of processed updates across all nodes. The x axis represents the number of aggregating nodes in the tree, i.e., the degree of decentralization. Every point in Fig. 5 corresponds to one simulation run.

We observe that as we increase the degree of decentralization, the overhead decreases. For low decentralization degrees, the overhead decreases faster than for high decentralization degrees.

From these results, we conclude that the overhead can be significantly reduced adopting configurations with a low degree of decentralization. For instance, for a 200-node network, using just 7 aggregating nodes (less than 5% of the nodes in the network) the overhead can be reduced by more than 75% compared to using one aggregating node (i.e., using a centralized approach).

An interesting observation is that these results suggest that the overhead is almost independent from the tree degree for this type of trees. This holds for all our experiments except

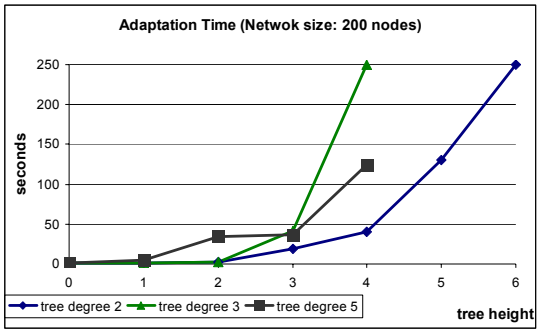

Fig. 4. Adaptation time for A-GAP as a function of the tree height for different aggregation tree degrees for a network of size 200.

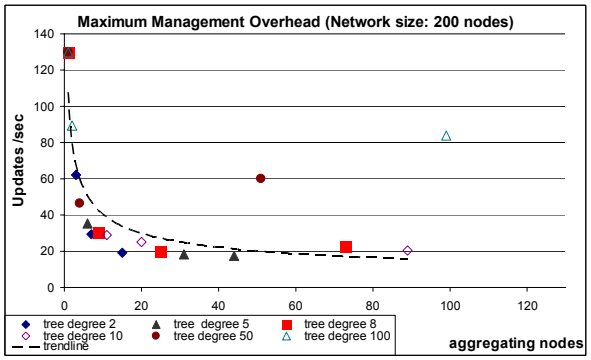

Fig. 5. Management overhead incurred by A-GAP as a function of the number of aggregating nodes (degree of decentralization) for different aggregation tree degrees.

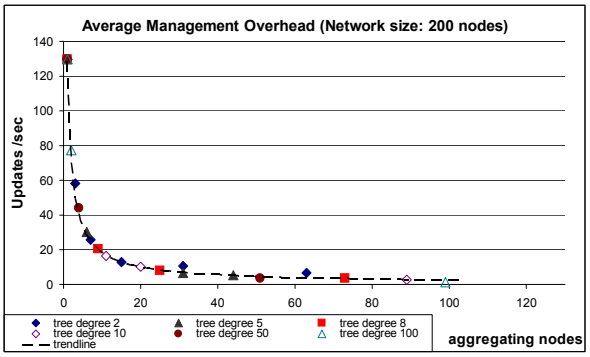

Fig. 6. Average management overhead incurred by A-GAP as a function of the number of aggregating nodes (degree of decentralization) for different aggregation tree degrees.

those where the tree degree is comparable to the network size (for tree degrees of 50 and 100 in the case of a 200-node network. This shows in Fig. 5, where two points, corresponding to tree degrees 50 and 100, do not fit into the negative exponential trend). This suggests that (in terms of protocol overhead, and for balanced trees of constant degree) it is more important how much to decentralize, rather than how. In other words, it is more important the fraction of nodes in the system acting as aggregating nodes than other characteristics of the aggregation tree.

For this set of simulations, we have also analyzed the average number of processed updates per node (Fig. 6). We observe that the average overhead behaves similarly to the maximum overhead. It follows a negative exponential trend (i.e., increasing the degree of decentralization reduces both the maximum load and the average load) and the tree degree does not impact the average overhead either.

We have also computed the distribution of the average load, i.e., the average number of processed updates over the aggregating nodes. Fig. 7 shows the results for a network of 200 nodes and aggregation trees of degree 5 and different heights *h*. Each curve represents the distribution of the average load for a single run.

The key observation is that the higher the tree height *h* (and therefore, the degree of decentralization), the larger the variation in management overhead among aggregating nodes. In other words, the higher the degree of decentralization, the worse the system balances the load among the aggregating nodes.

Finally, we have computed the overall management overhead, i.e., the sum of the loads across all aggregating nodes, for these experiments (Fig. 8). The first observation is that it increases as we increase the degree of decentralization. A second observation is that this increase follows a logarithmic function. This means that the cost of over-decentralizing is very limited in terms of management overhead.

Note that the amount of computational resources devoted to the control plane (the number of control cycles executed per second) does not impact the overhead in the steady-state. It only affects the duration of the adaptation time

E. Impact of network size

In this section, we analyze the performance of A-GAP as a function of the network size for different aggregation tree topologies.

Fig. 9 shows the adaptation time as a function of the tree height for a network of size 100. Fig. 4 shows the results for a network size of 200 nodes. Every measurement point corresponds to a simulation run.

We observe that, qualitatively, the behavior is similar for different network sizes. The adaptation time grows with the height of the tree. First, the growth is slow. Then, it increases sharply.

We also observe the turning point tends to slowly increase

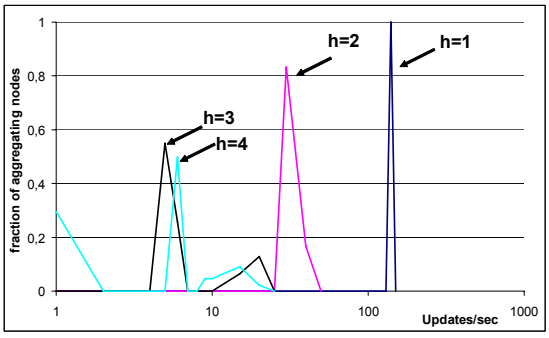

Fig. 7. Distribution of the management overhead across all aggregating nodes for aggregation trees of degree 5 and different tree heights *h* (the size of the network is 200 nodes).

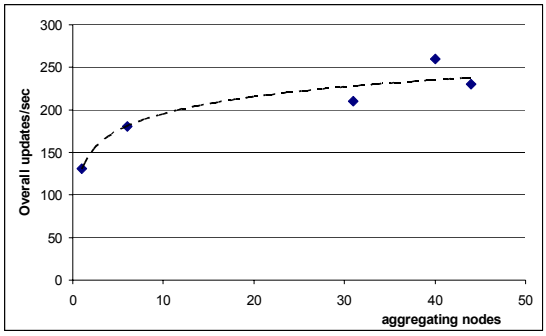

Fig. 8. Overall management overhead for different aggregation trees of degree 5 (the size of the network is 200 nodes). The black dotted line represents a logarithmic trend.

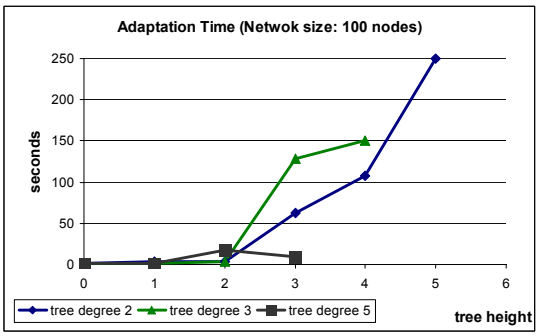

Fig. 9. Adaptation time for A-GAP as a function of the tree height for different aggregation tree degrees for a network of size 100.

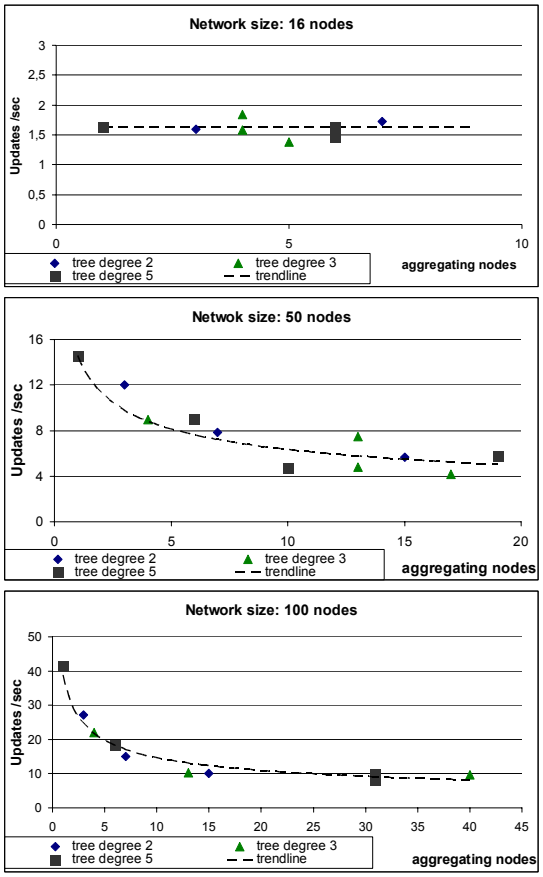

Fig. 10. Management overhead incurred by A-GAP as a function of the number of aggregating nodes for different network sizes.

with the network size. For a network with 100 nodes, this rise happens for a tree height of 3 levels. For a network double that size, the rise happens for a tree height of 4 levels.

Fig. 10 shows the protocol overhead as a function of the decentralization degree for different network sizes and aggregation tree degrees. Every measurement point corresponds to a simulation run.

We observe that for a very small network (16 nodes) the overhead is very similar for all decentralization degrees. This is an expected result: the benefits of decentralization are commonly slim or non-existent in small networks.

We also observe that for networks with 50 nodes or more, the overhead is significantly reduced by decentralizing the monitoring task. For instance, using 10 aggregating nodes, in a network of 50 nodes, the overhead can be reduced by 75% compared to using one aggregating node (i.e., using a centralized approach).

While we expect an overhead reduction by decentralizing the task in large networks, it has surprised us, that this is the case even for networks as small as 50-node ones. This strongly speaks for decentralization in a wide range of scenarios.

IV. Self-configuring A-GAP

In this section, our objective is to extend the A-GAP protocol so that it self-configures in such a way that a set of performance objectives are met. Specifically, we want to determine the topology of the aggregation tree and the filters so that the adaptation time is minimized for a given maximum overhead and a given maximum estimation error. The determination of such a configuration can be formalized as follows.

Let n be a node in the network graph, ω^n the rate of updates received by node n from its children, λ the overhead objective, α the adaptation time to changes in the networking conditions, E^{root} the distribution of the estimation error at the root node, and ε the accuracy objective. We formulate the problem as:

$$\text{Minimize} \quad \alpha \qquad (3)$$
$$\text{s.t.} \quad \omega^n \le \lambda \quad \forall\, n$$
$$E[|E^{root}|] \le \varepsilon$$

To date, we do not have a closed form formulation for the optimization problem (eq. 3), which would show explicitly how the performance objectives depend on the decision variables, i.e., the topology of the aggregation tree and the filter widths.

Our approach for solving the optimization problem consists in evaluating different candidate solutions. These evaluations are performed by the control plane of A-GAP and are based on our stochastic model.

A. Accuracy of A-GAP performance predictions

As the extension of A-GAP relies on predicting the performance of candidate configurations, we need to assess the accuracy of the predictions made by A-GAP. The predictions are based on our stochastic model for the monitoring process. The model permits to predict, for any given tree, the error distribution, the average overhead on each node in steady-state and the adaptation time.

In our previous works, we have shown that the error distribution predictions by A-GAP are very accurate [2][3]. Now, we focus on the adaptation time and the overhead.

For all the experiments we have run, we have analyzed both the performance estimation by A-GAP and the actual performance. Regarding the average overhead on a node, the difference between the prediction and the measurement is never above 1.75 updates per second (commonly, it is below 1 update/sec). This makes the predictions for the overhead-related performance objectives very accurate.

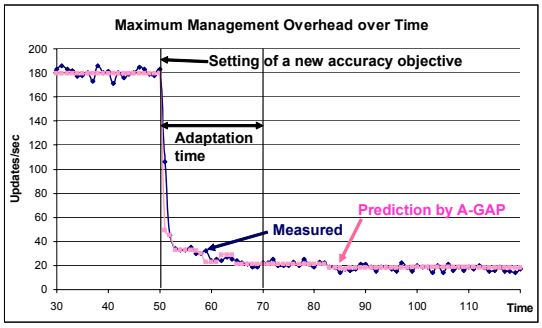

Fig. 11. Maximum management overhead over time incurred by A-GAP for a network of 200 nodes and an aggregation tree with 37 aggregating nodes and tree degree = 5.

Fig. 11 shows a representative example of performance prediction by A-GAP. It corresponds to a simulation run for a network of 200 nodes and an aggregation tree with 37 aggregating nodes and tree degree= 5. The figure shows the maximum overhead (1-second averages) across all nodes over time predicted by A-GAP. The figure also shows the measured maximum overhead. We can see that after setting a new accuracy objective (t= 50 seconds) a transient of a few seconds follows. (In other words, it takes A-GAP a few seconds to adapt to the new accuracy objective.) The predictions of the overhead are very accurate during both the transient and the steady-state.

Based on these and other experiments, we conclude that A-GAP predicts the performance of a given aggregation tree with high accuracy.

B. Extending A-GAP to support several objectives simultaneously – adaptation time, overhead, and estimation accuracy

As discussed in section III and also in [2], the performance of A-GAP depends on its configuration, which is given by the filters and the tree topology. A-GAP configures the filters dynamically with the objective of minimizing the overhead. The topology of the aggregation tree self-configures into a BFS tree. Note though that the topology of the aggregation tree is not constructed with the goal of meeting the performance objectives in Fig. 1.

Our goal therefore is to extend A-GAP in such a way that it also self-configures the tree topology with the objective of meeting a given set of performance metrics, specifically those captured in (eq.3).

Our approach is to use A-GAP's control plane (see Section II) to evaluate different candidate tree topologies and have the protocol re-configure into the topology that best meets the performance objectives. The basis for evaluating a candidate tree topology is the stochastic model of the A-GAP monitoring process. This model allows the protocol to predict the management overhead and error distribution for any given tree topology.

Realizing this approach primarily means extending the control plane functionality of A-GAP. We have added two new processes to this plane, E1 and E2, which dynamically compute the topology of the aggregation tree. They run in parallel to the original processes of A-GAP, which dynamically configure the filters.

The root node runs process E1. All nodes run process E2. The pseudo-code for these new processes is given below.

E1 (root node)

```
Input: performance objectives, system size

1 initiate creation of candidate trees
2 while (not all candidates evaluated)
3   send candidate definition to nodes
4   request candidate evaluation
5   collect estimated candidate performance
6 select best-performing candidate
7 send new topology definition to nodes
```

E2 (node of candidate tree)

```
Input: node id

8 on candidate evaluation request
9   creation of candidate tree
10  run filter computation until steady-state is
    reached
11  aggregate and report predicted overhead
    towards root
```

Note that E1 and E2 are high-level descriptions. For instance, the candidate trees are created in a decentralized way, which is not further elaborated here.

Note that the evaluation of candidate trees does not impact the estimation of the global metric, i.e., the output of A-GAP towards the administrator. This is because the estimation is performed in the aggregation plane, while the evaluation of candidate trees is performed exclusively in the control plane.

Finally, note that during the filter computation step (10), each node periodically solves an optimization problem. The time it takes to complete step 10 depends linearly on this period and, consequently, on the computational resources assigned to the control plane, responsible for these computations.

V. Related Work

We have shown that the topology of the aggregation tree controls the trade-off between the adaptation time of the protocol and the protocol overhead in steady-state.

While it is well known that the topology of the aggregation tree strongly influences the performance of monitoring protocols [5][6][7][8], to the best of our knowledge, there is no literature on how to determine the optimal topology for meeting performance objectives in the context of continuous real-time monitoring of global metrics.

The work in [6] focuses on 1-time queries (which ask for a state snapshot) with the best possible accuracy. For such queries, the authors argue that the problem of finding the optimal tree topology (in terms of overall overhead) is identical to the Steiner tree problem on graphs, which is NP-complete. In contrast to [6], we consider queries that request a continuous estimation aggregate with controllable accuracy. An additional difference with that work is that the

performance metric we consider is the maximum overhead across all nodes. (Our goal is to avoid overloading any node, while the goal in [6] is the maximization of the lifetime of a wireless sensor network, which is associated with the overall overhead in the literature.). Because of these differences we can not apply those solutions in the scenarios we consider.

The work in [5] also focuses on 1-time queries. The authors illustrate the dependency between the overhead of their algorithm and the tree topology with two extreme topologies, a star and a chain. In contrast, we investigate a wider range of topologies, and consider not only the overhead, but also the adaptation time and the trade-off between them.

The work in [7] focuses on wireless environments and discusses the relevance of using a tree composed of branches with low packet losses and delays. In this work, we focus on scenarios with fixed networks and regard the work in [7] as complementary.

In [8], the authors demonstrate that considering the query semantics when creating the aggregation tree can reduce the size of the monitoring messages for 1-time queries having (sql-like) "group by" clauses. However, it is unclear the benefits of such an approach in the context of continuous queries, which is our focus.

The existence of a trade-off between adaptation time and overhead in steady-state has been reported in [4]. That work shows that as the number of mobile agents deployed is increased, the management overhead (in steady-state) decreases, but the adaptation to changing conditions becomes more costly in terms of management traffic.

VI. Conclusion

The contribution of this paper is three-fold. First, using A-GAP as an example, we show that the adaptation time of a distributed monitoring protocol can be controlled. This result adds to our previous work, showing that *the performance of a distributed monitoring protocol can be controlled along different performance dimensions*, including protocol overhead, protocol accuracy, and adaptation time. This is illustrated in Fig. 1.

Second, we demonstrate that, for the case of A-GAP, *there is a trade-off between the adaptation time of the protocol and the protocol overhead in steady-state*. This trade-off is controlled by the topology of the aggregation tree, and we show that the choice of the topology can be used to achieve performance objectives. Generally, allowing a larger protocol overhead permits reducing the adaptation time. Furthermore, a small increase in overhead significantly reduces the adaptation time (for instance, in one of the scenarios, a 10% increase in overhead, reduces the adaptation time by 55%).

The adaptation time primarily depends on the height of the aggregation tree. Our results show that the adaptation time remains short for low tree heights, and then it rises sharply with growing tree height. (In our experiments, this rise happens at the height of 3 or 4 levels.)

Our results suggest that *the protocol overhead is strongly influenced by the number of aggregating nodes*, while other characteristics of the aggregation tree seem to have a weak impact. In addition, the overall management overhead scales logarithmically with the number of aggregating nodes, which speaks for the scalability of A-GAP.

As the third contribution of this paper, we outline how *A-GAP can be extended to dynamically self-configure*, in order to meet a set of performance objectives, while continuously adapting to changing conditions. (This means that the protocol can move the system state towards a target operating point in the feasible region in Fig. 1.). Note that the basis for such autonomic behavior is the stochastic model for A-GAP, since it permits predicting the performance for a given aggregation tree topology and local filters. Based on the model, A-GAP's control plane can evaluate different candidate trees and determine the best topology for achieving the objectives.

As for future work, we want to evaluate various design options for an autonomic A-GAP, test the protocol in a prototype environment and extend our simulation studies to larger network sizes.

Acknowledgment

This work has been part of 4WARD, a 7th Framework ISP project funded by the European Commission. It has further been supported by the ACCESS Linnaeus Center at KTH.

References

[1] K. Lim and R. Stadler, "SIMPSON — a SIMple Pattern Simulator fOr Networks". http://www.s3.kth.se/lcn/software/simpson.shtml, Jan 2009.

[2] A. Gonzalez Prieto and R.Stadler, "A-GAP: An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives", IEEE Transactions on Network and Service Management (TNSM), Vol. 4, No. 1, pp. 2-12, June 2007.

[3] A. Gonzalez Prieto and R.Stadler, "Monitoring Flow Aggregates with Controllable Accuracy", 10th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS 2007), San José, California, USA, October 2007.

[4] A. Liotta et al., "A Self-Adaptable Agent System for Efficient Information Gathering", 3rd IEEE/ACM International Workshop on Mobile Agents for Telecommunication Applications (MATA 2001), Montreal, Canada, August 2001.

[5] S.R. Madden et al., "TAG: a tiny aggregation service for ad-hoc sensor networks", 5th Symposium on Operating Systems Design and Implementation, Boston, USA, December 2002.

[6] B. Krishnamachari et al., "Modelling data-centric routing in wireless sensor networks", Technical Report CENG 02-14, USC Computer Engineering, 2002.

[7] C. Intanagonwiwat et al., "Impact of network density on data aggregation in wireless sensor networks", 22nd International Conference on Distributed Computing Systems, Vienna, Austria, July 2002.

[8] M. A. Sharaf et al., "Balancing energy efficiency and quality of aggregate data in sensor networks", ACM International Journal on Very Large Data Bases, Vol. 13, No. 4, pp. 384-403, December 2004.

[9] C. Olston et al., "Adaptive precision setting for cached approximate values", ACM SIGMOD 2001, Santa Barbara, USA, May 2001.

[10] A. Deligiannakis et al., "Hierarchical in-network data aggregation with quality guarantees", 9th International Conference on Extending Database Technology (EDBT), Crete, Greece, March 2004.

[11] J. O. Kephart and D. M. Chess. "The Vision of Autonomic Computing", IEEE Computer, Vol. 36, No. 1, pp. 41-50, January, 2003.