A learning packet-based autonomic network architecture

Zeinab Movahedi, Rami Langar, and Guy Pujolle

Abstract—Autonomic computing paradigm represents an emerging solution to deal with the ever-increasing size and complexity of managing IT systems. When applied to the networking, it relates to the capability of the network to operate and serve its objectives optimally by managing its own self without external intervention. We distinguish between two levels of management: Intra-application level which consists in managing different parameters within an application or service in order to optimize it; and inter-application level which is in charge of optimizing between several different applications and services. In this paper, we propose a generic approach applicable to both area. Our proposed architecture is based on random neural networks and uses the reinforcement learning. This enables the network to continuously converge to optimal configurations when network conditions change. In addition, the random neural network decision-making process is fed by normal transiting packets in the network, which significantly reduce the amount of control traffic. To show the effectiveness or our architecture, a case study consisting in optimizing the performance of mobile ad hoc network routing protocols is used. We proposed a dynamic routing protocol which interacts continuously with the architecture in order to enhance the network operation. Simulations show that the architecture improves significantly the QoS performance of ad hoc routing protocols.

Index Terms—Autonomic architecture, cognitive networking, learning mechanisms.

1 Introduction

The recent advances in computing and communication technologies have resulted in an explosive growth in scale and complexity of computing environments. This evolution overwhelms the capabilities of current human-centric management approaches. To overcome this ever-increasing size and complexity of management, autonomic computing paradigm has been proposed as a promising approach. By analogy with the autonomous nervous system of the human body, which checks blood sugar levels and maintains normal body temperature without any conscious effort, the autonomic computing aims at developing self-managing computing systems that manage themselves optimally given high-level objectives from administrators.

IBM proposed a generic reference model, called MAPE-K model (Monitor, Analyze, Plan, Execute and Knowledge) which provides a conceptual view of autonomic management process. Following this model, the self-management is achieved through monitoring computing resources, analyzing the collected data, planing adaptation tasks and executing the decided plans. This process forms an autonomic control loop (ACL) around computing resources which is guided by high-level objectives previously dictated by the administrator.

An immediate field of application of autonomic principles is on network management where the complexity becomes already an issue. Hence, a significant body of work has been dedicated on applying autonomic properties for managing network resources. The existing autonomic network architectures rely on monitoring both local and network-wide views which enables the optimization of the overall network operation. While some of existing architectures provide all network elements with network-wide knowledge, there are some others defining a monitoring overlay composed of a set of network elements. Whether approach is used, the challenge between network awareness and monitoring overhead should be considered.

In most of existing architectural solutions, the analyze and plan components are based on policies [1] which determine actions to be taken when an event occurs or certain conditions are met. Although the straightforwardness of policy-based approach, the need for low-level configuration details does not meet the ultimate objective of autonomic computing. Moreover, policies are prone to administrator errors which may be conflicted or lead to non optimized resource usage.

In few other research work, goal policies [2] and utility functions [3] are used as alternative approaches. Goal policies divide the states of the system into desirable

• *Authors are with the Department of Network and Computer Engineering, Laboratoire d'Informatique de Paris 6, Paris, France. E-mail: name.lastname@lip6.fr*

and undesirable ones and specifies criteria that characterize desirable network states. Although this approach removes the limitations of the policy-based adaptation, it only attempts to lead the system to a desirable state. However, when the system reaches a desirable state, it would not try to improve its performance anymore. Moreover, when a desirable state cannot be reached, the system does not know which among the undesirable states is least bad.

Utility functions, on the other hand, consists in evaluating the usefulness of possible configurations of network parameters. The adaptation engine bases its reaction on the configuration which meets higher network utility value. Comparing to policy and goal based reasoning, the utility function fits better the optimal solution since the system selects the configuration which meets higher network utility value according to the applied utility function. However, the definition of the corresponding utility function remains an issue.

To address the aforementioned issues of current autonomic solutions, we propose the Learning Packet-Based Autonomic (LPBA) architecture which uses random neural network to model the decision-making process within the autonomic architecture. The proposed adaptation scheme is enriched with reinforcement learning mechanism which uses past experiences to enhance future operations. Compared with existing autonomic architectures which rely on resource consuming monitoring approaches, normal transiting packets in the network are used to carry necessary global views.

The rest of this article is organized as follows. Section 2 reports the related work carried out on autonomic network management architectures. Section 3 describes our proposal followed by some performance results presented in section 4. Finally, section 5 concludes this paper.

2 Proposed architectures

Several autonomic network architectures have been proposed in the literature, each one based on different self-adaptation and monitoring mechanisms.

In [4], Pavlou et al. proposed a hierarchical policy-based management architecture for MANETs, denoted as DA-MANET. It consists in a two-level distributed cluster-based architecture: the lower-level deals with cluster-wide management. The high-level tier orchestrates between lower-level clusters. Both levels of manager nodes are dynamically selected using a hyper-cluster formation algorithm inspired from the backbone overlay networks. Context information is collected from terminal nodes and reported hierarchically up to the corresponding highest management level. The main issue of this architecture is the use of unique policy-based adaptation. The convergence of the entire network operation is also questionable since it does not encompass any coordination mechanism among high-level manager nodes.

Authors in [5] propose a hierarchical policy-based architecture with a centralized highest management level, called DRAMA. The architecture makes use of several proprietary protocols to distribute policies from the highest level down to the lower levels and for reporting management information to the higher levels reciprocally. However, this architecture is practicable only for small networks.

Autonomic Internet (AutoI) [6] proposed a two-level distributed hierarchical architecture: The lower-level deals with autonomic policy domain-wide management, while the high-level tier orchestrates between different autonomic management domains in order to accommodate the federation of networks. Both orchestration and management levels use policy-based adaptation mechanisms to face with network changes. The context information is collected according to a dynamic monitoring hierarchy and sent to a node that exploits this information. The monitoring hierarchy is setup and optimized by a centralized policy-based controller in accordance with the optimization requirements of management applications (e.g. cpu/memory, network resources, or response time). Although the AutoI architecture presents interesting features, its self-adaptation mechanism is limited by inherent insufficiencies of policy-based adaptation. Moreover, the centralized structure of its monitoring approach presents an issue against its scalability.

Authors in [7] presents the ADMA architecture, which is a distributed management architecture designed for MANETs. All network elements are involved equally to network management in a peer fashion. A policy management protocol distributes the predefined policies throughout the network with a minimum generated overhead. While this architecture presents a scalable policy distribution mechanism, the consistency and optimization of network operation are only based on the first definition of predefined policies by the network administrator, which is prone to human errors. Moreover, it does not specify any global view monitoring mechanism which may lead to an un-optimized network operation.

Authors in [8] uses several policy concepts to achieve autonomic behaviors in the network. The management guidelines are captured from human administrators and then expressed into network utility functions. These utility functions are then described in terms of the goal policies achieving the previous utility functions. The identified goal policies are forwarded to the coordinator node within a domain which derives the related behavioral policies, describing the behaviors that should be followed by network equipments to react to context changes and to achieve the given goal policies. Finally, these behavioral policies are sent to policy enforcement points within the domain which represents a switch, a router, etc. Although this architecture represents a promising approach to enable network elements deriving low-level tasks satisfying high-level management objectives, the effect of local decisions on the entire

network remains an issue.

Authors in [9] [3] proposed an agent-based and service oriented autonomic architecture, called Unity. It consists in a set of application clusters communicating to each other to optimize the overall system performance and fulfill user requirements. Each cluster consists in a set of application environments, each one being managed by an application manager. Within a cluster, a resource arbiter allocates resources to each application manager based on utility functions and high-level policies. The Unity architecture is more likely to converge to an optimal network resources usage since it relies on utility function. However, the architecture did not specify how different resource arbiters interact, which can become an issue regarding policy conflicts and overall network optimization.

Another trend aiming at proposing a self-adapting network architecture is addressed by cognitive networks [10]. It combines cognitive algorithms, cooperative networking, and cross-layer design in order to provide real-time optimization of complex communication systems. A cognitive network consists in a distributed architecture composed of a set of cognitive nodes exchanging cognitive information in order to enable a network-wide management. The proposed cognitive architecture allows dynamic reconfiguration of main protocol stack parameters at different layers for achieving performance goals driven by target quality metrics (e.g. overall packet delivery, end-to-end delay, etc.). To achieve this, a history of network operations with different settings is stored based on local and global views. Each protocol parameter P is continuously adjusted to the mean of the normal distribution of the value of P that provides the best performance under current network conditions. The main limitation of this approach is that it is only applicable for adjusting parameters taking values in a continuous interval.

To summarize, two main limitations can be identified in current autonomic solutions: first, the self-adaptation process is based on previous definition of either policies or utility functions which may deviate from the optimal solution specially when the network environment changes. The cognitive solution, which is the only architecture using a learning mechanism to enhance its configuration, does not consider all types of configuration parameters. Second, the employed monitoring approaches generate significant traffic overhead which may decrease the gain of self-adaptation processes. The proposed Learning Packet-Based Architecture tackles these issues by considering the overall optimization of network operation as its ultimate objective. This will be discussed in the next section.

3 LPBA - Learning Packet Based Architecture

The autonomic LPBA architecture consists in a cooperative distributed management layer composed of a set of network elements. Each network element represents an autonomic manager which manages directly its resources and implicitly the resources of other network elements. As depicted in figure 1, the distributed management layer is composed of two management entities implemented in each autonomic manager: The *Cognitive Engine* and the *Knowledge Management Engine*. The Cognitive Engine is responsible for reasoning based on local and network-wide knowledge. It represents the analyze and plan components of the MAPE-K model. The Knowledge Management Engine, on the other hand, is in charge of monitoring and processing the local and network-wide knowledge. With respect to the MAPE-K model, it represents the monitoring and knowledge components. In next sections, we describe the architectural details of each management entities.

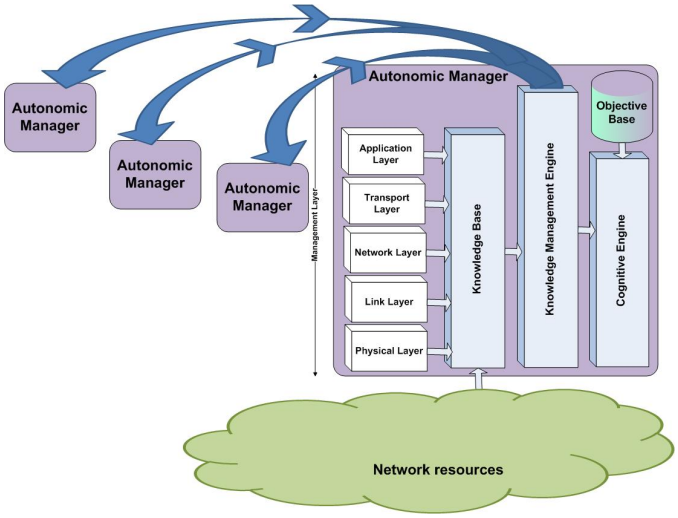

Fig. 1. LPBA architecture

3.1 Cognitive engine

The cognitive engine represents the decision making point of the architecture. It consists in two components:

- The *cognitive evaluation engine*, which continuously evaluates the performance of the underlying network based on aggregated knowledge provided by the knowledge management engine.
- The *cognitive reasoning engine*, which plans new decisions based on the analysis carried out by the cognitive evaluation engine.

The cognitive reasoning engine is based on the Random Neural Network (RNN) [11] and uses the reinforcement learning process (as explained hereafter) in order to adjust adaptation plans. Indeed, the RNN is a biologically inspired neural network model which is characterized by the existence of positive (excitation) and negative (inhabitation) signals defining the state of each neuron. Each neuron j is connected to other neurons i and sends to them excitation or inhabitation signals with a certain rate. These rates, noted respectively w_{ji}^{+} and

w_{ji}^{-}, represent the weight metric of each connection and should be learned from input data. In a RNN, the state q_i is the probability that the neuron i is excited. It can be calculated as follows:

$$q_i = \frac{\lambda^+(i)}{r(i) + \lambda^-} \quad (1)$$

where $\lambda^+(i)$ and $\lambda^-(i)$ are respectively the total excitation and inhabitation signals that the neuron i receives from other neurons j. $r(i)$ represents the total firing rate from the neuron i. These parameters can be calculated as follows:

$$\lambda^+(i) = \sum_{j \in\ neurons, j \neq\ i} q_j w_{ji}^{+} \quad (2)$$

$$\lambda^-(i) = \sum_{j \in\ neurons, j \neq\ i} q_j w_{ji}^{-} \quad (3)$$

$$r(i) = \sum_{j \in\ neurons, j \neq\ i} [w_{ij}^{+} + w_{ij}^{-}] \quad (4)$$

A decision corresponds to the selection of the most excited neuron in the RNN, i.e. the neuron i with highest value of q_i. Based on feedbacks provided by the cognitive evaluation engine, the Reinforcement Learning process adjust the weights to reward or punish a previous decision: if it reasons that the previous decision was successful, then the excitatory weights going into that neuron are significantly increased and the inhibitory weights leading to other neurons are slightly increased. Otherwise, all excitatory weights leading to all neurons except that corresponding to the previous decision are moderately increased, and the inhibitory weights lading to the previous wining neuron are significantly increased.

In the context of our architecture, we correspond a special RNN for each decision problem. Each neuron within a RNN represents a potential choice (value) for that decision problem. The RNN continuously seeks the most appropriate choice for each network conditions. For instance, we can consider a RNN for managing the key length of the cryptography defense line, a RNN for managing a particular parameter of the routing protocol, a RNN for managing the evolution of congestion window when a TCP application is running, etc.

As an example of neurons within a decision problem, we can imagine a RNN with three neurons or more for the case of cryptography security management: a neuron for representing each of 128-bit, 256-bit, 512-bit key lengths, etc. If the node detects that there were a lot of successful attacks using the last key length, it means that the last decision was not successful considering for the current network conditions. If it reasons that the last key length is good enough to counter potential attacks in current situation and the Quality of Service (QoS) performance is not significantly decreased, it enforces that last decision. If it detects that there is not necessary to have high key length which may decrease the QoS performance, it should also react on that accordingly.

In order to manage the inter influence of decisions made by inter dependant decision problems, we propose to define only one RNN representing conjointly those decision problems, where neurons corresponds to possible combination of choices among them. As an example, we consider that the performance of a routing protocol depends on the value of two protocol parameters, $P1$ and $P2$, which should be adjusted according to network conditions. $P1$ can select a choice between v_1, v_2 and v_3 while $P2$ could be either w_1 or w_2. The corresponding RNN represents the combination set of these choices which are (v_1, w_1), (v_1, w_2), (v_1, w_3), (v_2, w_1), (v_2, w_2), (v_2, w_3).

The cognitive evaluation engine evaluates the network operation based on the knowledge provided by the knowledge management engine. The performance of the last decision is evaluated based on objectives of the executed RNN either regarding one application (e.g. the optimization of TCP throughput) or regarding the overall network performance (e.g. the performance of the overall network regarding the packet delivery ratio, mean delay of real-time applications, etc.). The evaluation process compares the performance of the network before and after applying the new decision, which will guide the reinforcement learning process as described earlier.

3.2 Knowledge management engine

The knowledge management engine represents the second entity of the proposed LPBA architecture. Each node implements an instance of the knowledge management engine which performs monitoring and constructs a correct network view according to the requirements of the cognitive engine (decision-making RNNs). As most of autonomic architectures, we consider two types of knowledge: internal (local) view and external view. The internal view is obtained from monitoring local resources or/and cross-layer information. The external view represents the knowledge of a subset of network resources required for decisions with a more global scope. Such a metric could be energy level, load or neighbor degree, just to mention a few. In order to avoid significant overhead for acquiring the external view, any data or signaling packet traveling in the network will be used. For instance, a packet can carry information on metrics such as energy level, load or lifetime of links, and update it continually when visiting more nodes. The external view of each intermediate node will be updated for the region from where the packet arrived.

4 Case study: Routing in mobile ad hoc networks

As a proof of concept, we have applied the LPBA architecture for optimizing the performance of mobile

ad hoc routing protocols. The target objective of the management decision problem is the optimization of data delivery within the network. To do so, we implemented the cognitive and knowledge management engines under the network simulator ns-2 and provided necessary interfaces to link them to the routing agent. For ease of application, the Cognitive Packet Network (CPN) protocol [12] is selected as routing protocol which uses inherently the RNN in the route discovery process of fixed networks. This similarity simplifies the adaptation of the protocol to be linked to the cognitive engine component of our architecture. To do so, all decision process within the routing protocol is leaved to the cognitive engine. Moreover, the protocol is assisted by the knowledge management engine which feeds the decision process. Evidently, any other routing protocol can be adapted to use the features of the LPBA architecture following a similar process.

Each node of the ad hoc network is equipped with an instance of cognitive and knowledge management engines. The objective is to select routes offering a better solution for a given metric, such as route lifetime, delay, etc. To do so, each node has to select the more appropriate next hop for route request packets. This phase is ensured by the cognitive engine based on RNN when sufficient information exist in a node. Otherwise, broadcasting is used until the destination or a next hop with sufficient information to execute the RNN algorithm is reached. Within the cognitive engine of each node, a RNN is defined/updated by any route request packet for each pair of destination and QoS metric request. Each RNN is composed of N neurons, each one representing a neighbor (a potential next hop). The neighbor representing the most excited neuron is selected as the next hop. The knowledge is updated regarding the metric of choice based on the knowledge management process. For instance, the hop count, route lifetime and delay are used in our simulations. A packet updates the metric in all intermediate nodes towards its destination (which can be the destination of a flow for route request and data packets and the source for ACK packets). To illustrate this, let us consider the route lifetime as the metric of choice. When a node has a packet to send to a destination for which it has no fresh route in its cache, it generates a route request packet with the lifetime metric initiated. If a RNN exists already for that destination in the cognitive engine of the corresponding node, that RNN will be executed to find the more appropriate next hop. Otherwise, the route request packet will be broadcasted. In the next hop, the local node will be added to the route carrying by the route request packet. In addition, it updates the partial route lifetime considering its distance with the last hop, its transmission range and their relative speeds. Then, it runs again the RNN to find the next hop and so on. When the route request is received by the destination, an ACK is generated back to the source. This ACK updates the view of the intermediate nodes regarding the success of the last destination. If the QoS metric corresponding to the same pair of destination and QoS metric request is enhanced based on the last decision, the neuron corresponding to that decision will be rewarded, otherwise, that neuron will be punished according to the reinforcement learning process. The ACK packet is used as well to update the cache of intermediate node regarding a route towards the same destination with a same QoS metric. Note that the knowledge about route lifetime stored in the knowledge base can also be used to temporize the route in the cache.

4.1 simulation environment

Simulation experiments were performed using the network simulator ns2. Table 1 summarizes some simulation parameters. The scenario consists of 50 nodes with a transmission range of 250m in a 1300m x 500m area. Nodes are moving in random directions according to the Random Waypoint Model with maximum speed varying from 5 m/s to 30 m/s. The pause time is varied from 0s to 250s in order to show the impact of mobility on the network performance. We considered 20 CBR data flows in the network, each one generating 4 packets/sec with a packet size of 512 bytes. The performance of the CPN protocol using our architecture (denoted as LPBA) is compared with AODV, DSR protocols which are based on normal IP architecture. Simulations are repeated 20 times.

TABLE 1
Simulation parameters

Parameter	Value
Transmission Rate	11 Mbps
Radio Proagation Model	TwoRay Ground
Mobility Model	Random waypoint
Network area	1300m x 500m
Transmission Range	250m
Interface queue size	64 packets
CBR rate	4 packets/seconds
CBR sources	20
Data packet size	512 bytes
Maximum speed	1, 2, 10, 20, 30 m/s
Pause time	0, 50, 100, 250s
Simulation time	500s

4.2 Simulation results

We first evaluate the QoS performance of our architecture compared to the AODV and DSR routing protocols. We considered two QoS metrics: the Average End to End Delay (AE2ED) which represents the transmission delay of data packets that are delivered successfully, and the Packet Delivery Ratio (PDR) which is the ratio of the delivered data packets to the total number of data packets sent.

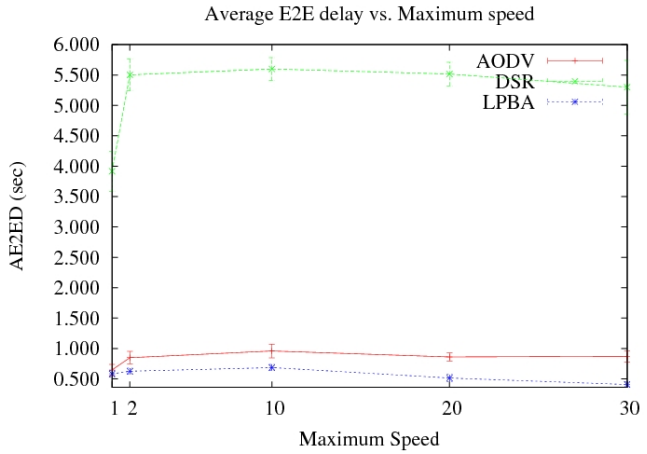

Fig. 2. End-to-End Delay vs. Maximum Speed

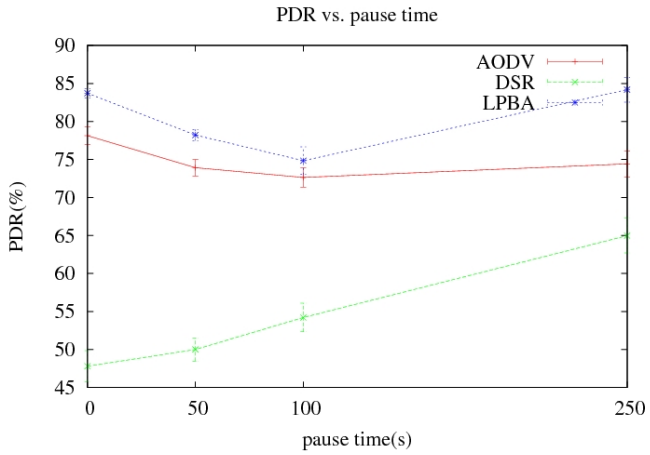

Fig. 3. PDR vs. Pause time

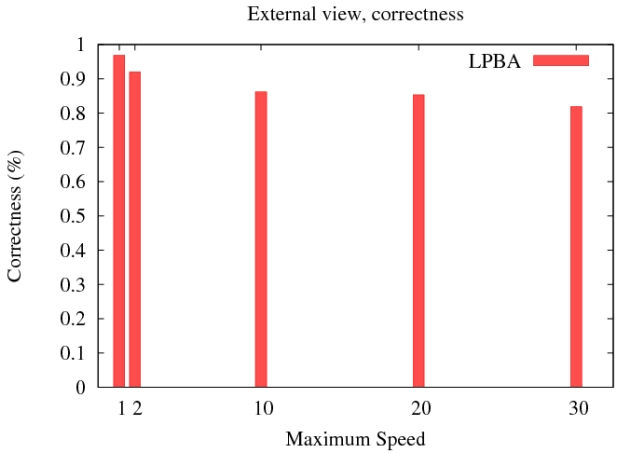

Fig. 4. Global view correctness

Figure 2 plots the average end-to-end delay of LPBA architecture compared to AODV and DSR protocols for different mobility speeds. The results are shown for a pause time of 250s. Other tested scenarios yield similar results. As depicted in this figure, the LPBA architecture presents a better delay for all maximum speeds. Indeed, the average end-to-end delay is reduced by around 650 ms compared to AODV which represents an enhancement up to 76%. Compared to the DSR protocol, the average end-to-end delay is enhanced up to 96%. In addition, we can see from that figure that the impact of the LPBA architecture becomes more relevant as the nodes' mobility increases.

In figure 3, the packet delivery ratio is analyzed for different pause times with the maximum speed fixed to 30 m/s. The results show that the LPBA architecture outperforms AODV and DSR protocols for all scenarios up to 10% and 37%, respectively.

In order to show the effectiveness of the packet-based knowledge management engine, we also evaluated the quality of the external (global) view using the correctness metric. This latter metric consists in calculating the fraction of nodes which have a correct view about the network when their external view is compared against the reality of the underlying network. In our case, the disseminated information used to construct the external view is the route lifetime. Hence, the correctness metric represents the ratio of nodes having a correct view about their links towards different destinations in the network. Figure 4 shows the correctness of the external view for different simulation parameters. As we can see, the correctness slightly decreases for higher dynamic scenarios but always remains higher than 83%. This degree of correctness can sufficiently be used for the decision process, as shown in QoS performance results depicted previously in figures 2 and 3.

5 Conclusion

In this paper, we presented LPBA, a learning Packet-based Autonomic architecture which uses a random neural network based on a cognitive engine with a reinforcement learning mechanism. The use of random neural networks enables the network to converge to the optimal configuration by enabling to learn from the past experiences to enhance its future operation. The proposed cognitive engine is fed by internal and external views which are provided by the knowledge management engine. The latter uses the normal transiting packets in the network to disseminate the necessary global views. As a case study for the proposed architecture, a modified version of the Cognitive Packet Network protocol was proposed which uses the proposed cognitive and knowledge management engines.

Simulations carried on the ns-2 simulator under different mobility scenarios show that the results are improved significantly for the end-to-end delay and for the packet delivery ratio when comparing with the AODV and DSR protocols. In addition, the high quality of the external view was evaluated which shows that

TABLE 2
Overview of some proposed architectures

	DRAMA	CA-MANET	AutoI	ADMA	ANEMA	Unity	Cog-Prot	LPBA
Decision process	Policy-based	Policy-based	Policy-based	Policy-based	Goal policy-based	Utility function + Policy-based	a Primary cognitive process	RNN + reinforcement learning
Learning	NO	NO	NO	NO	NO	NO	YES	YES
Cost of autonomicity (overhead)	Medium	Medium	Medium	Not specified	Medium	Medium	Not specified	Low
Optimal reasoning	NO	NO	NO	NO	NO	YES	YES	YES

the proposed knowledge management engine based on packets in the network can give a sufficient good image about the network as they represent the most mobile element in the network. The characteristics of existing autonomic network architectures as well as the proposed LPBA architecture are summarized in table 2.

As future work, we intend to model other management decision problems using the LPBA architecture. For instance, we will focus on the security related configuration management. Moreover, we will define a reference application with several inter-dependent management problems and study the effectiveness of our architecture to optimize the overall solution. We will also examine the performance of the LPBA architecture comparing to other autonomic architectures.

ACKNOWLEDGMENTS

REFERENCES

[1] S. Calo and M. Sloman, "Guest editorial: Policy-based management of networks and services," *Journal of Network System Management*, vol. 11, no. 3, pp. 249–252, 2003.

[2] A. K. Bandara, E. Lupu, J. D. Moffett, and A. Russo, "A goal-based approach to policy refinement," in *POLICY*, 2004, pp. 229–239.

[3] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Computing*, vol. 11, no. 1, pp. 40–48, 2007.

[4] Z. Movahedi, M. Abid, D. Fernandes Macedo, and G. Pujolle, "A policy-based orchestration plane for the autonomic management of future networks," in *6th International Workshop on Next-Generation Networking Middleware (NGNM) as part of Manweek 2009*, October 2009.

[5] R. Chadha, Y.-H. Cheng, J. Chiang, G. Levin, S.-W. Li, and A. Poylisher, "Policy-based mobile ad hoc network management for drama," *MILCOM Journal*, vol. 3, pp. 1317–1323, December 2004.

[6] "Autonomic Internet (AutoI) FP7 project." [Online]. Available: http://ist-autoi.eu/

[7] M. Ayari, Z. Movahedi, F. Kamoun, and G. Pujolle, "Adma: Autonomous decentralized management architecture for manets - a simple self-configuring case study," in *Prodeedings of the International Wireless Communications and Mobile Computing Conference (IWCMC), Autonomic Wireless Networking Workshop*. Leipzig, Germany, June 2000, pp. Leipzig, Germany.

[8] H. Derbel, N. Agoulmine, and M. Salaün, "Anema: Autonomic network management architecture to support self-configuration and self-optimization in ip networks," *Computer Networks*, vol. 53, no. 3, pp. 418–430, 2009.

[9] D. M. Chess, A. Segal, and I. Whalley, "Unity: Experiences with a prototype autonomic computing system," in *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 140–147.

[10] R. W. Thomas, L. A. Dasilva, and A. B. Mackenzie, "Cognitive networks," in *Proceedings of IEEE DySPAN 2005*, November 2005, pp. 352–360.

[11] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural Comput.*, vol. 1, pp. 502–510, December 1989. [Online]. Available: http://portal.acm.org/citation.cfm?id=1351079.1351086

[12] G. Sakellari, "The cognitive packet network: A survey," *Comput. J.*, vol. 53, no. 3, pp. 268–279, 2010.