Distributed Decision Engine – An Information Management Architecture for Autonomic Wireless Networking

Markus Luoto, Teemu Rautio, Tiia Ojanperä and Jukka Mäkelä
VTT Technical Research Centre of Finland
Kaitoväylä 1, 90100 Oulu, Finland
Email: firstname.lastname@vtt.fi

***Abstract*—With wireless networks becoming ever more ubiquitous and the capabilities of the devices connected to them rivalling those of desktop computers, there is an acute need for the communications in these networks to be managed as efficiently and autonomously as possible. Moreover, as the number of connected devices is rapidly increasing, the problem is becoming all the more complex and the amount of up-to-date status information needed for the management is multiplying. In order to meet the needs of future network management in terms of knowledge building and dissemination, this paper presents an architecture for collecting, processing, and disseminating vast amounts of network management information efficiently in wireless networks. The information may originate from the different layers of the networking protocol stack either on the wireless clients or network devices, and the architecture facilitates its signalling within a distributed network management system. Also, as described in this paper with practical examples, the architecture has been validated in a laboratory with real devices in the context of multiple use cases ranging from enhanced multimedia delivery for mobile clients to balancing the workload of network nodes. In addition, this paper evaluates the performance of the Event Cache prototype in terms of its capability of handling event dissemination in a real testbed environment.**

I. Introduction

Providing satisfactory Quality of Experience (QoE) cost-efficiently to wireless clients in the future will require highly optimized usage of network resources. As wireless networks are diverging into multiple access technologies with overlapping coverages and providing a diverse set of features, the optimization of this system presents a complex problem. The problem is further aggravated on one hand by the evolution of very powerful handsets able to stream high resolution video content in both up- and downlink directions, and on the other hand by the emergence of Internet of Things (IoT) and the multitude of connected devices it brings along.

Nevertheless, these challenges also provide new possibilities for network management. The diverse operation environment and the rich set of wireless devices provides a basis for a very productive optimization of the resource usage. By automatically learning from past experience the characteristics and usage patterns of the connected devices can be somewhat predicted. This will enable better QoE with a lower cost. All this can be done with cognitive network management, which has been widely proposed as a solution to combat complexity and replace the current systems relying on human expertise or simple self* mechanisms. This further automation of the network management operations will be the key to building viable networks in the future.

In order to function effectively and reliably, cognitive network management requires vast collaboration between the devices connected to the network as the best sources of information are scattered throughout the network and ranging from the core devices to the wireless end nodes. One key enabler for the distributed cognitive network management is to solve the issue of simple and efficient dissemination of information. This calls for a thought out architecture designed to disseminate diverse information of the different aspects of the network conditions from different perspectives.

In this paper, we present the Distributed Decision Engine (DDE) – an architecture designed to disseminate information and manage multiple optimization functions in a controlled and coordinated manner in a cognitive network management system. We describe how DDE is able to collect, process, and disseminate vast amounts of network management information efficiently. Furthermore, we also present the evaluation results for performance of the DDE prototype in terms of its capability of handling event dissemination in a real testbed environment. Finally, we present with practical examples, how the architecture has been validated in the contexts of enhanced multimedia delivery for mobile clients and balancing the workload of network nodes.

The rest of the paper is organized as follows. Section II gives an overview of the related work. Section III presents the concept of DDE. The main entity in the DDE architecture, namely Event Cache, is discussed in detail in section IV. The current prototype of the system is presented in section V and experimentation results are discussed in section VI. Finally, section VII concludes the paper.

II. Related work

The work described in this paper falls mostly in the category of autonomic or cognitive network management. The majority of approaches in this area are based on the principles of autonomic computing published by *Kephart et al.* in [1]. The main principles are self-management by self-configuration, self-optimisation, self-healing and self-protection as well as the MAPE-K autonomic control loop consisting of monitoring, analysing, planning and execution phases linking to a knowledge-base. As with related research, these influences can also be found in our work.

The work on autonomic or cognitive network management can be roughly divided in to evolutionary and revolutionary approaches. As an example of the revolutionary approaches, the most ambitious proposals have aimed for a completely redesigned network architecture such as the clean slate 4D architecture proposed by *Greenberg et al.* in [2] or the 4WARD architecture proposed by *Niebert et al.* in [3]. In many of the more recent proposals, such as the GARSON architecture proposed by *Kuklinski et al.* in [4], the Unified Management Framework proposed by *Tsagkaris et al.* in [5], and the SEMAFOUR vision described by *Litjens et al.* in [6], the authors have taken a more evolutionary path and the approach is more virtual. As such, the architecture can be built on top of current infrastructure and is thus much easier to implement in reality. Our approach also follows this evolutionary path.

In addition to full scale network management solutions our work is also closely related to other frameworks also mediating information and taking advantage of cross-layer design such as Media Independent Handover (MIH) [7]. Presented DDE concept and its functionalities follows the principles of the complex event processing (CEP) [8] concept. CEP is introduced as a set of techniques and tools to control event driven information systems and usually consists a mix of different mechansims, old and new. CEP tries to help to understand and control different event-driven information systems.

The implementation choices made in our work have been aimed at providing fast prototyping and easy portability for research purposes. As such, we have mostly opted to use the Python [9] scripting language and simple socket-based interfaces for communication between entities instead of choosing a more efficient language such as C/C++ [10] or a standard messaging system such as D-Bus [11].

III. Distributed Decision Engine Concept

The Distributed Decision Engine, depicted in Figure 1, is a concept that enables building a cognitive network management system with components ranging from simple information producers and actors to cognitive algorithms. The DDE concept comprises of three kinds of entities, namely (a) the Event Producers, which feed DDE with information; (b) the Event Consumers, which receive information in the form of events they have subscribed; and (c) Event Caches that orchestrate this information exchange and can be interconnected in cascade fashion. An algorithm is considered to be both an Event Consumer for its inputs and an Event Producer for its outputs in the DDE concept.

The main entities in the system are the Event Caches which enable short time storage and delivery of events between different entities. The Event Cache has been designed to implement a distributed publish-subscribe message delivery system and defines a common interface for different information producers, decision algorithms and actors. It also implements an information caching functionality, in order to keep the latest information available and up-to-date, as well as information processing for minimizing the data to be sent over the network. Furthermore, it can perform information filtering based on policies.

In comparison to TRG [12], from which the concept of DDE has evolved, the design of DDE has improved on

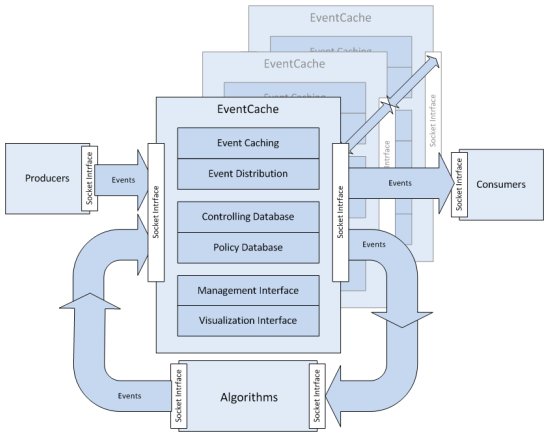

Fig. 1. Distributed Decision Engine

many levels. For example, the message format in DDE is much more flexible, enabling a more diverse set of use cases. Also, more security aspects have been taken into account in development of DDE from the very beginning. For example, all the messaging in DDE is signed with digital signatures. In addition, the DDE also has a management and a visualization interface defined as part of the Event Cache to enable better control and management of the system.

As an example of basic message exchange in DDE, a short sequence diagram is depicted in Figure 2. This example includes all the main types of entities in a DDE system and simple messaging scenario between them. In this scenario, the Event Cache facilitates a message exchange in which the Algorithm requires a piece of information from the Event Producer (Event 1) to calculate another piece of information (Event 2) that Event Consumer is interested in.

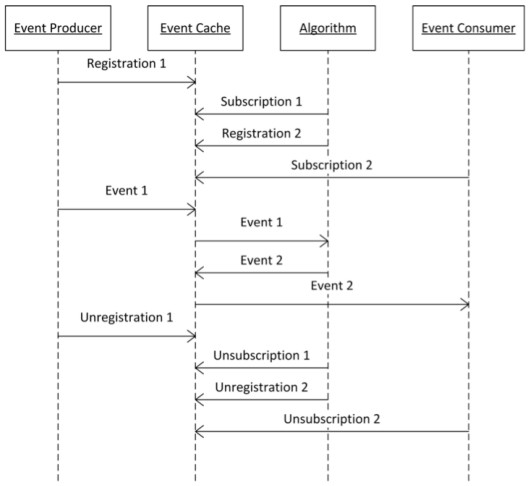

Fig. 2. DDE signalling

IV. DDE Event Cache

The Event Cache is the key component of the DDE concept as all the other entities use the services of the Event Cache for communication. In this section, we describe the services it provides in more detail.

TABLE I. MESSAGE FORMATS IN DDE ARCHITECTURE.

Type of Message	**Fields of the message**							
Registration	Message Type	Producer ID	Producer Name	Event ID	Event Name	Digital Signature		
Subscription	Message Type	Consumer ID	Consumer Name	Consumer Address	Event ID	Type Filter	Producer Filter	Digital Signature
Event	Message Type	Producer ID	Event ID	Event Type	Time-to-Live	Payload Length	Payload	Digital Signature
Policy	Message Type	Updater ID	Event ID	Consumer ID	Producer ID	Digital Signature		

A. Messaging

The messages in DDE are encoded using XDR [13]. Information fields in different messages are shown in Table I. As shown in the table there are four types of messages used in DDE, namely Registration, Subscription, Event and Policy messages. The message type field in each message is an integer that denotes the type of the message to the Event Cache.

To be able to send events, a producer sends a Registration message identifying itself to the Event Cache. Respectively, when leaving the system, the producer sends a Registration message with different Message Type to inform the Event Cache of this. The Producer ID field in the Registration message represents the public key of the producer and the Event ID field is an integer identifying the registered event while producer name and event name fields are human readable information about the producer and event. Additionally, there is a digital signature to verify the message. A consumer can order events with a Subscription message from the Event Cache. In addition to fields similar to a Registration message, a Subscription message also contains the address in which the consumer listens for the events it subscribed, two filter fields to restrict the type and producer of the event.

Event messages carry the actual information disseminated by DDE. An Event message includes an Event Type field which specifies a subtype of a certain Event ID, a Time-to-Live field which contains the validity time of the event in seconds and naturally the Payload Length and Payload fields. The contents and the purpose of a policy message is considered in Section IV-C.

In the DDE concept, a typical use case for collecting and distributing events is to distribute the functions among different network entities. Consequently, the events will occasionally be delivered over the network, which raises concerns of malicious modifications or creation of events. Therefore, DDE builds trust between different entities by supporting self-certifiable Event, Registration, Subscription and Policy messages by following similar principle as for information in [14].

In practise, this is implemented by digital signatures in the respective DDE messages. A public key of the message origin is carried in the corresponding field of the message. For example, in a case of event, producer id stores public key of that information producer, which can be then used to verify that the message content really corresponds to the signature, i.e. all information needed to verify message integrity is carried by the message itself. In addition, since the public key is included into message (as an identifier), also the message origin can be confirmed. Basically, another entity cannot create verifiable message for the certain public key unless it also knows the private key. As said, DDE has self-certification function for registration, subscription, event and policy messages.

The minimum size of the Event messages and the related overhead in DDE can be calculated with the values in Table II. As can bee seen from the table, for a message with a single element payload the minimum message size is effectively 164 bytes in addition to payload and it grows by 4 bytes per every additional element in the payload. Nevertheless, it must be noted that when using a Producer/Consumer/Updater ID of realistic size, their size will become dominating compared to static reservation of other fields in all the messages within DDE. Additionally, the Payload in the Event message dominates the message size in most normal usage scenarios and the overhead from the other fields becomes negligible.

B. Management

The main configuration options of the Event Cache can be set through a configuration file. The basic options include parameters such as the address and port on which to listen for messages on and the address for sending Visualization messages. Also, more advanced options such as cascading multiple Event Caches through cascade registrations and subscriptions between Event Caches, can be configured in the configuration file. Finally, the configuration file also includes security parameters such as allowed originating ID for policy updates, whether or not a signature check is enforced on different messages and preferred algorithms and key lengths for message signing.

The Event Cache includes an embedded web server providing a run-time user interface for its management purposes. Through the user interface, the operator of the Event Cache can view details about the current information in the Event Cache, such as registrations, subscriptions and policies. Each registration, subscription or a policy that exist in the Event Cache, may be added, edited or removed through the user interface. Editing, for instance, filters of the existing subscription on the fly can be used as a very practical tool in testing the performance of an intelligent algorithm with various inputs. In addition, the contents of an individual database may be flushed by clicking a single button.

By default, the Event Cache saves the information contained in its main databases also to files and loads the information stored in the files when starting up. This allows the Event Cache or the node running the Event Cache to be restarted during operation without the need to send again the registrations, subscriptions and policies contained in the databases.

C. Policies

Policies used in the DDE and enforced by the Event Cache are access policies. If the policies are enabled in the settings, they are permissive by nature. This means, that all event communication is prohibited unless explicitly permitted by a policy. These policies are communicated to the Event Cache via Policy Messages, see structure in Table I.

TABLE II. SIZE OF A DDE EVENT MESSAGE.

Field	Message Type	Producer ID	Event ID	Event Type	Time-to-Live	Payload Length	Payload	Digital Signature
Type	integer	string	integer	integer	integer	integer	array with n elements	string
Size	4 bytes	4+64 bytes (minimum)	4 bytes	4 bytes	4 bytes	4 bytes	4*n + payload bytes	20+4+48 bytes (minimum)

Policies may be added or deleted by a message with the specific Message Type. Updater ID indicates the identity of the updater in the same way as a Producer ID does in an Event Message. Events affected by the policy are specified by the Event ID. Producer and Consumer IDs provide a way to target the policy to allow just certain producers and consumers. For example, that a specific producer can (or cannot anymore) send events with that Event ID to the Event Cache; or that a specific consumer can now on receive events with a specific Event ID. The policy message is proofed by a digital signature to ensure message integrity. A producer or a consumer can always try to make a registration or a subscription to an Event Cache, but the absence of a policy allowing the action leads to the Event Cache discarding the incoming message from the producer, or prevents the event from being sent to the consumer.

At the time of writing, only policies at the consumer end are supported in the prototype but nothing prevents the support to be easily extended also to the producer side. In practise, consumers cannot obtain events before a policy is added for that specific event and for that specific consumer. Policy messages are only accepted from an authorised policy updater, which is specified by the Updater ID in the Event Cache configuration file (and seen in the policy message as well). The updater can be either the same operator of the Event Cache that is able to control the contents of the databases also through the web-based management interfaces, or it can be a custom built autonomic algorithm implemented for that purpose.

V. PROTOTYPING

This section gives a short description of the current prototype implementation we have of the DDE architecture. The description focuses mainly on the Event Cache as it forms the backbone for the whole architecture. We also present our visualization implementation as it is an essential tool in understanding the prototype and the whole architecture.

A. Prototype

The main component of the framework, that is, the Event Cache is implemented in Python, and while most of the development is done in Linux, it can also be run in Windows and OSX environments. There are reference implementations of Event Producer and Event Consumer in Python, C and C++, as well as, a reference implementation of an Algorithm in Python, which connects to the DDE both as a consumer for its inputs and as a producer for its outputs.

The reference implementations of the Event Producer, the Event Consumer and the Algorithm are also modular by design. As such, they are easily customized for various purposes. The Producer and Consumer also have simple functionality that allows them to be used as debugging tools configured to send or receive and print Events via command line arguments or configuration files.

B. Visualization

In order to visualize the operation of the DDE architecture for demonstration and evaluation purposes, visualization software has also been developed and has proven to be indispensable in understanding and debugging as well as demonstrating the DDE functionality in a laboratory environment. The latest generation of the software, shown in Figure 3, is capable of showing a system of two Event Caches and their clients. The visualization tool is also implemented in Python using Cocos2d and Pyglet visualization libraries and they get the information to be visualized through the visualization interface of the Event Cache.

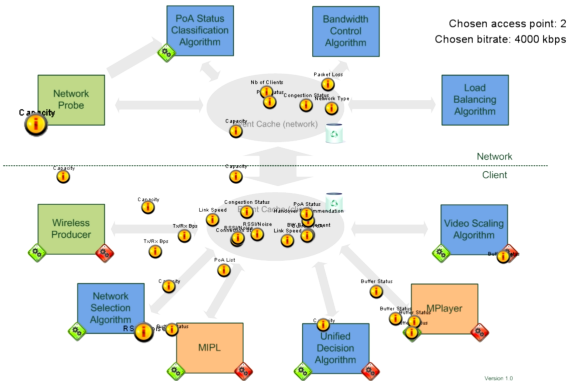

Fig. 3. Visualization of a system with two Event Caches in cascade.

Contents of the different visualization messages provided by the Event Cache Visualization Interface are listed in Table III. This table provides insight on what is possible to be visualized with the current version of the interface. The visualization message fields Name, Sender ID and Receiver ID are optional in the sense that they only carry values when appropriate. The Name field can contain a human readable name string of the visualization element and the Sender and Receiver IDs are also used as Producer and Consumer IDs respectively when appropriate.

TABLE III. VISUALIZATION MESSAGES

Meaning	Type	Name	Sender ID	Receiver ID	Event ID	Event Name
add producer	7	Value	Value	-	Value	Value
delete producer	8	Value	Value	-	Value	Value
add consumer	7	Value	-	Value	-	-
delete consumer	8	Value	-	Value	-	-
event from producer	7	-	Value	-	Value	-
event to consumer	7	-	-	Value	Value	-
event to history	8	-	-	-	Value	-

VI. EXPERIMENTS

As we have a working prototype of the DDE architecture, we have been already able to apply the architecture to multiple use cases in a laboratory environment and evaluate the performance. We present the newest measurements on performance of the Event Cache prototype in VI-A. Two of the use case

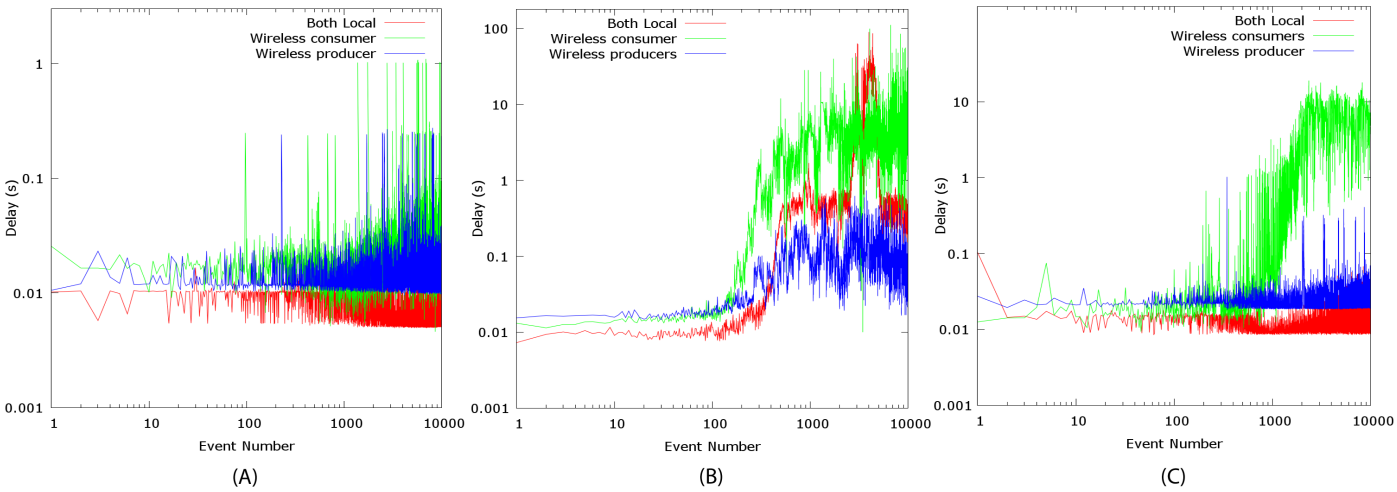

Fig. 4. Event delivery delays in laboratory measurements

evaluations are also summarized shortly in this section, namely a use case related to adaptive media delivery in VI-B and a use case related to load balancing in VI-C.

A. Event Cache

The Event Cache facilitates communication between all the components in the DDE architecture and as such is the most critical component in the system. In the following the Event Cache is evaluated in terms of scalability and reliability. The testing was done with two 2.6GHz computers with 8GB of RAM running Ubuntu 12.04 LTS connected to a single 802.11g Cisco Aironet 1242AG Wi-Fi access point, although in terms of performance the tests could have been run on a much lighter hardware as the Event Cache is not resource intensive. The Event Cache is located on one of the computers and both computers have ten event producers and ten event consumers, which are activated as required by the test cases. The testing of the Event Cache performance was carried out with 9 different test cases listed below:

1) 1 local producer and 1 local consumer
2) 1 local producer and 1 Wi-Fi consumer
3) 1 Wi-Fi producer and 1 local consumer
4) 1 local producer and 10 local consumers
5) 1 local producer and 10 Wi-Fi consumers
6) 1 Wi-Fi producer and 10 local consumers
7) 10 local producers and 1 local consumer
8) 10 local producers and 1 Wi-Fi consumer
9) 10 Wi-Fi producers and 1 local consumer

The aim for these tests was to assess the ability of the current Event Cache prototype to handle different types of event loads in different situations. In each case the producer(s) were producing 10000 events with generation intervals between 1 to 0.001 seconds according to the formula $\frac{1}{n} = t$, where n is event number and t time delay after the previous event. In the measurements, the time for the event to reach the consumer(s) was logged. Although the actual spacing between events in time does not strictly follow the formula as the generation and sending of the event also takes a small amount of time, this testing gives valuable information on the scalability and the reliability of the current Event Cache prototype. The following presents the evaluation results by grouping the nine test cases in three groups with similar cases.

Cases 1-3: In cases 1-3 single producer and consumer were tested both locally and wirelessly over a Wi-Fi (802.11g) link. Figure 4 (A) shows the delay experienced by the events. All the events were delivered in cases 1 and 2. In case 3 one event out of the 10000 was lost.

Cases 4-6: In cases 4-6 10 simultaneous producers and a single consumer were tested both locally and wirelessly over a Wi-Fi (802.11g) link. Figure 4 (B) shows the delay experienced by the events. The tests show, that up until the 10 producers are sending 20 events per second each (around event number 200 in the figure), the delay behavior is consistent and all the events are delivered. After this point, as the event flow increases; the delay behavior of the Event Cache becomes more erratic which can be seen from the delay figure. In addition, events are lost with a rate of 15% in case 4, 78% in case 5 and 2% in case 6. This behavior suggests that problems originate from the fact that Event Cache isnt able to process the events fast enough as the number of events lost is the smallest in the case that the producers are behind the Wi-Fi link and as such are not able to overload the Event Cache as much.

Cases 7-9: In cases 7-9 a single producer and 10 simultaneous consumers were tested both locally and wirelessly over a Wi-Fi (802.11g) link. Figure 4 (C) shows the delay experienced by the events. In case 7, with both local producer and local consumers, all the events get delivered and the delay behavior is logical. Case 9, with the wireless producer, gives similar results with only 30 (out of 100000) lost events and quite even delays. Case 8, with the 10 wireless consumers, has over 50% packet loss and erratic delay behavior during the higher event flow. This is similar in behavior to case 5.

Based on these test we can conclude that both the Event Cache and the DDE architecture scale as expected in all cases except when a high volume of events must be sent to a consumer through a slow (wireless) link. However, this kind of situation would be impolitic in the DDE architecture and can as such be neglected, as the aim of the architecture is to

handle the event flow over wireless links by interconnecting two Event Caches and minimizing the number of sent events.

B. Adaptive media delivery

Internet-based multimedia services today incorporate flexible coding and transmission technologies, such as Scalable Video Coding (SVC) and adaptive streaming, in order to cope with heterogeneous and fluctuating network capacity. The decision-making in current solutions, including for example MPEG's Dynamic Adaptive Streaming over HTTP (DASH) and Apple HTTP Live Streaming (HLS), nevertheless, is solely based on application layer monitoring of media streaming performance. That is, based on metrics such as receiver buffer status or perceived media throughput. Additionally, the media stream may initially be adjusted to the theoretical capacity of the used access network technology (e.g. WLAN or mobile). This can be done by selecting the most suitable streaming rate ether manually by the user or automatically by the application during start-up.

Nevertheless, adaptive media streaming would benefit from more extensive cross-layer information in its decision-making, especially in heterogeneous wireless network environments. Also, automated management should be incorporated in order to free humans from any bitrate selection responsibility. Enabling such informed decision-making based on cross-layer and cross-domain metrics that reflect the actual transmission conditions as well as the multimedia application's requirements, nevertheless, calls for an efficient and extensive signalling framework.

The DDE framework has been utilized for providing the required knowledge building and decision-making for intelligent management of adaptive video streaming in a heterogeneous multi-access network environment in [15], [16]. For the management system presented in these papers, DDE allows for incorporating multiple terminal or network-side event sources and hierarchical decision algorithms for dynamic controlling of video streaming bitrate under network impairments and handovers. The prototype implementation of the proposed system uses a HLS-based video streaming client that was integrated with DDE for the required signalling. The experimental evaluation performed on the prototype in the two papers attested the benefits of using advanced cognitive decision algorithms for the controlling of the video streaming bitrate in the considered multi-access scenario. The management system was shown to improve the QoE for the video user as well as to enhance the stability and performance of the decision-making in the dynamic network environment. These results were enabled by the advanced signalling and event management capabilities of the DDE framework, which are not supported by the current technologies.

C. Load balancing

The extensive growth of mobile data traffic will most likely to be distributed inequally between network resources. Some of the network equipments and paths will be therefore used more than they can efficiently serve, whereas some of them are barely used at all. Today's end-user equipments are quite often equipped with multiple network interfaces, which could be potentially be used even at the same time if possible. But without any knowledge on current network conditions, for example, different content distribution systems can cause issues in wireless environment and experienced quality of other users.

DDE has been used for building multiaccess capable access selection and load balancing proof-of-concept prototype, as described in [17]. The prototype brought end-user a possibility to use multiple wireless network connections at the same time for file download, but was doing this without causing significant quality impairments to the other user, which was using one of access points for video streaming service. The prototype employed DDE for collecting information from various sources at different network locations, for delivering the information for a distributed classification system and for delivering the final classification outcome to be used as a basis for autonomic network management decisions for load balancing. With a help of DDE, the prototype outperformed the reference system, which was without the network awareness, in access point overloading situations. Moreover, the prototype proved that DDE can provide easily deployable common interface for distributing event between different scenarios in realistic real-time environments.

VII. Conclusion

As wireless networks have become ubiquitous and capabilities of the devices connected to those networks are rivalling the capabilities of desktop computers, the need for the communications in these networks to be managed as efficiently as possible has become imperative. One key aspect in this distributed cognitive network management is to the solve the issue of simple and efficient dissemination of information.

In this paper, we presented Distributed Decision Engine which is an architecture designed to disseminate information and manage multiple optimization functions in a controlled and coordinated manner in a cognitive network management system. We concluded that in our analysis both the Event Cache and the DDE architecture scale well considering the intended deployment architecture. The results from two tested use cases ranging from enhancing multimedia delivery for a mobile client to balancing the workload of the network nodes were also presented and these proved that DDE can provide easily deployable common interface for distributing events between different entities in realistic real-time environments.

As future work, the authors plan to investigate open issues such as the organization of the naming scheme for the events and their producers and consumers as well as conduct wider scaling studies in a real networking environment.

Acknowledgment

The work reported in this paper was partly supported by the Finnish Funding Agency for Technology and Innovation (Tekes) in the framework of the EUREKA/Celtic Cognitive Network Management under Uncertainty (COMMUNE) project and partly supported by the VTT Technical Research Centre of Finland in the framework of the AWARENESS: Energy Aware Learning in Cognitive Radios and Networks project. The authors would like to thank their colleagues who have contributed to the projects and especially those who have participated in the implementation of the prototype.

REFERENCES

[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[2] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[3] N. Niebert, S. Baucke, I. El-Khayat, M. Johnsson, B. Ohlman, H. Abramowicz, K. Wuenstel, H. Woesner, J. Quittek, and L. Correia, "The way 4ward to the creation of a future internet," in *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on*, Sept 2008, pp. 1–5.

[4] S. Kuklinski, M. Skrocki, L. Rajewski, J. Meseguer Llopis, and Z. Wereszczynski, "Garson: Management performance aware approach to autonomic and cognitive networks," in *Globecom Workshops (GC Wkshps), 2012 IEEE*, Dec 2012, pp. 914–918.

[5] K. Tsagkaris, P. Vlacheas, A. Bantouna, P. Demestichas, G. Nguengang, M. Bouet, L. Ciavaglia, P. Peloso, I. Grida Ben Yahia, and C. Destre, "Operator-driven framework for establishing and unifying autonomic network and service management solutions," in *GLOBECOM Workshops (GC Wkshps), 2011 IEEE*, Dec 2011, pp. 684–689.

[6] R. Litjens, F. Gunnarsson, B. Sayrac, K. Spaey, C. Willcock, A. Eisenblatter, B. Gonzalez Rodriguez, and T. Kurner, "Self-management for unified heterogeneous radio access networks," in *Vehicular Technology Conference (VTC Spring), 2013 IEEE 77th*, June 2013, pp. 1–5.

[7] IEEE, "Standard for local and metropolitan area networks - media independent handover services," *IEEE Std 802.21-2008*, pp. 1–0, Jan 2009.

[8] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[9] G. v. Rossum and F. L. Drake, "Python reference manual," Virginia, USA, 2001. [Online]. Available: http://www.python.org

[10] B. Stroustrup, *The C++ Programming Language*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[11] H. Pennington, A. Carlsson, A. adn Larsson, S. Herzberg, S. McVittie, and D. Zeuthen, "D-bus specification." [Online]. Available: http://dbus.freedesktop.org/doc/dbus-specification.html

[12] J. Makela and K. Pentikousis, "Trigger management mechanisms," in *Proceedings of the International Symposium on Wireless Pervasive Computing*, San Juan, Puerto Rico, February 2007, pp. 378–383.

[13] M. Eisler, "Xdr: External data representation standard," RFC 4506 (INTERNET STANDARD), Internet Engineering Task Force, may 2006.

[14] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *13th Global Internet Symposium, 2011 IEEE*, Mar 2010, pp. 1–6.

[15] T. Ojanperä, M. Luoto, M. Uitto, and H. Kokkoniemi-Tarkkanen, "Hierarchical management architecture and testbed for mobile video service optimization," in *ICNC 2014*, Feb 2014, pp. 999–1005.

[16] T. Ojanperä, M. Luoto, M. Majanen, P. Mannersalo, and P. Savolainen, "Cognitive network management framework and approach for video streaming optimization in heterogeneous networks," Submitted, 2015.

[17] T. Rautio, M. Luoto, J. Mäkelä, and P. Mannersalo, "Evaluation of autonomic load balancing in wireless multiaccess environment," in *IEEE Wireless Communications and Networking Conference (WCNC), 2013*, April 2013, pp. 1416 – 1421.