Monitoring as First Class Citizen in an Autonomic Network Universe

Martin May
ETH Zurich
Switzerland
maym@tik.ee.ethz.ch

Matti Siekkinen, Vera Goebel, Thomas Plagemann
University of Oslo, Dept. of Informatics
Norway
{siekkinen,goebel,plageman}@ifi.uio.no

Ranganai Chaparadza, Lorenzo Peluso
Fraunhofer FOKUS Institute
Berlin, Germany
{chaparadza, lorenzo.peluso}@fokus.fraunhofer.de

ABSTRACT

When developing a new autonomic networking architecture from scratch with monitoring as a first class citizen, a whole set of new requirements have to be addressed. The main reason for this is that no a priori knowledge about the network, the monitoring tasks, etc. is available in the architecture itself. Monitoring could be placed everywhere in the network and it must be possible for monitoring modules to explore the available monitoring support in its surrounding at runtime. Monitoring needs also to be dynamic, adaptive and programmable. This paper presents the new requirements and how these requirements on monitoring are addressed in the ANA architecture.

Keywords

Monitoring, autonomic networks, network architecture

1. INTRODUCTION

Traditionally, network monitoring is used as a tool for network management to constantly monitor the state of the network and to notify the network administrator in case of failures or other exceptional events. More advanced usages of network monitoring include for example Quality-of-Service monitoring and workload balancing in distributed systems. Independent of the particular use of network monitoring there is one common factor for all monitoring solutions: monitoring is added to the network after the network and basic protocols have been designed.

In the context of the recent quest for the design and architecture of the future Internet, the concept of autonomic networking and networks with self-* properties, like self-configuration, self-optimization, self-healing, and self-protection [4] are intensively studied. Obviously, monitoring is a fundamental part of such autonomic systems, without it no self-* properties can be achieved, because these tasks are performed by a kind of feedback control loop that is driven by monitored (observed) events or changes in variables. Monitoring is the element responsible for measuring the parameters of the controlled system that are relevant for the task under control (e.g., current load for performance optimization, link availability for fault tolerance, etc.). In the *Autonomic Network Architecture* (ANA) project [1], we develop an autonomic network architecture from scratch, with monitoring as first class citizen, meaning monitoring is as fundamental in the ANA architecture as other basic networking concepts, like addresses, naming, labeling, forwarding, routing, etc. This is a paradigm shift for monitoring. Classical monitoring solutions can be characterized as follows: they are designed for particular problems, the monitoring task is static, placement of monitoring probes is done manually with a priori knowledge about the network topology, they assume minimal support from nodes and protocols (e.g., active measurement), or they place the entire burden on the nodes (routers) to support heavy-weight mechanisms (e.g., NetFlow or passive measurements in general).

Anchoring monitoring as first class citizen in the ANA architecture puts a whole set of new requirements onto the monitoring concepts and principles that need to be supported in the autonomic network architecture. The main difference between today's solution and the ANA approach for monitoring is that the monitoring concepts and principles in ANA cannot rely on any a priori knowledge like the monitoring tasks, the network topology, and also the particular support for monitoring in a particular network environment and location. As identified in [2] [3], monitoring parameters may change frequently and the particular context determines how to best solve a monitoring task, e.g., how to structure a monitoring system in a distributed efficient manner; and how to bound monitoring activities with respect to granularity in time and in volume. Therefore, the monitoring concepts and principles that are adopted in an autonomic network architecture need to be open and support flexibility and adaptivity.

In this paper, we present the results of our requirements analysis for monitoring in autonomic networking. One central requirement is the need for a framework to dynamically configure a distributed monitoring system in which an arbitrary number of monitoring components gather data, and exchange, combine and aggregate it to provide input to decision modules. The decision modules in turn reason about the monitoring results and determine how to control the network respectively a part of the network. We call this complex process of gathering data, exchanging and processing it *information* and *knowledge management* and present its concepts that are supported in ANA. Information and knowledge management enables an autonomic network to freely configure distributed monitoring systems and we demonstrate this by discussing how information and knowledge management can

BIONETICS 2007, December 10-13, 2007, Budapest, Hungary.

be used to establish a dynamic and adaptable monitoring solution for a P2P overlay for video streaming.

Thus, the remainder of this paper is structured as follows: in Section 2, we discuss the results of the requirements analysis. In Section 3, we present an overview of the ANA concepts and architecture. We describe in Section 4 the information and knowledge management concepts. How monitoring is integrated in the ANA architecture and a P2P video streaming example are described in Section 5 followed by the concluding Section 6.

2. REQUIREMENTS ANALYSIS

In this section, we study the fundamental challenges for monitoring that are introduced when supporting monitoring as first class citizen in an autonomic network architecture [9], followed by an analysis of more general problems and requirements for monitoring. We conclude this section by introducing those concepts that need to be supported in the ANA architecture in order to address the previously identified problems.

2.1 Fundamental Challenges for Monitoring

Existing monitoring solutions rely on *a priori* knowledge during their design and implementation phase. For example, the designer knows which monitoring task needs to be performed, which data types need to be supported, which interfaces and services are available, the generic network topology, the approximate amount of data that has to be handled, etc. Such *a priori* knowledge is not available for monitoring principles that are to be integrated as first class citizens in an autonomic network architecture. The invocation time, the execution lifetime, the dynamics and the monitoring requirements of all possible automated tasks in an autonomic network are unknown at the architectural level and often only available at run-time. Depending on the particular autonomic tasks, the monitoring needs vary in terms of the monitoring points, the target interfaces for monitoring and the monitoring behavior(s), services or tasks to be in place or to be triggered on those points, the invocation time of the monitoring behavior and its execution lifetime. Some automated tasks such as an automated troubleshooting or fault-diagnosis task may be triggered by events such as failures detected at some point and time in the network. Such an automated task may therefore require that some monitoring services/ behaviors be in place/active at some point(s) and interface(s) in the network or that if not active, be activated. All such varying needs and dynamics of diverse automated tasks impose some design and operational requirements on monitoring facilities (components and platforms) of a self-managing network, like the need to explore the interfaces and services that are available and which data types are supported through information and knowledge management, and dynamic and programmable monitoring to be able to change the configuration and behavior of monitoring services (see Section 4.3).

2.2 Generic Design Guidelines

The diverse monitoring requirements imposed on monitoring facilities of a self-managing network by automated tasks, require us to find design and operational principles of monitoring components/tasks that satisfy identified requirements. The sub-sections below provide an insight on such design and operational principles of monitoring components.

2.2.1 Cope with Change

While monitoring requirements vary from one automated task to another one, these requirements may also change in time within a given automated task.

For example, a monitoring system wants to monitor network traffic for security analysis. Depending on the threat level, more detailed information (or less detailed) is captured and analyzed. That means, if no danger is reported, only traffic volume data is captured, as soon as there is an indication of a threat or attack, the monitoring system will automatically capture more traffic statistics and may start capturing data packets or flow information.

Similarly, the monitoring system may be built on sampled flow information. The system will sample traffic data at a given rate and only on demand, the sampling rate is decreased (e.g., from 1 out of 1000 to 1 out of 50) such that more fine-grained network data is available.

2.2.2 Distributed Monitoring

Many automated tasks would require or at least benefit from several viewpoints provided by the monitoring services. Having a number of perspectives provided by monitoring in multiple locations enables wider situation awareness. To this end, we would need distributed monitoring where participant efforts are coordinated and orchestrated. Such a cooperative system makes it possible to correlate events of interest between locations, for instance. The P2P video streaming scenario in Section 5.3 gives an insight on the need for distributed monitoring services.

2.2.3 Intelligent Use of Resources

As in any context, economic usage, resources and resource sharing, opportunistic resource usage and allocation are important also in monitoring context. Certain monitoring tasks can produce a lot of data and other consume significant amount of processing power.

In order to mitigate the problem of resource demands (e.g. storage capacity for monitoring data and processing power) on a monitoring component, e.g., a monitoring probe/ sensor / system, *it is wise to have the necessary monitoring task(s), and only those tasks, invoked only when monitoring is required*, otherwise resources on the node may be used opportunistically for something else because, its not only monitoring that requires resources. This and the need to support the notion of monitoring requests, behavior specifications from automated tasks (local and remote), and the execution of requested monitoring behaviors, entails that certain design principles must be followed when designing monitoring tasks of a monitoring component.

2.2.4 Avoid Active Measurements

In the current Internet, monitoring tools apply active and passive measurement techniques. Passive monitoring captures system parameters, e.g., traffic volume and flow counts. Active monitoring implies the interaction with the system, e.g., traffic, such as probing packet pairs, is injected into the network and captured at a destination node for measurement purposes.

With the current Internet architecture, many metrics can only be measured with the help of active measurements because elements of the Internet architecture do not expose enough (hardly any) information about themselves and their current state to other elements.

Active measurements are not preferable for two reasons: scalability and accuracy. Topology discovery is a good example of why active measurements do not scale well: Performing exhaustive active measurements with the well-known tool *traceroute* in order to discover the Internet's topology on a global scale is not doable. However, if we were able to passively collect, aggregate, and distribute routing table entries, this would enable the construction of accurate topology maps.

As for accuracy, bandwidth measurements, primarily done with active measurement tools that inject probing packets into the network [12], provide estimates of end-to-end capacity and available bandwidth while we could accurately obtain such information if the elements on the path (e.g. routers and switches) would passively measure and then expose these metrics.

2.2.5 Allow Retrospective Analysis

Analysis of historical monitoring data is necessary in order to identify trends or "go back in time" to understand reasons for certain events [5] [6]. For example, resilient systems should be able to diagnose the faults by looking at historical data, e.g. traffic traces after a successful attack, in order to improve the system design.

2.3 Monitoring Concepts

Based on the challenges and guidelines presented in Sections 2.1 and 2.2, we discuss in this section the necessary concepts that we believe a successful monitoring architecture of an autonomic network architecture should support.

2.3.1 Dynamic, Adaptive, Programmable Monitoring

The monitoring tasks of a component should be designed in such a way that each task or a group of tasks can be triggered or invoked on-demand: A monitoring request issued by an automated task triggers the monitoring task. To this end, the monitoring component needs to support certain parameterized primitives, such as *start (task-behavior-spec, Time-To-Live), pause (task-id), resume (task-id), refresh (task-id, new Time-To-Live), modify (task-id, new parameters),* and *terminate (task-id),* where *task-ids* are created and communicated by the component. This type of *dynamic monitoring* enables establishment of monitoring tasks in unpredictable locations when the need arises. Furthermore, by supporting the above kind of primitives, intelligent use of resources can be ensured. Resources can be freed whenever monitoring is temporarily not required or no longer required. In addition, a component should check whether resources are available to satisfy the requirements expressed in the request and perform admission control on monitoring requests. It must also ensure that additional behaviors are not unnecessarily started by inspecting whether a particular monitoring behavior could be shared by a several tasks.

While the ability to dynamically establish, suspend, and tear down monitoring tasks is necessary, the monitoring framework should be designed also to *adapt* to changes within monitoring tasks. A particular monitoring task should adapt to both unexpected and expected events, and to changing resource availability. If for example, additional information is required to detect traffic anomalies, the data capturing is updated "on-the-fly". To limit the resource consumption of the monitoring service, the service must be able to self-adapt, e.g., by adapting the capturing rate or by applying data sampling techniques.

Programmable monitoring components allow for the specification of the behavior of a monitoring task. This type of functionality implicitly requires a description language for specifying monitoring behavior(s). Such a language should include the possibility to specify actions to be performed on the target by the behavior, event-descriptions and event-notification propagations to designated parties, including actions to be performed by notification recipients, etc.

Dynamic, adaptive, and programmable monitoring is one of the key concepts in the monitoring framework for autonomic networks. These three properties makes it possible in principle to establish required monitoring services, be it of any kind, at any necessary place and time. Furthermore, these tasks are automated as they adjust themselves to the specific environment. These properties are vital for the concept that we discuss next, the monitoring compartment.

2.3.2 Monitoring Realm

In a self-managing network, monitoring information from one single component is not sufficient for supervision tasks that cover a large partition or a whole network. That is why the ability to perform distributed monitoring tasks is necessary. We call a set of components that cooperate with each other for a common objective (a specific monitoring task), a *monitoring realm*.

There must be component that orchestrates and coordinates the efforts of such a monitoring realm. Note that this orchestration could be performed in a completely distributed fashion by the components themselves or by a central component. To form a monitoring realm, the coordinating components have to be aware of the monitoring capabilities of the nodes in the realm and must be able to locate the monitoring point of interest in the topology.

The monitoring realm should be dynamic and adaptive itself which means that such realms should be able to be created on-demand and they should react to changing needs. These realms should also be able to self-optimize by relocating monitoring services from a component to another and adding services to new components, for instance. Given that monitoring components support the concept of dynamic, adaptive, and programmable monitoring, it is possible to add new components to a particular monitoring realm and specify and invoke monitoring tasks on them whenever and wherever necessary.

2.3.3 Information and Knowledge Management

We have discussed the requirements for monitoring to enable an autonomic network to sense its operating environment and to monitor its state. However, achieving self-* properties requires more than just acquiring raw data about the operating environment and state. This data has to be used, i.e., transformed to information and knowledge to be applied to achieve these properties. For instance, self-configuration requires that an entity knows its configuration, that it knows the components and

resources which are available for potential addition, and that it can reason about the impact of configuration changes. Furthermore, self-healing requires that a system is able to define or to learn what the normal condition is and compare it with monitoring results in order to recognize deviations from the normal condition. The capacity to proactively circumvent issues that could cause service disruptions means that the system must be able to perform a retrospective analysis after a service disruption, i.e., to study data from the past in order to identify which sequence of events might lead to a service disruption.

In many cases a single source of data and information is not sufficient. Hence, we introduced earlier the concept of a monitoring compartment for distributed monitoring. However, simply collecting monitoring data in various locations is not sufficient. This is because of the fundamental property of autonomic networks of not having any a-priori knowledge about the environment in which a particular node/host or simply a task integrates itself. The unknown environment can be a network instance for a node/host but also the node/host itself for a task. Therefore, it is necessary to develop proper abstractions for providing and sharing the information between entities. This information can be derived from monitoring data or it is part of a description of an entity. Such a description could be a service description, but it could also be the description of the entities configuration, description of available components, description about available resources, etc. Furthermore, the architecture should provide means to persistently store data and information, because for certain tasks, like the retrospective analysis in self-healing it is necessary to study historical data, as we have discussed earlier. We discuss in detail the way we handle information and knowledge management in ANA in Section 4.2.

3. ANA ARCHITECTURE AND CONCEPTS

The *Autonomic Network Architecture* (ANA) project [1] has the goal to explore novel ways of designing and building networks beyond legacy Internet technology. The specific objective is to provide a meta-architecture that allows the inter-networking between different types of networks. As a result, ANA encompasses the concept of *Compartments* as a key abstraction that allows co-existence and inter-working of different types of network through a minimum generic interface. Furthermore, the operation of different types of compartments must be analyzed in order to identify the fundamental building blocks of the abstraction. The de-composition of compartments into the core tasks helps to understand how the necessary flexibility (functional scalability) to provide autonomicity can be achieved through compartments.

A second core building block of ANA is the *Functional Block* (FB). FBs are code instances and state that can process (send, receive, forward, etc.) information. Typically, the communicating entities of a compartment are represented in ANA through FBs. They implement the functionality that is required to interact with the compartment and communicate within the compartment. As such, the FBs can also be regarded as the processing elements or tasks hosted by an ANA node that constitute the *compartment stack*.

Communication inside a compartment is mediated via *Information Channels* (ICs). In order to connect FBs and ICs in a flexible manner, the ANA framework introduces *Information Dispatch Points* (IDPs). IDPs are bound to FBs or ICs, and hence provide a decoupled access to the FB or IC. This decoupling allows dynamic and transparent re-binding of ANA entities.

In the context of the ANA node architecture, monitoring is implemented as a generic FB, the *Monitoring Functional Block* (MFB). When other FBs require monitoring information for their decision processes, the MFB initiates and orchestrates the data collection and distribution of monitoring results. Note that in Section 5, we describe more specifically how monitoring is integrated in the ANA node architecture.

In Section 2.2, we introduced the concept of *monitoring realms*. In the ANA context, such monitoring realms are implemented as network compartment. A *Network Compartment* is a compartment that encompasses several ANA nodes and involves communication across an underlying network infrastructure. Like for any compartment, a network compartment consists of a policed set of FBs, ICs, and IDPs. In case of a monitoring compartment, the MFBs providing the compartment's monitoring capabilities are located on multiple distributed ANA nodes. Note that monitoring compartments may also span over multiple network compartments.

4. INFORMATION AND KNOWLEDGE MANAGEMENT

In this section, we describe the general information and knowledge management concepts needed for autonomic networks and how we use them in ANA.

4.1 Information Hook

An information flow includes at least two functional blocks, of which one has the role as information provider and the other as information consumer (respectively decision maker). The information provider is the entity that could provide the information about which data it can provide and how it could be used. Thus, the information provider needs to provide a self-descriptive interface. This interface would serve as a hook to which the consumer can connect and exploit how to make use of the data. Such a hook should exist in each element of the autonomic network architecture. This information hook needs to have two main properties:

(1) Unified generic interface: The hook needs to have a unified interface so that each entity in the autonomic network knows how to access it. Otherwise, information flow establishment is impossible. The interface needs to be generic and as simple as possible. In this way, the interface can be used in an extensible way: Imagine a simple interface that provides by request a more complex interface more specific to the particular information hook.

(2) Self-describing: The hook needs to be able to describe what kind of information it can provide. In this way, any entity can query the information hook of a particular entity in order to learn if that entity can provide the information that it needs. Without such a capability, the abstraction of providing information and sharing information between entities becomes pointless. The information flow architecture needs to consider what kind of syntax and semantics the descriptions should have.

An autonomic network has to provide a new architecture for data transfer. Instead of providing a fixed network stack for communication, the required functional blocks are combined into a function chain on demand. The elements which can be composed, i.e., FBs, can consist of whole protocol functionality (e.g. TCP, UDP) or may provide only micro-protocol functions, like error control or encryption. Information processing itself is done by FBs and is therefore not part of the information flow framework, even if it is using information flows. For example, a

monitoring FB that computes some aggregate metrics by combining several monitoring information flows is not an information flow itself, but it uses information flows to implement a specific kind of monitoring service.

Each entity that supports the information flow concept implements an information hook. This hook provides a description of the entity. The content, format, etc. of the description depends entirely on the entity itself, but the minimal mandatory description may be specified within the policies of a particular realm.

4.2 Information Hooks in ANA

An information hook has to be generic and self-describing. A generic hook allows in principle any ANA entity to access and request a description of any other entity having an information hook. As a consequence, the generic hook is kept minimal: it provides access to the description of the entity. This description provides information about how to interact with the entity bearing that hook, i.e. in essence a description of its interface. The element accessing the hook would then learn what information and services the entity bearing that hook can provide and how they can be accessed, because the entity description comprises a description of the semantics of the functions, i.e., what they do, and the syntax of the functions, i.e., how to properly invoke them with the correct parameters, etc.

The question which language and tools to use for entity descriptions to (a) describe the functions and (b) interpret these descriptions is not part of the generic information hook. Instead it must be determined/standardized for a particular ANA compartment. An example for possible language and tool that might be used for the syntactical aspects of functions is the Interface Definition Language (IDL) [11] and a kind of stub compiler to generate the two procedures for external representation handling. To describe the semantics of functions and their parameters, a combination of the Resource Description Format (RDF) [10] and XML [8] as it is proposed in [7] could be used.

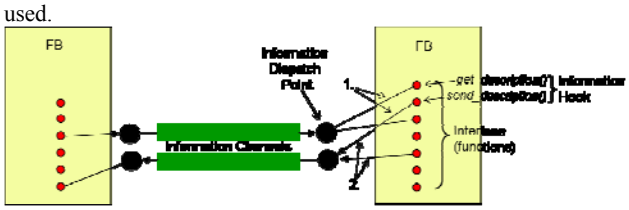

Figure 1: Accessing interfaces of Functional Blocks

Each ANA entity has a unique identifier within a compartment. This identifier can be used to access the information hook in order to get a description of that entity, i.e., the *get_description()* function corresponding to a specific identifier is called. Such a procedure is often the starting point of communication. For example, a FB would like to get certain information from another FB whose identifier it has resolved. It may be that the requesting FB does not know how to access the information or does not even know whether such information is provided by the other FB. In such a case, the first step for the requesting FB is to invoke the *get_description()* function corresponding to the resolved identifier.

The information hook is composed of two functions: *get_description()* and *send_description()*. *Get_description()* returns the entity description by invoking *send_description()*. After receiving the description, the requesting element knows how to interact with the other element. Thus, afterwards the interaction happens through the element specific interface, as illustrated in Figure 1. The entity description type is mandated by the compartment policies, which ensures that the receiving element is able to interpret the description.

4.3 Information Hooks for Monitoring

The monitoring framework is tightly coupled with the information and knowledge management framework. The main goal of information flows is to provide a principal concept for autonomic networks to enable entities to exchange information in order to use it for decision making. One of the key concepts of the information flow architecture is the information hook that is implemented by each ANA FB and IC. This hook provides a description of the entity bearing that hook, e.g., a FB. By interpreting the description another entity learns how to interact with that FB and what information it can provide.

Usage of information hooks and monitoring are intertwined: On one hand, MFBs use the information hooks of the other FBs residing in the node in order to collect information about potential monitored data exposed by the FBs. In this way, the MFB also build up (parts) of its own description that can be obtained via its information hook. On the other hand, the description of a composed FB (or an IC which is in fact a composed FB) can be maintained using monitoring services in the following way. If a composed FB chooses not to expose its internal structure through the information hook, it must then collect relevant information of the internal FBs and present it in a consistent way via its information hook. MFB within the composed FB can be requested to coordinate this collection procedure as monitoring tasks.

Figure 2 illustrates the usage of information hooks in the monitoring architecture. It should be possible to query the MFB through an information hook the services it currently provides or is capable of providing. In addition, the FB that implements monitoring storage should provide a hook as well through which it is possible to learn its properties, i.e. what type of storage systems can it support, which data models it supports, etc.

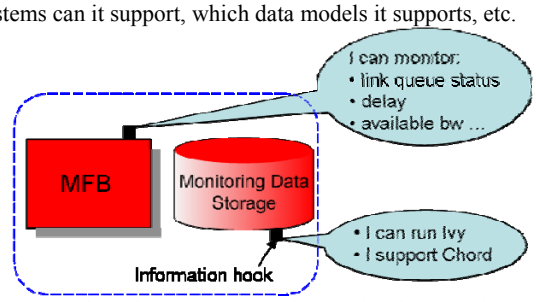

Figure 2: Information hook in monitoring architecture

5. MONITORING IN ANA

In this section, we explain the monitoring concepts that we use in ANA.

5.1 Monitoring in the ANA Node Architecture

Figure 3 shows the ANA node architecture from the monitoring perspective. The MFB triggers the capturing of information. This FB is also used to request information. Monitoring storage is used to store the collected data. This block may be located in the node itself or maybe attached to it in an external device or virtual device (P2P approach). The "conceptual" membership database within a node compartment stores information about locally

available functionality and services, including access to other compartments and sibling applications. The MFB checks from that "database" what FBs are present in the node and establish necessary IC to them (via IDPs) through the resolution process. Then MFB would then query the information hook of those FBs to find out what information it can obtain and how, and then collect the necessary information. This process can be used to establish also the description of available monitoring services that a particular MFB can provide, i.e., (part of) the service description obtainable from the MFB's information hook.

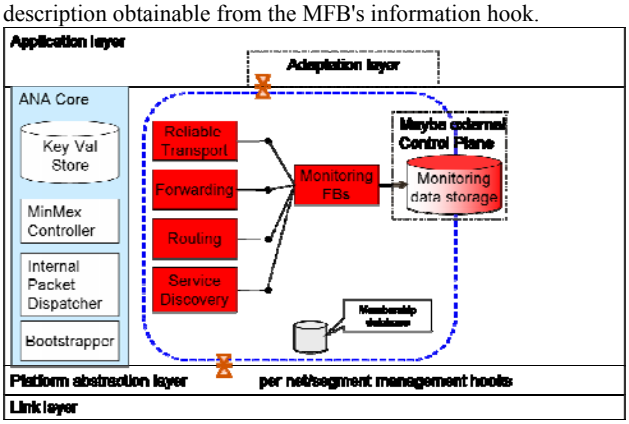

Figure 3: ANA node architecture for monitoring

Each FB provides a service description, i.e., it announces the parameters it provides to the MFB. Via the information hooks, the MFB is then capable of reading the to-be-monitored parameters, processing the collected data, and sharing the results with other MFBs or FBs. Also, the MFB may request a specific FB to expose information.

From a design point of view, specific monitoring tasks such as tasks for traffic monitoring/capturing/filtering/ analyzing; functions for storing, managing and disseminating monitoring data; functions for detecting events occurring internally and externally to the system etc. may need to be placed into separate inter-working functional blocks dedicated to providing specific monitoring services.

5.2 Monitoring Compartment in ANA

The monitoring compartment in ANA is a set of MFBs that cooperate and join in their monitoring efforts. In the next section we show one possible form of such a monitoring compartment in ANA. In that example the orchestration is decentralized in the sense that there is no central component that manages the MFBs. Instead, each MFB may choose to trigger monitoring services to be executed by other MFBs.

5.3 Example Scenario

In this section, we describe an example application scenario for ANA and show how the monitoring part of the architecture can be used in this scenario. Our example application is P2P-based Video-on-Demand streaming.

The term content distribution network (CDN) covers many different ways of moving data between computers. There are three main categories. The first is downloading based, where content is accessed only after having been completely downloaded. The second is broadcast based, where all receivers receive the same data more or less simultaneously. The third is CoD (Content-on-Demand) based streaming, where data is accessed as it is being received. Content is typically located in one of two ways; the identifier based approach used on the WWW, and the message digest based file identification used for file sharing in many P2P networks.

Content distribution using P2P technologies is a very promising application domain. We use here P2P-based Video-on-Demand (VoD) streaming as an example. VoD streaming is a service with high resource requirements, because video files are usually large and need to be delivered in a timely manner. Thus, high bit rates are needed for the duration of the video, which could be up to several hours. In order to efficiently deploy a large scale VoD system, end user resources can be utilized in a P2P fashion. However, such a solution requires prompt and detailed information about the network.

Content is organized in files, which are logically divided into blocks, and these blocks can be retrieved from any peer which already has a copy. Client applications are responsible for requesting blocks in a timely fashion, and presenting the video to the user.

We view the set of FBs in various nodes as a compartment that forms the P2P overlay for VoD services. These FBs are such that wish to participate in distributing the particular video content. In this example scenario, communication, i.e. the distribution of content, within the overlay compartment is optimized according to specific criteria. The overlay should be optimized so that overall the bandwidth is optimally utilized among the nodes. As ANA architecture is autonomic, the overlay is able to self-configure and self-optimize. Ideally, the application only needs to specify the content and then issue requests for blocks to a handle (in an anycast style) that it gets from the underlying ANA, which takes care of all the rest. For instance, the application does not necessarily need to know which node to contact.

Figure 4 shows the way monitoring services could be used in this scenario. The MFBs of individual nodes that monitor the available bandwidth to surrounding nodes form a monitoring compartment. We assume that each of the members of that compartment knows the neighboring members, i.e., adjacent nodes which belong to that monitoring compartment and to which the data path does not pass through another member node. We say that these neighboring nodes form the vicinity of the MFB.

Monitoring services are first invoked within the MFB of the downloading node, node 1. This triggering could be done, e.g. by the overlay FB or another FB that is concerned about the self-optimization in that node. The task is to monitor available bandwidth on the paths from that node to nodes 2-6 that contain blocks that the application would like to download. The MFB starts to monitor the available bandwidth on the link to the next node in the vicinity and establishes an IC to the MFB of that next node in order to receive similar monitoring information from that node. Similar monitoring tasks are thus dynamically started in that MFB and new ICs established to the MFBs of the next nodes in the vicinity towards the final destinations (nodes 2-6). This behavior continues until the entire paths to the nodes 2-6 are covered. After that, monitoring information flows through the channels all the way to node 1. The information is aggregated along the way in the intermediate nodes, i.e., nodes always report minimum available link bandwidth along the path to a given node. As can be seen in top-right part of the figure, sometimes the adjacent nodes do not belong to the monitoring compartment. In

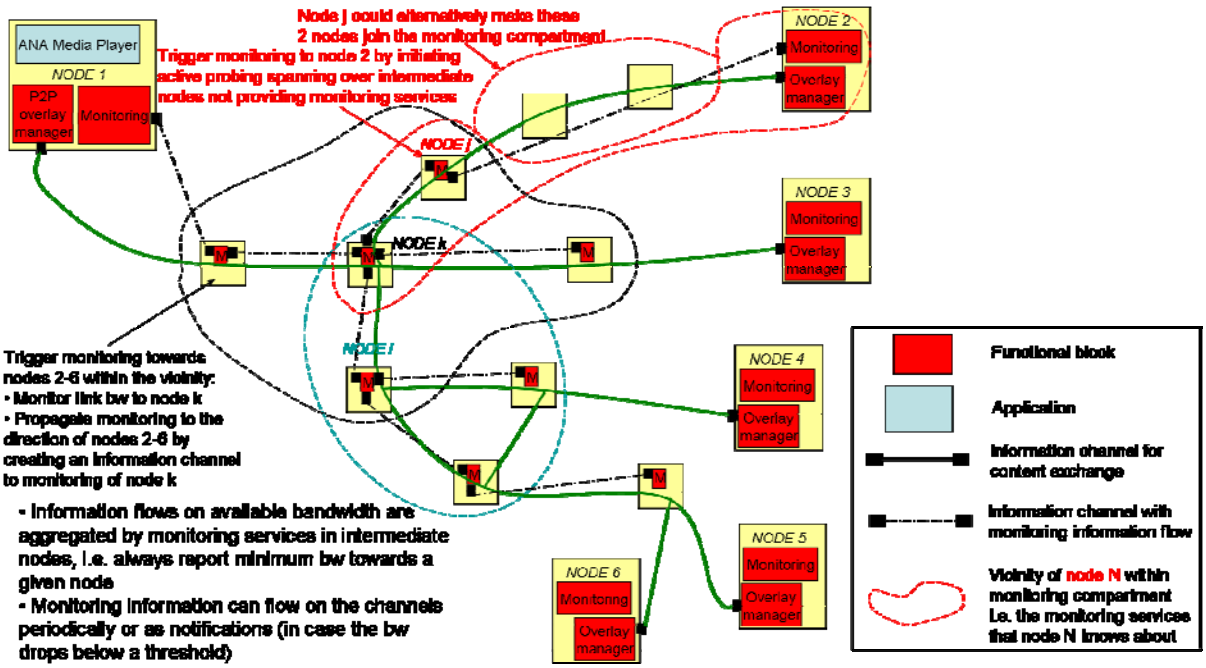

Figure 4: Monitoring in P2P VoD streaming scenario

that case, it is naturally not sufficient that node j in the figure only monitors link bandwidth to the next node. In such a case, given that MFBs within nodes support the concept of dynamic and programmable monitoring, the MFB in node j could invite the MFB of the next node join the monitoring compartment by specifying and invoking the bandwidth monitoring task for that MFB. The newly joined MFB would then in turn make the MFB in the other intermediate node join as well. As an alternate approach, if the intermediate nodes do not support monitoring of their link bandwidth, node j could perform active probing towards node 2 in order to determine the available bandwidth to the whole remainder of the path.

In this scenario, monitoring of available bandwidth along a set of paths is required. In the Internet today, such monitoring can be only performed via active end-to-end probing. In ANA, the ideal case from the monitoring information timeliness and accuracy point of view is to be able to receive such information from each intermediate hop along a given path. However, it may be that not all nodes along the path provide such monitoring services initially. In such a case the MFBs of such nodes could be integrated into the monitoring compartment by making a request and describing to them how to perform the needed monitoring tasks.

The scenario described requires having dynamic and adaptive monitoring services that can be started and stopped on demand. Only certain nodes need to perform monitoring of available bandwidth while other nodes save their resources. It is especially important when the available bandwidth needs to be estimated through active probing that can potentially create a significant extra load to the network.

The example pointed out also another important aspect: the usefulness of programmability of monitoring services. In this way, nodes could request and instruct other nodes to join a particular monitoring compartment. It should also be possible for a node to learn which monitoring capabilities another node may have.

6. CONCLUSIONS

In the ANA Project, we design a new autonomic network architecture from scratch with monitoring as a first class citizen in the architecture. The fact that monitoring is considered as a fundamental element in the architecture introduces a new set of challenges for monitoring at this abstraction level. The main reason for this is that no a priori knowledge is available for monitoring concepts in the architecture. Therefore, monitoring needs to be able to dynamically explore the environment in which it should work and monitoring elements should be dynamically composable in such a way that distributed monitoring realms can be created on the fly. In order to achieve this, the ANA architecture provides the concept of information hooks in the information and knowledge management framework. Furthermore, monitoring needs to flexible and programmable to dynamically address the given tasks.

It is the contribution of this paper to provide an analysis of these new requirements and a discussion how they can be supported in the ANA architecture with different concepts and how these concepts inter-work. We use the example of a monitoring compartment for P2P video on demand streaming to explain the idea behind our proposal for monitoring in ANA and as a kind of a first conceptual validation of the presented monitoring approach.

So far, the presented results are at the conceptual and architectural level. Therefore, many research challenges have to be solved to implement and validate these concepts. For example, the dynamics and varying monitoring needs of envisaged automated tasks meant for self-managing networks impose some design and operational requirements on the monitoring facilities (component, functions, and platforms) of a self-managing network. Capturing and

specifying those requirements along with finding and designing suitable monitoring paradigms/frameworks, is not a trivial issue.

7. ACKNOWLEDGMENTS

This research has been performed in the ANA project No. FP6-IST-27489, funded by the EU 6th Framework Programme, Situated and Autonomic Communications (SAC).

8. REFERENCES

[1] ANA (Autonomic Network Architecture) Project: http://www.ana-project.org.

[2] R. Boutaba, J. Xiao, *Self-Managing Networks*, in: Cognitive Networks, J. Wiley & Sons, 2007

[3] M. Crovella, B. Krishnamurthy, *Internet Measurement: Infrastructure, Traffic and Applications*, J. Wiley & Sons, Inc., 2006

[4] B. Jacob, R. Lanyon-Hogg, D.K. Nadgir, A.F. Yassin, *A Practical Guide to the IBM, Autonomic Computing Toolkit* (Book Title), IBM Corporation, April 2004

[5] F. Reiss, J.M. Hellerstein*, Declarative Network Monitoring with an Underprovisioned Query Processor*, ICDE '06: Proc. of the 22nd Int. IEEE Conf. on Data Engineering (ICDE'06), Washington (USA), 2006

[6] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, R. Sommer, *Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic*. 2005. Proceedings of 5th ACM SIGCOMM conference on Internet Measurement (IMC '05), Berkeley (USA), Oct. 2005, ACM Press

[7] J. van der Ham, F. Dijkstra, F. Travostino, H. Andree, C. de Laat, *Using RDF to Describe Networks,* Future Generation Computer Systems, Feature topic iGrid 2005,2006

[8] D. Beckett,B. McBride,*RDF/XML Syntax Specification*, February 2004,http://www.w3.org/TR/rdf-syntax-grammar/

[9] R. Mortier, E. Kiciman, *Autonomic Network Management: Some Pragmatic Considerations*, Proc. of ACM SIGCOMM workshop on Internet Management, Piza (Italy), 2006

[10] *Resource Description Framework (RDF)*, URL http://www.w3.org/RDF/

[11] *Specification of OMG IDL*. http://www.omg.org/cgi-bin/doc?formal/02-06-39

[12] G. Varghese, C. Estan, *The Measurement Manifesto*, ACM SIGCOMM Computer Communications Review, Vol. 34, No. 1, Jan. 2004, pp. 9-14