DEMAND DRIVEN NETWORK MONITORING INFRASTRUCTURE: A PROTOTYPE*

Augusto Ciuffoletti
INFN/CNAF
Via B. Pichat 6a
Bologna - Italy
augusto@di.unipi.it

Yari Marchetti
Dept. of Computer Science of the University of Pisa
Largo Pontecorvo
Pisa - Italy
marchetti@cli.di.unipi.it

Antonis Papadogiannakis, Michalis Polychronakis
FORTH
Heraklion (Crete) - Greece
[mikepo,papadog]@ics.forth.gr

Abstract The capability of dynamically monitoring the perfomance of the communication infrastructure is one of the emerging requirements for a Grid. We claim that such a capability is in fact orthogonal to the more popular collection of data for scheduling and diagnosis, which needs large storage and indexing capabilities, but may disregard real-time performance issues. We discuss such claim analyzing the gLite NPM architecture, and we describe a novel network monitoring infrastructure specifically designed for *demand driven* monitoring, named *gd2*, that can be potentially integrated in the gLite framework. We describe a Java implementation of *gd2* on a virtual testbed.

Keywords: Network Monitoring, gLite, Network Measurement, XML Schema Description, Java, User Mode Linux.

*This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1. Introduction

End-to-end network monitoring is a key issue in the management of production Grids: with reference to a frequent situation in a replica management scenario, it would be useful to monitor network performance both before and during access to replicated data, in order to dynamically select a replica that offers an acceptable accessibility. However, end-to-end network monitoring introduces distinctive problems.

For one, its complexity potentially scales up with the square of the size of the system, while other resource monitoring activities (for instance processing power) scale linearly. To ensure its scalability, end-to-end network monitoring must be selective in its targets: only a significantly small fraction of end-to-end paths can be monitored at each time. As a consequence, whatever the criteria to select which path is to be monitored, we need some sort of distributed infrastructure in order to activate and deactivate network monitoring selectively.

Another problem comes from the accessibility of the resource. When we monitor other kinds of resources, e.g. processing capabilities, the sensor has direct access to the resource. In the case of network monitoring, we often observe that the monitoring tool requires some sort of cooperation from the resource itself: for instance, even the trivial ICMP ping requires that packets are freely propagated, which is not always true. As a general rule, an end-to-end network element must be treated as an opaque box, showing a performance which is traffic specific. One way to overcome this problem is to use passive measurement techniques, instead of active, thus analyzing existing traffic: such solution is also the foundation of the `IPFIX` [11]protocol, currently discussed within the IETF. In our approach, traffic analysis is delegated to specialized units, located where it is possible to intercept traffic between end-points. The result of such activity should be collected and published only after checking the credentials of the requester: these data should be regarded as subject to security restrictions.

Summarizing, we establish two cornerstones for an end-to-end network monitoring architecture capable of managing the scalability challenge offered by a Grid environment: i) *demand driven*, in the sense that its activity is not set by default, or with static configurations, but controlled by external agents, and ii) *passive monitoring oriented*, in the sense that only existing traffic is analyzed in order to obtain the requested measurements.

The next section goes into the details of a novel architecture which is based on the above foundations: it is the result of a joint activity of INFN-CNAF (Italy) and FORTH (Greece), in the frame of the European CoreGRID project.

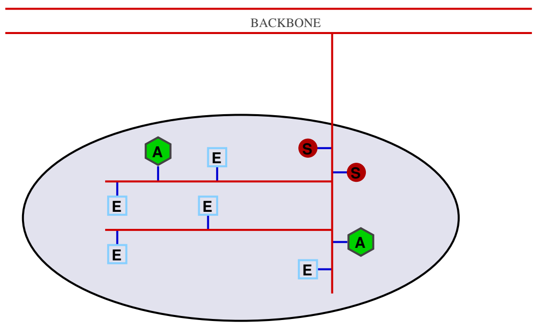

Figure 1. Deployment of *gd2* components in a Domain: E units represent generic monitoring endpoints, A labeled units represent Network Monitoring Agents, S units represent Network Monitoring Sensors

2. The components of a demand driven network monitoring architecture

Our architecture partitions Grid end-points into *Domains* (see figure 1). A Network Monitoring Agent (Agent, in the rest of this paper) takes the responsibility of managing a number of Network Monitoring Sensors (Sensors, in the rest of the paper), and of agents enabled to submit network monitoring requests, the Network Monitoring Clients (Clients, in the rest of the paper) that compose the Domain. There are good reasons to introduce a partitioning, roughly the same that motivate its introduction in many aspects of networking: *reducing complexity* – one Agent concentrates the interface to the entities inside a domain; *security containment* – security issues can be managed using local credentials inside a domain; *limiting global state access* – only Agents have access to the global state, thus simplifying its management and ensuring security.

2.1 The Network Monitoring Agent

The services offered by an Agent can be divided into two quite separate interfaces: one towards the other Agents (back end), and another towards local sensors and clients (front end). In figure 2 the triangular shapes indicate front end interfaces. We examine the two faces, and next detail the internal structure of the agent.

The *back end* interface is in charge of maintaining the membership of the Agents in the system. Such membership is the repository of two relevant data: 1) the credentials of the Agents, needed to enforce security in communications among the agents, and 2) the components of each domain.

As for the first point, we envision a public/private key scheme as adequate for our purpose: we consider that security primarily avoids the intrusion of

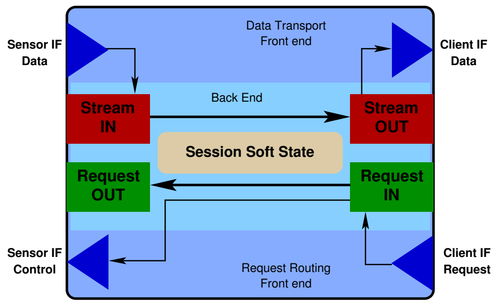

Figure 2. Internal architecture of a Network Monitoring Agent. The Back End interfaces are located in the innermost stripe

malicious entities disguised as Agents. Whenever the results of the monitoring activity are considered confidential, Clients and Sensors will be in charge of encrypting sensitive data according to agreed methods. In order to control access to the membership, we assume the existence of an external entity in charge of key creation and assignment. This Authority, upon admission of a new agent, releases a certificate, which entrusts the use of the public key as authorized by the Certification Authority. Each Agent has access to a repository containing the certified public keys, and each communication within the membership is accompanied by the signature of the sender (not encrypted, in principle), which can be checked using the public key.

The reader understands that the implementation of the certificate repository is a component whose implementation affects the scalability of the whole architecture: we deal with such component in page 7.

The back end, upon receiving a request, submits its content to the *front end* in order to assess its capability to take it up: it is possibile that the information in the domain directory was insufficient to determine the appropriate Agent for the task. Therefore we consider that the front end may fail to fulfill a request: in such case, the back end will trap the failure, and resubmit the request to another Agent. Such re-routing will be controlled either by alternate Agents whose identity is indicated in the global directory (as a general rule several agents are responsible for the monitoring of a given network element), or by information available to the front end, obtained from inside the domain. We indicate the capability of re-routing requests as a "Proxy" functionality.

An Agent offers another back end service for the transport of Network Monitoring data to the Client that requested it: such transport service consists of a stream from the Sensor to the Client, and is routed transparently through the reverse of the path used to deliver the request. The content of the stream may be encrypted, in case the network monitoring results are considered as confidential, but the client(s) must own the key to decrypt the data: here we assume that such keys are negotiated when the network monitoring task is accepted for execution.

The *front end* of the Agent is in charge of interacting with Clients and Sensors inside the Domain: the Agent accepts requests for Network Monitoring from the Clients, and drives the Sensors in order to perform the requested network monitoring activity.

The network monitoring activity is organized into *Network Monitoring Sessions* (or Sessions, in the rest of this paper). A session describes the endpoints of the Network Monitoring activity, as well as the kind of activity required. The request must determine, either implicitly or explicitly, the features of the stream that will be produced to return observations to the Client. In [5]we give an XML Schema Definition for such data structure, the Session Description.

The Clients submit their requests to the Agent as Session Descriptions. The Agent is in charge of checking whether the request comes from an authorized client: this functionality is supported by a trust supported internally to the domain, independent from that used within the membership of the agents. This allows the possibility of merging domains with distinct security policies and support. The request is then passed to the back end.

The front end, upon receiving a request from the back end, analyzes its content to assess its ability to configure a Sensor that performs the task: to this purpose, the Agent must have access to a directory, internal to the domain, containing the descriptions of the sensors.

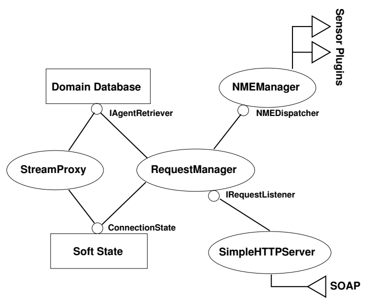

Figure 3. Modular view of a Network Monitoring Agent

The abstract functionalities described above have been implemented as a multi-threaded daemon (see figure 3). The *StreamProxy* thread is in charge of passing through the streams of data from sensors. It is composed of four threads that implement a pipe composed of four tasks: to receive the packet, to verify its signature, to generate the new signature, and to send the packet to the next hop. These threads utilize the AgentRetriever API provided by the database in order to have access to the Domain Directory, and the APIs used to access the shared Soft State through the interface *ConnectionState*. It implements the "Stream IN, Stream OUT" boxes in Figure 2.

The *RequestManager* is another thread in charge of routing network monitoring requests, and implements the "Request IN, Request OUT" boxes in Figure 2. As in the case of the *StreamProxy*, the ConnectionState and the AgentRetriever interfaces grant access to the Soft State and to the Domain Directory.

Requests are acquired by a *SimpleHTTPServer* thread that offers a SOAP interface to the Clients, and they are delivered to the *RequestManager* through its *IRequestInterface* interface.

The *RequestManager* controls the Sensors through a set of plugins, each of them specifically designed in order to drive a certain kind of sensor. Sensor plugins offer an interface with a single **dispatchRequest** method.

2.2 A passive Sensor and its plugin

Passive monitoring sensors are usually located at selected vantage points in the network that offer a broad view of the traffic of a domain, such as the access link that connects a LAN with another, or an Autonomous System to the Internet.

To support passive network measurements using the *gd2* architecture, we have developed a plugin within the Network Monitoring Agent which controls the passive monitoring sensors. The passive monitoring plugin first receives the configuration parameters for the passive network measurements from the client's request: available measurements are round-trip time [7], delay and jitter, packet loss rate [8], available bandwidth, and per-application bandwidth usage [1], based on the the Distributed Monitoring Application Programming Interface (DiMAPI) [13]developed at FORTH. These parameters are derived from the measurement specific part of the session description document, while the **MAPIOptions** element provides the relevant parameters for the passive monitoring tools.

When the starting time of a measurement comes, the passive monitoring plugin invokes the execution of a DiMAPI program that coordinates the remote monitoring sensors for the task. Dynamic configuration of the sensor includes the specification of packet filters, the definition of the processing operations that should be performed for each network packet, and the kind of results that should be produced, using the suitable DiMAPI functions [13]. The measurement results from each sensor are periodically sent to the DiMAPI program for aggregation and then returned to the plugin in the NMA. Finally, the plugin parses the results and sends them to the consumer through an encrypted connection.

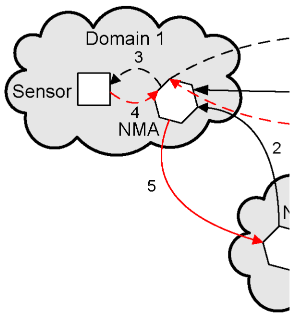

Figure 4. Invoking a passive measurement for packet loss ratio in *gd2*: a plugin inside NMA initiates a DiMAPI program which gathers results from two remote sensors

Figure 4 presents an example of a passive measurement session for the packet loss ratio between two different domains: we emphasize that such a measurement requires sophisticated techniques in order to be performed according with a passive approach to network monitoring. Initially a client submits a request to the local NMA (**1**), and the request is forwarded to a corresponding NMA (**2**) that should perform the measurement. Then, the passive monitoring plugin parses the request and initiates the execution of a DiMAPI program that computes the packet loss ratio between the two domains using data from two corresponding monitoring sensors. The program first configures the two sensors (**3**) and then the results are streamed from the sensors to the DiMAPI program (**4**), which computes the packet loss ratio and reports it to the passive monitoring plugin. Finally, the results are streamed to the local NMA (**5**) and to the client (**6**).

We have currently implemented the passive monitoring plugin to support `appmon` [1], a DiMAPI based tool that reports the accurate bandwidth usage for individual protocols and applications, and `packet loss` [8]measurement tools.

2.3 The Domain Database

In order to understand the role of the Domain Database, we illustrate the decisions that the Agents take on the basis of its content.

The first decision step on this way is performed by the *Agent* once it receives a request from a *Client*: it consists in determining the Source and Destination domain of the network element under test. Such information is obtained by way of a query to the Domain Directory. The request is then forwarded to an Agent in such domains: the identity of such agents and their address is again obtained from the Domain Directory. We exclude that a monitoring activity is performed by an intermediate domain, since this would require the availability of routing information for the overall system.

Each agent on the way of the Request will in turn check the signature associated to the request, and replace it with its own. A query to the Domain Directory returns the public key needed to check the signature.

Each agent in turn will check the availability of the network monitoring functionality within the domain. This step is performed without further access to the Domain Directory, but browsing the capabilities available within the domain. We emphasize that our perspective helps to simplify this task: the adoption of passive tools helps us in limiting the number of potential producers (the *sensors*) in our architecture. Therefore the search for a producer is restricted within a limited number of sensors: such search can be either based on a local directory, or simply carried out broadcasting the request template to the local sensors.

The above discussion explains why the Domain Directory is to be considered a critical component in the structure: it is a potential single point of failure, and a performance bottleneck. A centralized implementation is therefore incompatible with the scalability of our architecture. However, the information stored in the Domain Directory is seldom updated, and this opens the way to strongly distributed solutions.

There are several options, that depend on the scale of the Grid of concern. One is to apply to a LDAP or DNS based implementation. Such well known tools are ready solutions for the maintenance of a distributed, that allow data replication in order to improve performance and fault tolerance. Such solution is probably adequate to most current scenarios.

Going beyond such scale, we indicate the implementation of a fully delocalized solutions: in essence, all Agents cache a part of the database, and updates are propagated according with a peer to peer protocol. Such approach may significantly improve scalability, while reducing the footprint for the maintenance of the Domain Directory. A theoretical investigation about the topic are reported in [3], while experimental results are in [4].

3. Related works

The NPM architecture [9]is one of the most promising proposals for network monitoring, and is presently embedded in the gLite infrastructure, designed and implemented in the framework of the European Project EGEE. NPM is designed to provide two types of information: measurement data, in the form of data records conforming to OGF standards, and metadata, indicating what kind of data are available for a given network element. Such information is delivered to clients, whose role is to diagnose network performance problems.

The client submits its request to intermediate entities, the *mediators*, through a web service interface. Such request may either exhaustively describe a measurement series, or ask for the retrieval of metadata about the measurements available for a given network element. In the former case, the requested data will be delivered to the client, while in the latter the client will be presented with a list of available measurements to choose from. In either case the *mediator* will use services offered by another kind of component, the *discoverer*, which is in charge to either locate the requested data, or to produce the listing of available sources. The source of the monitoring data is called *framework*, and it provides access to the tools that extract network monitoring data. A detailed description of the above services is in [10].

NPM strongly focuses on the accessibility of historical data: this makes a relevant difference compared to our perspective. In fact, since we mainly address data collected on demand, we necessarily exclude, for performance reasons, a web service oriented architecture for the retrieval of measurements. Instead we introduce a long lived communication entity, a stream. For the same reason we need not to address a large database of collected data: data are delivered to interested users, without being stored anywhere (unless a Client wants to do so). This avoids the need of *indexing* data, one of the functionalities associated to the *discoverer*. In our architecture the discovery activity focusses on a far less complex task: determining where to fire the measurement session.

We conclude our discussion remarking that a direct comparison is in fact inappropriate: the two frameworks, NPM and *gd2* address two distinct problems, and each of them is a poor solution when applied to the problem for which it has not been explicitely designed. A *gd2* Agent is designed to diagnose network problems once they have been detected, but has no detection tools: here we present a framework that helps detecting a network problem, and possibly overcome its presence without diagnosing its source. The NPM has an extremely heavy footprint when used to receive real time updates of the performance of a network element, which is needed to detect problems; our framework has no way to explore the past of an observation, tracking up to its cause.

Since their application domains are different, one may guess that they may live side-to-side in the same infrastructure. We believe that this is possible, at least in perspective. For instance, a *client* in our framework might be embedded in a NPM framework: its *request* might consists of a long-lived, continuous monitoring activity, and the flow of observations might be recorded for future use of NPM diagnostic tools. However, such a publication modality cannot replace the stream introduced in *gd2* when the client is an entity in charge of monitoring the real time performance of an end-to-end path.

The approach presented in this paper is also complementary with the **IPFIX** project [11]: the purpose of the IETF initiative is to design a protocol for flow metering data exchange between IPFIX Devices (corresponding to sensors in out framework) and IPFIX Collectors (Clients in our framework). Such a protocol roughly corresponds to the payload of the Sensor to Client stream, and can be used whenever netwrok utiliziation has the characteristics of a flow. We plan to converge to an IPFIX compliant architecture, and an IPFIX interface for MAPI is under work.

A monitoring infrastructure which inspired our work is **CoMo** [6], a passive monitoring infrastructure ideated by Intel. A branch of such project covers the placement of passive sensors [2], a relevant issue that is not considered in our paper. The CoMo research stream explores many relevant aspects of network monitoring, but fails to give an exhaustive description of the conversation between the Sensor and the Client, which is the main purpose fo our work.

4. Prototype layout and operation

The purpose of our prototype was to assess the feasibility of the whole *gd2* architecture, focussing on the communication infrastructure: therefore we tried to concentrate our efforts in order to produce a real scale support for a community of Agents, leaving behind other aspects of our architecture.

We implemented a fully functional request delivery infrastructure, as well as the streaming in charge of returning the data to the requester. We took into account the security issues mentioned above, using signed communications among the Agents, taking care of the organization of the content of the database.

One of the aspects that are considered to a limited extent is the implementation of the database: we have implemented a solution based on an LDAP directory, whose scalability is similar to other solutions based on this technology. Although we are actively working to design a solution with better scalability, we have evidence that the pragmatic solution given by LDAP is satisfactory at the current scale of real Grids.

In order to debug and demonstrate the functionality of the prototype, we have implemenmented a virtual testbed using the NETKIT toolset [12], based on the User Mode Linux technology, which allows to virtualize several distinct hosts using a single computer. The virtual hosts appear as complete PCs, with independent storage, computing and networking facilities. They can be interconnected, using ordinary interconnection tools, into a virtual network. Aside from the limited amount of resources needed to synthesize the testbed, the major advantage of such an approach is that the experiments can be easely replicated on distant sites, thus allowing a collaborative development of the software without need of sharing hardware facilities, and always run under extremely controlled conditions. Demonstrations can be produced using any available Linux machine, and without installing experimental software on the real computer[1].

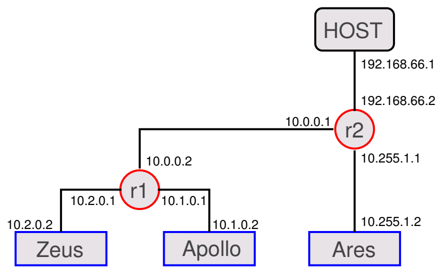

Figure 5. Development testbed

In our testbed we synthesize a network composed of three Agents and two routers (see figure 5): each of the Agents lives in a distinct domain. One of the Agents was equipped of a Client interface able to generate Network Monitoring Requests. We observed the delivery of the Request to the Sensor, with one or more hops within the Agents membership, and the flow of observations from the Sensor to the Client.

5. Conclusions

Our investigation, which attained the detail of a real scale implementation, lead to a clear view of the problems related to *on demand* network monitoring, and to the change of attitude needed with respect to the, so to say, diagnosis-oriented network monitoring. A demand driven architecture is not data-centric, in the sense that storage and indexing of measurements are not relevant, but more capability-centric, in the sense that operational network monitoring capabilities must be indexed, and protected against misuse. Therefore we need an architecture that is able to give a structure to the membership of the components that have monitoring capabilities, so to provide a capability based addressing of the monitoring resources.

We have identified such structure in a topology-bound partitioning: such structure must be sufficiently stable, in order to allow a distributed management of the directory that describes such partioning. In order to effectively abstract from the internal structure of a domain, we introduce components that manage the monitoring capabilities within a domain.

[1]The package with the virtual testbed (designed for Ubuntu Linux) is available at `http://network-monitoring-rp.di.unipi.it/`, with instruction for its installation

The primary security need is to avoid unauthorized access to network monitoring capabilities: to this purpose we need a robust authentication scheme, which is again based on information contained inside the distributed directory.

Data transfer must focus on long lived, low bandwidth data transfers: a *less than best effort* paradigm seems appropriate for their definition. This seems to match with a stream oriented protocol, that uses routing information obtained during the delivery of the network monitoring request.

In such scenario, passive monitoring is not only an option motivated by a low footprint. Passive end-to-end monitoring capabilities can be concentrated in a few locations within a domain, thus simplifying the indexing of available capabilities, instead of scattered on each possible endpoint: which comes as a crucial advantage also in the deployment of the network monitoring infrastructure.

References

[1] Demetres Antoniades, Michalis Polychronakis, Spiros Antonatos, Evangelos P. Markatos, Sven Ubik, and Arne Øslebø. Appmon: An application for accurate per application network traffic characterization. In *In IST Broadband Europe 2006 Conference*, 2006.

[2] Gion Reto Cantieni, Gianluca Iannaccone, Christophe Barakat, Chadi Diot, and Patrick Thiran. Reformulating the monitor placement problem: Optimal network-wide sampling. Technical report, Intel Research, 2005.

[3] A. Ciuffoletti. The wandering token: Congestion avoidance of a shared resource. In *Austrian-Hungarian Workshop on Distributed and Parallel Systems*, page 10, Innsbruck (Austria), September 2006.

[4] Augusto Ciuffoletti. Secure token passing at application level. In *1st International Workshop on Security Trust and Privacy in Grid Systems*, page 6, Nice, September 2007. submitted to FGCS through GRID-STP.

[5] Augusto Ciuffoletti, Papadogiannakis Antonis, and Michalis Polychronakis. Network monitoring session description. Technical Report TR-0087, CoreGRID Project, July 2007.

[6] Gianluca Iannaccone, Christophe Diot, Derek McAuley, Andrew Moore, Ian Pratt, and Luigi Rizzo. The CoMo white paper. Technical Report IRC-TR-04-17, Intel Research, 2004.

[7] Hao Jiang and Constantinos Dovrolis. Passive estimation of tcp round-trip times. *SIGCOMM Comput. Commun. Rev.*, 32(3):75–88, 2002.

[8] Antonis Papadogiannakis, Alexandros Kapravelos, Michalis Polychronakis, Evangelos P. Markatos, and Augusto Ciuffoletti. Passive end-to-end packet loss estimation for grid traffic monitoring. In *Proceedings of the CoreGRID Integration Workshop*, 2006.

[9] Alistair Phipps. Network performance monitoring architecture. Technical Report EGEE-JRA4-TEC-606702-NPM NMWG Model Design, JRA4 Design Team, September 2005.

[10] Alistair Phipps. NPM services functional specification. Technical Report EGEE-JRA4-TEC-593401-NPM Services Func Spec-1.2, JRA4 Design Team, October 2005.

[11] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917 (Informational), October 2004.

[12] Massimo Rimondini. Emulation of computer networks with Netkit. Technical Report RT-DIA-113-2007, Roma Tre University, January 2007.

[13] Panos Trimintzios, Michalis Polychronakis, Antonis Papadogiannakis, Michalis Foukarakis, Evangelos P. Markatos, and Arne Øslebø. DiMAPI: An application programming interface for distributed network monitoring. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2006.