A distributed infrastructure for monitoring network resources

Augusto Ciuffoletti *, Yari Marchetti
Department of Computer Science - University of Pisa - Pisa

December 15, 2011

Abstract

When an infrastructure is used to process complex workflows by way of an Infrastructure as a Service (IaaS), network monitoring becomes mandatory to ensure the required quality of service and to optimize the utilization of the whole infrastructure.

In this paper we explore this scenario, evaluate the issues that come with the network monitoring operation, and propose a practical solution. To support our claims, we introduce a network monitoring infrastrucure that has been implemented as a *proof of concept* for the foundations of our solution.

Keywords: Network Monitoring, Network Measurement, XML Schema Description, Grid infrastructure, Service Oriented Architecture, CoreGRID.

1 Introduction

The monitoring of network resources, to evaluate metrics like available bandwidth, or delay jitter, is a key issue in the management of distributed infrastructures. In fact, complex workflows supported by such infrastructures may stress the performance of communication facilities, for instance when they require the displacement of large amounts of data. To ensure that the performance of the networking infrastructure meets the demand, the performance of each network element that takes an active part in the distributed computation must be monitored before triggering a data transfer operation, or prior to engaging in a data intensive computation. A use case that is presently quite popular occurs when the Grid paradigms are applied to an IaaS infrastructure.

Thus the purpose of the monitoring activity is to provide preliminar information about the availability of network resources, and to verify network resources performance during the operation. Note that here the importance of end-to-end Network Monitoring is emphasized, since the computational abstraction represented by the workflow implemented over the distributed infrastructure, possibily using network virtualization techniques, is incompatible with the classical link level view of the communication infrastructure [2].

Here we have to take into account an issue that differentiates end-to-end network monitoring from other instances of resource monitoring. If we look at how its complexity scales up with the size of the system, we observe that, while other resource monitoring activities (for instance the assessment of the available storage space) scales linearly with the size of the system, end-to-end network monitoring potentially grows with the square of the size of the system, and so does the quantity of data collected during a time unit.

To make a straightforward example, let us consider a system with n end-points, with no hints about which end-to-end network element is being used for a massive data transfer: network monitoring will necessarily cover all of the n*(n-1) network elements, possibily wasting most of the resources dedicated to its operation, and to the storage of its results.

Although simplistic, the example justifies our claim that end-to-end network monitoring must be selective in its targets: only a small fraction of end-to-end routes can be monitored at each time. As a consequence,

*Contact author e.mail: augusto@di.unipi.it

whatever be the criteria to select which are the network elements that need to be monitored, some sort of distributed infrastructure must exist that allows the selective activation of network monitoring. Such an infrastructure is not needed for other kinds of monitoring activity: for instance, data about computing resources can be collected continuously, and historical records kept, while preserving system scalability. In a sense, the peculiar nature of network monitoring pushes in the direction of a Service Oriented approach for its implementation, where a client submits a (network monitoring) request that is processed in an opaque way by a service provider.

In addition, network monitoring tools exhibit relevant peculiarities in their operation. When we consider other kinds of resources, like processing capabilities, the only compatibility requirement that applies to a measurement tool is that its results are compliant to a standard, for interoperability reasons. When we come to network monitoring tools, we observe that certain network monitoring methodologies require some sort of cooperation from the environment: even the simplest tool of this kind, the ICMP ping, requires that its packets are propagated, which cannot be given for granted. More complex tools further require that the end-points participating in a monitoring activity share some functionality: the well-known Iperf tool belongs to this category. Such fact may make the deployment of such network measurement methodologies awkward. According with a widely accepted terminology, we aggregate them in the category of *active* network monitoring techinques. The basic idea behind these tools is to inject a traffic pattern, and extrapolate from its treatment a pattern-independent metric. However this generalization step is in itself critical; the result may depend from hidden variables, for instance the credentials used when performing the measurement.

The analysis of the traffic produced by the application itself is in many cases a more appropriate solution: as opposed to the previous, we call this one *passive* network monitoring. It is immediately applicable to the monitoring of an ongoing application that makes use of network resources, but can be adapted also to the case of a *preventive* monitoring; for instance, to assess the performability of a given operation. In this latter case, a *benchmark* test, using the same pattern used in the real operation, is submitted to the infrastructure, and the performance of the end-to-end network element is observed. It is relevant to stress that in both cases network monitoring is mainly under control of the client application.

We identify two distinct approaches for the implementation of a passive monitoring solution:

- by analysis of the log files produced by the bandwidth consuming activity,
- by measurement of the traffic on network interfaces that are traversed by labelled traffic.

In both cases we observe that the monitoring activity is strictly related with the application: it is the task of the application, or of the workflow manager that orchestrates its execution, to configure the data acquisition phase, and to manage its results.

Another aspect that must be taken into account is that the network monitoring infrastructure should operate regardless of the tools that actually perform the logging or monitoring activity: in other terms, it is not the role of the infrastructure to dictate which back-end sensor is supported, and which is not. In other terms, although the restless evolution of networking technologies stimulates the creation of new network monitoring methodologies, this fact cannot entail the upgrade of the infrastructure. Therefore a long lived framework should be agnostic with respect to the back-end tools used to carry out the task. Instead it is advisable an architecture that envisions monitoring tools that can be dynamically *plugged in* a running deployment, possibly wrapped in an appropriate adapter. As a general rule it will be the task of the user application to indicate the tool appropriate to carry out the measurement, although a default back-end sensor may exist for certain measurement.

For a similar reason limiting the scope to a specific representation of the measurements is regarded as an intrusion in user discretionality: the data can be encoded in any suitable format, that may be customized by the user and *tool-aware*. Similarly to the way the Real Time Protocol (RTP) works, the monitoring framework we have in mind treats network measurements as opaque payload.

Summarizing, we establish four cornerstones for an end-to-end network monitoring architecture capable of managing the scalability challenge offered by distributed infrastructures that support complex, network intensive workflows:

- *demand driven*: its activity is not set by default, or with static configurations, but controlled by the workflow manager;
- *passive monitoring oriented*: only application-related traffic is analyzed in order to obtain the requested measurements;
- *tool transparent*: a new network monitoring tool can be made dynamically pluggable with minimal coding effort;
- *encoding transparent*: the network monitoring infrastructure should not interfere with the data structure that represents measurements.

Security issues need to be taken into account and access to network monitoring has to be restricted, its footprint being possibly relevant. In order to keep under control the monitoring activity, we consider as appropriate to delegate such activity to specialized, suitably configured units. Security concerns also hit network monitoring results; measurements should be collected and published only after checking the credentials of the requester, and they should generally regarded as subject to security restrictions.

This paper describes a *proof of concept* implementation, named *gd2*, that follows the above guidelines. In summary, it monitors the traffic within a set of end-points clustered into domains, and is operated by a number of agents in charge of controlling network monitoring activity. These agents receive client requests that describe the required monitoring activity, that may target inter-domain routes.

The paper is organized as follows: after introducing an abstract view of the architecture, we detail the internal structure of the basic agent, the Network Monitoring Agent, and introduce the formal description of the piece of data in charge of describing an instance of network monitoring activity. Finally we discuss the scalability of our approach.

2 The components of a demand driven network monitoring architecture

The Network Monitoring Infrastructure concept introduced in the previous section identifies implicitly three distinct functionalities: *producers* of traffic measurements, *infrastructure agents* that deliver requests to producers, and *consumers*, that issue requests and consume measurements. Such a triad corresponds, in spirit, to the one introduced in a *Grid Forum* report [1] that laid the foundations of the Network Monitoring Working Group [8] of that association. However, we note one relevant difference with respect to this architecture in the role played by the *infrastructure agents*, that replace the *directory* block in the referenced report. In our case we refer to a service which routes requests from *consumers* to *producers*. Such routing requires coordination within the network monitoring infrastructure, and must enforce security policies that restrict access to control functions. Both facts indicate the need of restricting the set of agents enabled to perform such actions, avoiding to implement direct access between consumer and producer. Therefore the role played by agents in the network monitoring infrastructure is more similar to that of a mediator, as in NPM [10], discussed in section 3.

In *gd2* architecture, an example of it is in figure 1, each 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). Such a set of entities is indicated as a *domain*. There are good reasons to introduce such concept (roughly the same that motivate its introduction in many aspects of networking):

- *reducing complexity* – one Agent concentrates the interface to the entities inside the domain;
- *security containment* – security issues can be managed using local credentials inside the domain;

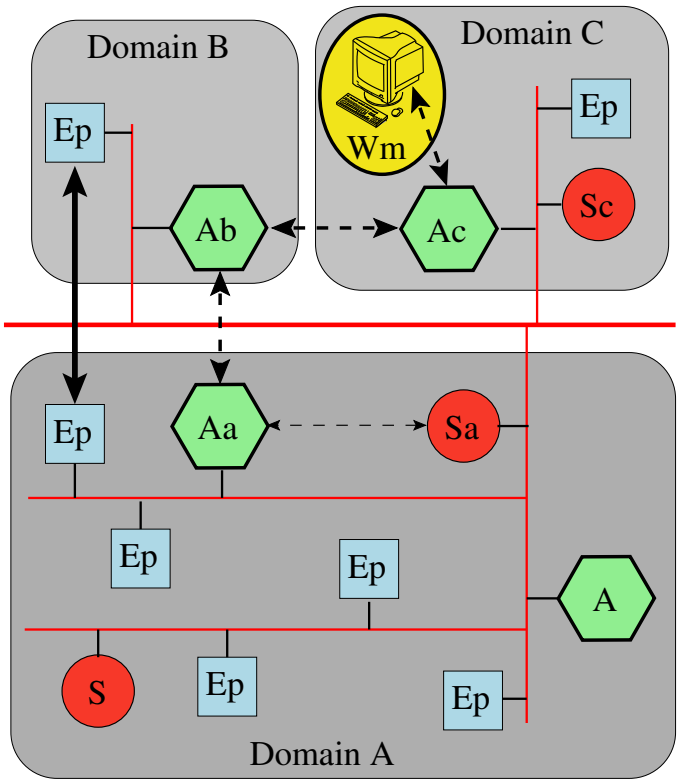

Figure 1: *gd2* components in a system with 3 Domains: **E** units represent generic monitoring endpoints, **A** labeled units representNetwork Monitoring Agents, **S** units represent Network Monitoring Sensors

- *limiting global state access* – only Agents have access to the global state, thus simplifying its management and ensuring security.

In Figure 1 the network is composed of 3 domains, Agents are represented by hexagons, Sensors by circles, network monitoring end-points by squares. A client of the network monitoring infrastructure *Wm* is placed in the *C* domain.

The arrows in Figure 1 illustrate the processing of a monitoring session. Let us consider that the client submits a complex workflow, which requires a massive data transfer between the two end-points connected by double arrows, located in domain *A* and *B*. The path connecting the two end-points starts with a branch inside Domain B, goes through the backbone that interconnects the three domains, enters Domain A through an internal backbone and finally reaches the target end-point.

The client *Wm* assembles a Request, and forwards it to *Ac*, a local Agent (dotted line), using a SOAP call. We consider that client credentials to submit requests have validity local to the domain.

Ac inspects the request and queries the DNS (or another equivalent service describing domain topology) in order to decide where it is appropriate to route the request. Before being forwarded, the request is encapsulated into an envelope, and signed by *Ac*. The request is then routed to *Ab*, the Agent in domain *B*, using another SOAP call. Unfortunately, there is no sensor available in domain *B*. In fact, the information about sensor capabilities is local to the domain, and is not global to the whole network: this justifies the non-optimality of *Ac* decision. Agent *Ab*, after checking request signature, concludes that it must route the request towards another candidate, and it selects *Aa*, an Agent of the domain hosting the other endpoint. The *Ac* signature in the request is replaced with *Ab* signature.

When *Aa* receives the request, after checking signature validity, it assigns the monitoring activity to *Sa*: the envelope is removed and the original request from *Wm* is used to (re)configure *Sa*, which starts collecting

measurements according with the request.

Measurements are now packaged as payload of UDP packets, streamed along a route that proceed backwards with respect to the path followed by the request: from *Aa* to *Ab*, to *Ac*. At each step the packet is signed by the sending agent. Finally *Ac* retrieves the payload and returns it to the client Wm. Data are streamed as long as the monitoring activity takes place, as indicated in the Network Monitoring request.

Concerning *security* aspects of our infrastructure, we note that we identify three distinct scopes for credentials. One is local to a *monitoring session*: the client and the sensor may agree and implement a specific security policy, for instance encrypting the data stream from the sensor to the client. The implementationof this agreement falls outside the scope of our infrastructure. Another extends within a *single domain*, as in the case of the signature used by Wm in order to access the local Agent, or the capability of a given Agent to control a Sensor. The last one defines the *membership of the Agents*, and its scope should extend to the whole Network. In essence any Agent should be able, upon receiving a request from an unknown peer, to check its signature. We observe that the problem is already solved in Internet applications, and we do not investigate it further: an LDAP directory containing public keys, or a DNS based solution would fit the purpose.

Also the information needed to route the requests can be retrieved from the DNS; inappropriate routing decisions, due to the assumption that the DNS stores only associations between IP addresses and domains, can be compensated by successive routing, as illustrated in the example. A directory of sensors, and related capabilities, available within the domain can be maintained, without incurring into scalability issues. The creation of loops is prevented since the route is recorded into the request, and routing decisions are also recorded in the *soft state* of each node: the presence of a soft state is needed to route the stream containing monitoring data.

The utilization of the reverse route to stream data packets is also introduced for reasons related to security: domain firewalls can be configured in order to allow incoming SOAP calls directed to agents. Once the agent has checked the credentials of the peer agent, UDP packets can be accepted from that source for the duration of the monitoring session.

As anticipated, the overall infrastructure does not make any assumption neither on the tool used to perform network measurements, nor on the nature of the measurements themselves. The adaptation of a new tool in the *gd2* infrastructure consists in writing a *plugin* component that is in charge of: i) driving the (possibly remote) Sensor using the data contained in the request, and ii) encoding the data returned by the sensor as specified in the request.

In order to have a better understanding of the operation of the *gd2* infrastructure, we briefly inspect the internal architecture of an Agent, and next we give a closer look to the structure of the XSD document that describes the Network Monitoring Request.

2.1 The Network Monitoring Agent

The services offered by a Network Monitoring 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 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 assume that security mainly addresses authentication: communication among Agents is not considered as confidential. Therefore we envision a public/private key scheme as adequate for our purpose. In order to control access to the membership, we assume the existence of an external entity in charge of key creation and assignment. Such 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 (but not encrypted, at least in principle), which can be checked using the public key.

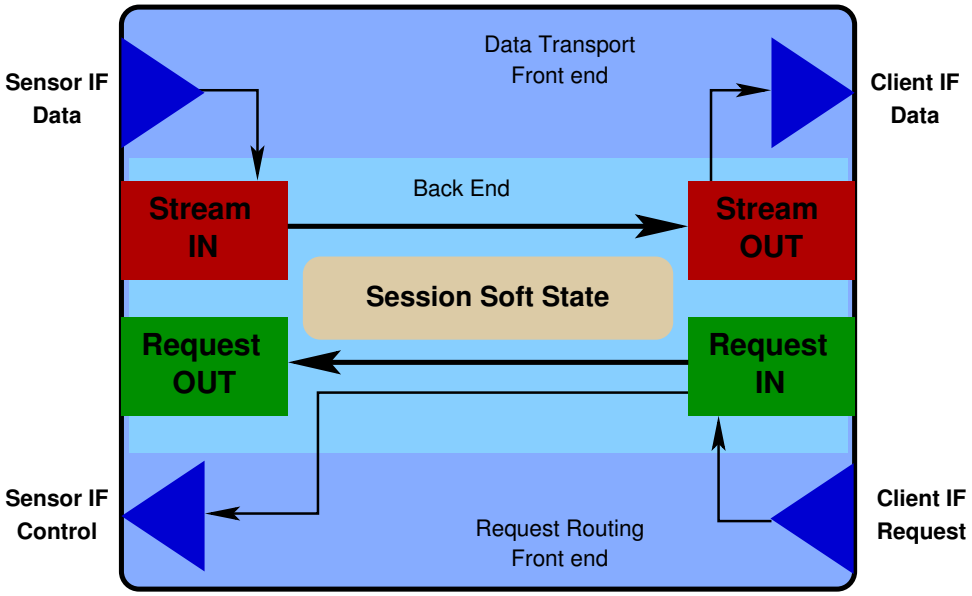

Figure 2: Internal architecture of a Network Monitoring Agent. The Back End interfaces are located in the innermost stripe

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 may happen that the information available to the Client that raises the request is not sufficient to determine the appropriate Agent for the task. In that case the front end fails to fulfill a request: however, the back end traps the failure, and resubmits the request to another Agent. We indicate as *Proxy* functionality the capability of re-routing a request.

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 its modality. The request must determine, either implicitly or explicitly, the features of the stream that returns the measurement to the Client. In section 2.2 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.

2.2 Summary of the Network Monitoring Session Description

In a Service Oriented perspective, this is the interface offered to access the Network Monitoring service: it plays a relevant role, so it is appropriate to inspect its internal structure.

The data contained in a Network Monitoring Session Description (for short Session Description) (see also [4] for an exhaustive description) is ideally split into three parts: one that is manipulated only by the Client, one that is updated by Agents, together forming an envelope carrying data that are relevant for the delivery, one that is delivered untouched from the Client to the Sensor and carries the modality of the monitoring activity. This kernel information corresponds to the Service interface.

The first part of the XSD document (see table 2 in the appendix) contains a **SessionId** string that is used to uniquely identify a session, and used also for data stream routing. The extent of a monitoring activity is specified by a **Schedule** element, which also contains indications of the resources required by the transport of the data stream. The **SessionId** and the **Schedule** characterize a Session, and are delivered from the Client to the Agents. They are not changed *en route*.

The Agents complement such data with others concerning the treatment of the Session: they are modified by the Agents themselves, and are represented as attributes. The **RequestFrom** element indicates the source of the request, and is used for admission control: in principle, several requesters may be indicated. The **NetworkElement** indicates the domains involved in the monitoring, and is used to select the appropriate Agents to deliver the request. The **Route** element is modified while the request is propagated, and it is used for backward delivery of the data stream.

The inner part of the Session Description (see table 3 in the appendix) consists of a complex element, the **MeasurementStream**, that details the kind of data that are requested from the monitoring activity. Such element is passed *as is* from agent to agent without being processed, and might be encrypted for security reasons.

The **MeasurementStream** element is made of a common part, and another which is specific for each measurement tool. The common part lists a **CharacteristicStreamId** attribute, a sampling period **SamplePeriod** together with the monitoring end-points (**SourceIP** and **DestinationIP**), which together identify and broadly characterize the monitoring activity. The tool specific part is one of a choice of elements: such elements may be specific for a given tool, or generically address any tool that may produce data conformant to a certain standard.

3 Relationships with other works

The **CoMo** [7], a passive monitoring infrastructure ideated by Intel, has been the primary source of inspiration for our design. Our infrastructure shares many of the basic requirements of CoMo, but we focus and propose solutions to a subset of them, and provide the tools to implement others. In addition, our infrastructure is influenced by a Grid-oriented approach, while CoMo tends to be telecom oriented. In the sequel, we briefly discuss these points.

The CoMo White Paper proposes three main challenges. From its abstract:

> ...(1) the system must allow any generic metric to be computed on the incoming query interfaces. (2) it must provide privacy and security guarantees to the owner of the monitored link, the network users and the users, and (3) it must be robust in the face of anomalous traffic patterns...

The three challenges are specifically addressed in our infrastructure: (1) using a tool-agnostic architecture and an open communication protocol, (2) using a three layers (network-domain-session) security scheme, (3) introducing a protocol that is not affected by the characteristics of observed traffic (but we exclude from our scope the monitoring tool). In addition we are specifically concerned by *scalability*, and we want our proposal to be valid in the Internet scale.

One relevant feature of our approach is that we do not propose an all-inclusive solution to the problem, but we identify a restricted set of sub-problems and well defined interfaces for extensions. This approach is evident in the option of considering specific monitoring tools as plugins in our infrastructure (we do not

want to bind our proposal to a given tool), in the opaque nature of the inner parts of the protocol (we are not interested in measurement representation), and in the reference to existing, well established tools for agent membership maintenance (namely, LDAP or DNS). CoMo shares this approach, by introducing plugin *modules* in its architecture, but extends the solution to cover both on-line monitoring (that does not require storage) and off-line monitoring (that requires storage).

In the introduction we motivate the distant nature of the two network monitoring approaches: one oriented to real-time streaming, the other mainly concerned with information storage (and indexing). The on-line approach is ideally coupled with network-aware applications: a running job represented with a complex workflow wants to verify that a certain FTP transfer proceeds regularly, in the present time.

Our focus on Grid system justifies our interest for on-line monitoring, which bypasses CoMo *core processes*: in CoMo terminology, our approach consists in passing data directly from the *capture* core process to the user. In *gd2* the intermediate transport architecture is completely transparent to data, that is never buffered. In CoMo data goes across three storage related *core processes*: *export* that manipulates and normalizes raw data, *storage* that buffers data for later reference, and a *query* front-end. From this perspective, the *gd2* approach is orthogonal to CoMo, and gives precise answers to scalability concerns.

The NPM **infrastructure** [10] is a pan-European proposal 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 GGF standards, and metadata, indicating what kind of data are available for a given network element. Such information is delivered to clients, whose role is typically to detect and 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 [11].

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 focuses on a far less complex task: determining where to fire the measurement session.

We conclude our discussion remarking that NPM and *gd2* in fact address two distinct problems, and each of them is a poor solution when applied to the problem for which it has not been explicitly designed. NPM 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 a measurement, 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 measurements 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 [12]: 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 description in the Sensor to Client stream, and can be used whenever network utilization has the characteristics of a flow. We plan to converge to an IPFIX compliant architecture, and an IPFIX interface for the *nprobe* sensor is under work.

A topic that is apparently distant is the design of an infrastructure for network management. To this purpose, the present Internet supports the SNMP protocol, but a more advanced architecture [14] is actively investigated. Such new architecture might integrate the infrastructure illustrated in this paper, embedding the *agent* functionalities into the hosts dedicated to network management. In [16] we find a transitional network monitoring infrastructure, bridging from IPv4 to IPv6, which could easily host the functionalities described in this paper.

A more structured framework envisions the integration of the infrastructure presented in this paper as one aspect of an IaaS able to support complex workflows. This option was announced in the *Grid Infrastructure Architecture* (GIA) presented in [3] and developed within the CoreGRID project: summarizing, the GIA identifies error recovery, security and accounting as non-functional requirements of a workflow, and describes an architecture where they are implemented using an infrastructure where the workflow management processes have access to a number of appropriate services. Among such services Network Monitoring is presented as one that returns a stream of measurements in response to a request, according to the abstraction presented in this paper.

Our work is marginally related to the vast literature on network monitoring *tools*: we mention two of them that have remarkably influenced our design. MAPI [15] was used as a reference for passive network monitoring tool during the conceptual design of our infrastructure [5], which was carried out in tight cooperation with the FORTH Institute in Greece in the frame of the CoreGRID project. For the experimental deployment we used NCAP [6], an easy to use passive monitoring tool for which we implemented a custom plugin.

4 Evaluation, proof of concept and experiments

This section is divided into two parts. First we explore the scalability of the concepts upon which our design is built, and next we introduce some experimental results that measure local footprints, like protocol efficiency, data rate for typical applications, CPU and memory usage for each session, collected using a Java prototype on a testbed with a few agents in distant countries.

With the purpose of giving a definition of scalability, we define a simplified context for an infrastructure that provides the IaaS: the whole computational capacity C of the infrastructure (aggegating storage and processors) is distributed over a network of nodes, and the number of nodes N grows linearly with the capacity of the system.

In such a context, *scalability* means that we want to avoid data structures or computations whose complexity grows faster than the computational capacity C: for instance, the case of one monitoring session for each end-to-end path is not scalable.

After this premise, we give a simple proof of the scalability of our basic assumption: the workload of a demand-driven network monitoring infrastructure is scalable, and it grows linearly with C. To simplify our argument, we assume that C is made available in *slices* of comparable capacity: each server contributes with one or more slices. Each *workflow* is granted access to an average of k slices, and we assume that k is small compared with, and independent from, the overall capacity C. As a consequence, the number of *workflows* in the system is large compared with, and linearly bound to, the number of slices (the computational capacity of the whole Grid), and can be estimated as C/k. Now, consider that each workflow submits a request for a distinct monitoring session for each of the network elements connecting the slices allocated for it, whose upper bound is estimated k^2: note that this statement is valid as long as network monitoring is associated with the resources allocated to the specific workflow. We compute the number of network monitoring sessions in the whole Grid as $(C/k) * k^2 = C * k$, that concludes our proof: the workload associated to network monitoring grows linearly with the capacity of the system.

The successive step is the analysis of the cost of each monitoring session. We split such cost into two

Name	Value	Unit	Sample size
Session duration	56331	seconds	NA
Total gd2 traffic	1.306	KBytes	NA
gd2 payload traffic	1.039	KBytes	NA
gd2 protocol overhead	267	KBytes	NA
gd2 protocol overhead (signatures)	240	KBytes	NA
activation delay (max)	2028	msecs	10
stream jitter (average)	1.78	msecs	88
stream jitter (max)	9	msecs	88

Table 1: Summary of experimental results

parts, one related to the request from the client to the sensor, another to the management of the stream from the sensor to the client.

The request is generated in a constant time, and its routing requires the existence of an infrastructure similar to the DNS (or even based on it). Each hop entails the processing of part of the *request description*, which accounts a $\log(N)$ complexity, assuming that resources are allocated in a balanced tree. Such price is payed once during the lifetime of a session.

Each packet in the data stream has a similar footprint, in case it is routed using the reverse path as indicated in the paper, while the payload is dependent on the kind of monitoring request, and does not depend on N, the size of the system.

Therefore the cost of each monitoring session is $\log(N)$ over time. Such result, which is acceptable, may be improved limiting the number of bounces, primarily in the reverse path: this can be obtained in case the *source routing* in the reverse path is avoided, at the expenses of security.

This allows to localize our analysis to the single workflow: the rate between the resources required (on the average) by a workflow, and those required by the network monitoring activity for the same workflow reflects the overall footprint of network monitoring. An administrative advice might be to limit the data rate associated with network monitoring for a given workflow (a field is provided for this purpose in the XSD) to a fraction of the requested data rate.

To evaluate the soundness of our design, a prototype has been implemented: by profiling its operation we evaluate the footprint of a single session (a summary is in table 1). To develop the prototype we followed an original approach that it is worth mentioning. We first developed the software in a virtual testbed, built using NETKIT [13], a tool based on the User Mode Linux technology. We took advantage of an environment where *restarting the system* was practically feasible, and most configuration and runtime conditions where under strict control and reproducible. This development phase lasted approximately 3 months, and was carried out in the course of the Master Thesis of Yari Marchetti [9]. The adaptation to a real environment took only a few weeks, and the result was presented in a public demonstration during the *CoreGRID Industrial Showcase* in July 2008, in conjunction with OGF23. We used a testbed deployed between Italy and Greece (thanks to the collaboration of the FORTH Institute in Crete), and the Client was resident on a laptop on the meeting site. The same testbed has been used for the experiments reported below, but the Client was connected to a POP of a commercial provider (TELECOM Italia).

To evaluate the efficiency and the stability of the infrastructure, we inspect the traffic generated by a session that monitors a long FTP transfer. The monitoring session lasts more than 15 hours, and returns a measurement of the data transfer rate every 15 seconds, using 3746 packets. The monitoring session measures the data transfer rate from a given IP address to another IP address on a given port: during the session, the data stream with monitoring data carries about 1.3 MBytes, divided into 1 MBytes of payload (i.e., measurements) and 300 KBytes of protocol overhead, corresponding to an efficiency of the protocol, in this specific case, around 75%. We observe that almost 90% of this overhead is related to security enforcement, and used for signatures (64 Bytes each).

The traffic generated overall is negligible, but protocol efficiency is significantly low. In order to reduce such overhead, more measurement data should be included in the same packet, thus complicating the struc-

ture of monitoring sessions: the interest for this option (which is supported by the current structure of the protocol) seems limited, given the low data transfer rate required.

One local parameter that may interfere with usability is the delay between the production of the request, and the activation of the measurement on the sensor. We measured such quantity in a set of 10 requests. The request was produced by the Client, and forwarded by an intermediate agent to the final agent that drives the sensor: we measured times in the order of the seconds (maximum 2 seconds). The result is considered acceptable, since relevant monitoring sessions usually last minutes, and therefore have time to complete their setup.

Another parameter that may interfere with the usability of the measurements is the jitter of the delay of packets composing the stream to the client. We observed an average jitter of 2 msecs in a small sample, with maximum below 10 msecs, which is negligible for foreseen applications.

Finally, we consider the overhead on local resources: processing power and RAM. Since the footprint of a single stream is not measurable, we stressed a obsolete laptop (hosting a Pentium 4, 1.5 GHz, 3000 BogoMips with 500 MBytes RAM) installing both a Sensor and an Agent, and activating as many monitoring sessions as possible, in the worst case where all session send data at the same time. The residual computing power available for Agent operation was 15% of the CPU capacity, and was saturated with 160 streams, while memory usage reached 50 MBytes. We extrapolate that a dual-core 2.4 GHz server might host about 1000 streams.

5 Conclusions

An end-to-end network monitoring infrastructure based on the concept of passive demand-driven sessions allows system size to scale up indefinitely: no matter how the interconnection infrastructure is organized, the *monitoring activity* will take up a constant fraction of the capabilities of the Grid. Concerning the capabilities spent in the *management* of a single activity, a $\log(N)$ term implies that, for each increment of one order of magnitude of the capacity of the system, each monitoring session incurs in a constant increment of workload dedicated to network monitoring. Such cost is the price paid to secure the infrastructure against intruders, since at each step, packets need to be signed by the sender.

In order to reach such results we need a specific infrastructure made of specialized agents that manage monitoring requests, submitted by clients and implemented by network sensors. Such an infrastructure can be either overlaid on the existing Internet, or integrated in a next generation Internet management infrastructure.

References

[1] Ruth Aydt, Dan Gunter, Warren Smith, Martin Swany, Valerie Taylor, Brian Tierney, and Rich Wolski. A grid monitoring architecture. Recommendation GWD-I (Rev. 16, jan. 2002), Global Grid Forum, 2000.

[2] Augusto Ciuffoletti. Monitoring a virtualized network infrastructure – an IaaS perspective. *Computer Communication Review*, 40(5):47–52, October 2010.

[3] Augusto Ciuffoletti, Antonio Congiusta, Gracjan Jankowski, Michal Jankowski, Norbert Meyer, and Ondrej Krajicek. Grid Infrastructure Architecture - a modular approach from CoreGRID. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies*, pages 46–54, Barcelona - Spain, March 2007. CoreGRID milestone M.IRWM.05.

[4] Augusto Ciuffoletti, Antonis Papadogiannakis, and Michalis Polychronakis. Network monitoring session description. In Domenico Talia, Ramin Yahyapour, and Wolfgang Ziegler, editors, *Grid Middleware and Services - Challenges and Solutions*, pages 79–93. Springer, 2008.

[5] Augusto Ciuffoletti and Michalis Polychronakis. Architecture of a network monitoring element. In *15th IEEE International Workshop on Enabling Technologies: Infrastuctures for Collaborative Enterprises (WETICE 2006)*, page 2, Manchester (UK), June 2006.

[6] Luca Deri. nCap: Wire-speed packet capture and transmission. In *IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, page 9, Nice-Acropolis, Nice, France, May 2005.

[7] 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.

[8] Bruce Lowekamp, Brian Tierney, Les Cottrell, Richard Hughes-Jones, Thilo Kielmann, and Martin Swany. A hierarchy of network performance characteristics for grid applications and services. Technical Report GFD-R-P.023, GGF Network Measurements Working Group, May 2004.

[9] Yari Marchetti. Architettura distribuita per il monitoraggio della rete di comunicazione in un sistema grid. Master's thesis, Dipartimento di Informatica - Università di Pisa, 2008.

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

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

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

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

[14] Jrgen Schnwlder, Aiko Pras, and Jean-Philippe Martin-Flatin. On the future of internet management technologies. *IEEE Communications Magazine*, October 2003.

[15] 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.

[16] Jilong Wang, Miaohui Zhang, and Jiahai Yang. *Managing Next Generation Networks and Services*, volume 4773/2007 of *Lecture Notes in Computer Science*, chapter Internet Management Network, pages 599–602. Springer Berlin / Heidelberg, 2007.

Appendix – XSD Schemas

```
<complexType name="NetworkMonitoringSessionType">
  <sequence>
   <element name="RequestFrom"
       type="nmsd:WorkflowMonitoringTaskType"
       maxOccurs="unbounded"/>
   <element name="Schedule"
       type="sched:NetworkMonitoringScheduleType"/>
   <element name="Route"
       type="nmsd:RouteStackType"
       minOccurs="0"/>
   <element name="NetworkElement"
       type="nmsd:NetworkElementType"/>
   <element name="MeasurementStream"
       type="nmsd:MeasurementStreamType"/>
  </sequence>
  <attribute name="SessionId"
      type="string"
      use="required"/>
 </complexType>
```

Table 2: Inter-agent part of the Network Monitoring Request Schema Description

```
<complexType name="MeasurementStreamType">
  <sequence>
   <element name="CharacteristicStream"
      minOccurs="1" maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element name="SamplePeriod"
         type="float"
         minOccurs="0"/>
      <element name="Path"
         type="path:NetworkMonitoringPathType"
         maxOccurs="unbounded"/>
      <choice>
       <element name="PingOptions"
          type="pt:PingOptionsType"/>
       <element name="MAPIOptions"
          type="am:MAPIMonitoringToolOptionsType"/>
       <element name="OGFCharacteristics"
          type="ogf:OGFCharacteristicsType"/>
      </choice>
     </sequence>
     <attribute name="CharacteristicStreamId"
          type="string"/>
    </complexType>
   </element>
  </sequence>
 </complexType>
```

Table 3: The measurement specific part of the Network Monitoring Request Schema Description