European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Architecture of a Network Monitoring Element

Augusto Ciuffoletti
augusto@di.unipi.it
CNAF-INFN
Via B. Pichat 6-2
I-40127 Bologna (ITALY)

Michalis Polychronakis
mikepo@ics.forth.gr
FORTH
Vassilika Vouton, P.O. Box 1385
71110 Heraklion, Crete, Greece

CoreGRID Technical Report
Number TR-0033
February 20, 2006

Institute on Grid Information, Resource and Workflow Monitoring Services

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Architecture of a Network Monitoring Element

Augusto Ciuffoletti
augusto@di.unipi.it
CNAF-INFN
Via B. Pichat 6-2
I-40127 Bologna (ITALY)

Michalis Polychronakis
mikepo@ics.forth.gr
FORTH
Vassilika Vouton, P.O. Box 1385
71110 Heraklion, Crete, Greece

CoreGRID TR-0033
February 20, 2006

Abstract

A Network Monitoring system is a vital component of a Grid; however, its scalability is a challenge. We propose a network monitoring approach which combines passive monitoring, a domain oriented overlay network, and an attitude for demand driven monitoring sessions. In order to keep into account the demand for extreme scalability, we introduce the solution to two problems that are inherent to the proposed approach: security and group membership maintenance.

1 Introduction

Grid applications require Storage, Computing, and Communication resources, and need to know the characteristics of such resources in order to setup an optimal execution environment. With current tools, monitoring and understanding the availability and status of Storage and Computing resources is sufficiently precise, can be translated into database schemas, and is used for experiments in system resources optimization. In contrast, monitoring of Communication resources is at a rather early stage, mostly due to the complexities of the infrastructure to monitor and of the monitoring activity.

Monitoring the network infrastructure of a Grid has a vital role in the management and the utilization of the Grid itself. While it gives to maintenance activities the basic information for identifying network problems and diagnosing the cause, thus contributing to Grid fault tolerance, it also provides to Grid-aware applications the ability to undertake actions in order to improve performance, as well as resource utilization. In the latter category, we also include accounting activities that are important when Grid resources are shared by different administrative authorities.

According to the Global Grid Forum (GGF) schema [2], the management of network measurements, which we call observations, is divided into three distinct activities: their *production*, their *publication*, and their *utilization*. Although the three activities tightly interoperate, based on carefully designed interfaces between them, each of them uses different tools that are appropriate for the specific activity. Network monitoring tools are used for the production activity, powerful databases for the publication, and various

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

other techniques, such as visualization tools for administration and workflow analysis, for the utilization. In this paper, we focus on the infrastructure related to production and publication.

For measurement production, we explore the usage of passive network monitoring techniques for deriving several metrics of interest for the quality of the Grid connectivity infrastructure, such as current traffic load, packet loss ratio, and round-trip time. For the publication activity, one of our main concerns is scalability, given the induced complexity when the various monitoring *producers* are increasing in size and monitoring data output. We use a domain-oriented overlay network which reduces the complexity of the task, thus improving the scalability of our architecture.

The rest of this paper is organized as follows. In Section 2 we discuss issues about the scalability of the integrated monitoring system, as well as a number of related security and privacy concerns. Section 3 describes the architecture of the Network Monitoring Element, while Section 4 presents the passive network monitoring component and the network metrics it currently supports. In Section 5 we describe the scalable caching mechanism of our system. Finally, Section 6 summarizes the main goals and issues addressed in this paper.

2 Scalability and security issues

One obstacle in the design of a scalable network monitoring system is the complexity of its layout. We observe that network monitoring should address network routes, not single links: this is the kind of information needed to optimize distributed applications, and cannot be derived from link observation (as discussed in [7]). Since each pair of Grid Services should be individually monitored, and results reported, this makes an $O(n^2)$ complexity for many aspects of network monitoring: from the size of the database that should contain monitoring results, to the number of pings that probe the system.

In order to make a network monitoring system scalable, we need to employ techniques that take into account this aspect of network monitoring. Failing to do this, brings us to a solution where the load on single resources progressively grows with system size. We combined a number of ideas in order to find a solution to this problem. Each of them is *per se* insufficient to solve the problem, but their combination should indicate a way through:

- monitoring shouldn't address single Grid resources, but groups with similar connectivity;
- monitoring tools shouldn't inject traffic, but observe existing traffic;
- monitoring activity should be tailored on application needs;

Only their integration can effectively control problem size, and, in some sense, the first two *open the way* to the application of the third one. Such approach can be compared with those at the foundations of the Astrolabe prototype [10], although the specific solutions adopted in our work are different. Let us examine and justify them in more detail.

2.1 Hierarchical overlay network

The *topology* of a Grid is always split into groups of resources reachable through a unique link, or with a load balanced set of links: the accessibility of resources within a group depends on the connectivity of the access link, and local administration avoids internal bottlenecks. Therefore the monitoring network can be simplified in order to monitor routes between ingress points: this mitigates, but does not solve, the scalability problems.

The experience with GlueDomains [8] showed that such an approach is feasible: Grid resources were divided into network monitoring *Domains*, thus allowing the monitoring of a Grid of tens of domains with negligible overhead and minimal administration efforts. However, this approach alone cannot scale to thousands of Domains.

One relevant result of GlueDomains experience is the identification of the role played by the agent that concentrates the network monitoring activity for a domain. In GlueDomains terminology, this is the *theodolite*, which serves as a reference point for active monitoring operated by other theodolites in other

domains. Such a role corresponds to a new resource in Grid architecture, which is mainly dedicated to network monitoring activity, which we call *Network Monitoring Element*.

Having associated this activity to a specific agent greatly simplifies network monitoring administration: configuration instructions are now targeted to the NM Element, which configures and runs network monitoring *sessions*. In GlueDomains, the configuration of monitoring sessions is under control, and mostly automatic: it is based on a centralized database which describes monitoring sessions.

2.2 Passive network monitoring

A NM Element should be ideally placed near the domain *border*: a place which is ideal to deploy *passive network monitoring techniques*. The main advantage of passive network monitoring, compared to active monitoring techniques, is its *non-intrusive* nature. Active monitoring techniques incur an unavoidable network overhead, which consists of probe packets which compete with user traffic. In contrast, passive monitoring techniques passively observe network traffic without introducing any network overhead. At the same time, they provide accurate measurement of fine-grained traffic characteristics.

A passive monitoring approach exhibits further points of interest. One is that its implementation is cleanly modularized: the most critical component is a daemon which captures packets flowing through a network interface, possibly sampling a subset of them, and processes them. Other components extract required measurements from data streams produced by the packet analyzer. Such an internal architecture allows user applications to tune measurement activities to obtain comparable data from distinct sources, tailored for the specific application [3].

Packet analyzer activity, however, interferes with the host architecture. Low level software needs to be modified to run packet analysis efficiently, and this activity degrades host performance. It comes natural to allocate this functionality to a specific host.

In addition, passive monitoring (as any monitoring activity) requires an adequate level of security: this not only to guarantee that monitoring data accessibility is restricted, but also to avoid Denial of Service conditions. The existence of specialized agents in charge of taking measurements is therefore appropriate in such a scenario, since it localizes sensitive activity and data.

2.3 Application driven sessions

Passive, domain oriented monitoring greatly reduces network monitoring overhead, yet does not solve complexity problems: individual NM element activity would still grow with system size, although such elements are now specialized and growth is slower.

The concept that really cuts down network monitoring complexity is an *application oriented approach* for controlling this activity: if network monitoring activity is bound to applications, it will increase linearly with system throughput, which seems reasonable.

Since network monitoring activity should depend on running applications, *Network Monitoring Sessions* should be configured on the fly after an explicit request. We expect a sort of locality of such requests, since each of them should span a limited range of domains: for instance, extend to the Storage Services that hold replicas of certain data, not to the whole list of available Storage Services.

Such an *on demand* approach stresses system security: only trusted applications should be able to start new monitoring sessions. Since, as we noted above, an adequate support for security is inherent to any network monitoring schema, this is a requirement shared with other aspects of our work.

2.4 Interface issues

A challenging aspect of an *on demand* approach is the interface an NM Element should offer to the outside. As a general rule, applications are not prepared to configure Network Monitoring tools, and then collect the results: instead, they expect that such measurements are already in GIS database, automatically collected by *preconfigured* network monitoring sessions. This attitude is not appropriate in an *on demand* scenario, and limits system scalability.

One way to cope with this attitude, although inefficient, is to instruct the GIS to interact with NM Elements. Given the flexibility of GIS implementations, such a capability might be included as a kind of GIS

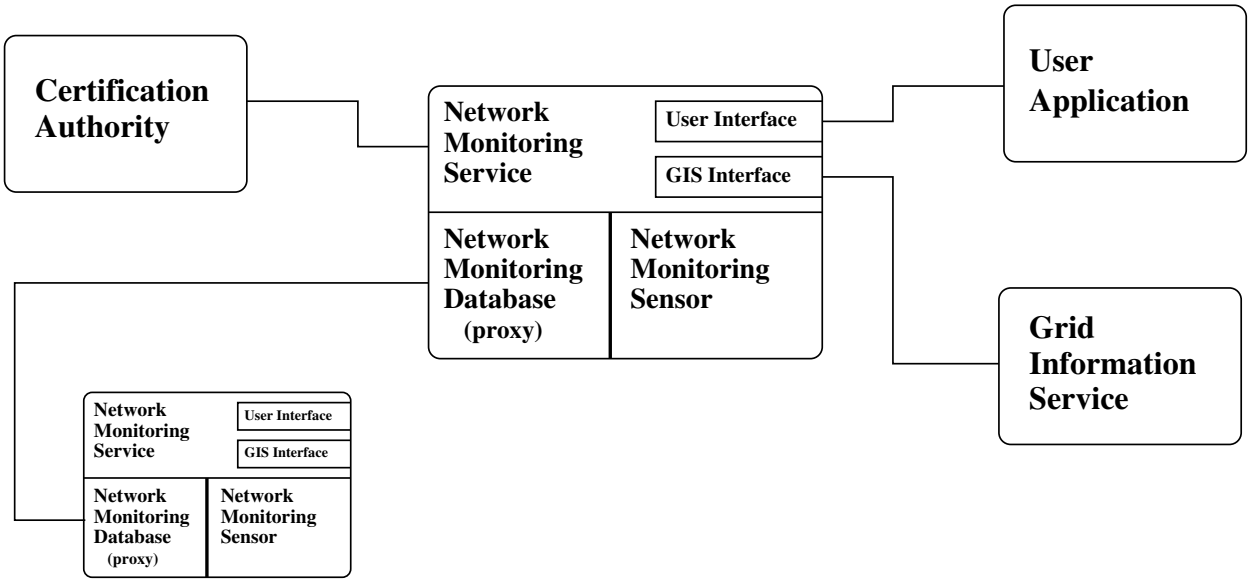

Figure 1: Interfaces between the NM Element and other Grid components.

plugin. However, the optimal way to connect the user applications to the NM Element would be through a publish-subscribe system.

Since we want our NM Element architecture to survive the realization of effective publish-subscribe mechanisms, we indicate a composite interface, which decouples the input, consisting of monitoring requests, from the output, consisting of observation records. As explained in section 3, this interface should effectively match the current state of the art, where most applications are based on *pre-configured* measurements, as well as with a more advanced scenario, where configuration input determines most of the activity of a NM Element, and information is delivered to the GIS, or to a more advanced P/S system.

2.5 Shared knowledge

The scheme described above is based on some knowledge shared by all NM Elements, which can be assimilated to a *group membership*. Such common knowledge consists of the certificates needed to enforce security. Such documents should also contain the description of Grid Resources partition into Domains.

Such data should be readily accessible by any Grid component, although throughput for access operations can be quite asymmetric: frequent read queries should be performed promptly, while infrequent updates can be treated lazily. This conforms to the expected performance of a directory service: we address this problem by replicating such directory on every NM Element, and using an epidemic algorithm in order to maintain replica consistency.

3 Architecture of the *Network Monitoring Element*

The internal structure of a *Network Monitoring Element* is layered according to the scheme in Figure 1. The upper layer is in charge of implementing the interfaces to the outside, offering a *Network Monitoring Service*.

The NM Service offers three distinct interfaces: one for user applications (resource brokers included), another for the *Grid Information Service* (GIS), and one that interacts with the Certification Authority. All of them are conceptually plugin modules for the NM Service implementation, and can be easily replaced or adapted to changing needs.

3.1 User Application Interface

The interface offered to the User Application allows the submission of a request of a specific monitoring session: such an *on demand* operation is required when the data from pre-configured sessions are not sufficient for the needs of the User Application (for instance, a resource broker). The NM Service will check application's credentials and local resource availability before accepting the request; in response, the user application will receive an acknowledgement.

A user application can also submit queries concerning the Domain Partition, such as the reference domain of a Grid Element. Such queries are forwarded to the Network Monitoring database, and the NM Element acts simply as an intermediary.

3.2 GIS Interface

The interface towards the GIS allows the publication of observations coming from network monitoring sessions: such module uses the interface provided by the GIS in order to include the results of the monitoring activity in the database managed by the GRID information system. Here we do not explore the architecture of this complex component, but we note that it should enforce certain access limits: for instance, the results of an *on demand* network monitoring activity should be visible, as a general rule, only to trusted users. The GIS should be informed of such limited access by the NM service which received the request.

We stress that results from on demand monitoring sessions are accessible only through the GIS, just like those coming from per-configured sessions: they are not returned in reply to the user application.

3.3 CA Interface

The role of the interface to the Certification Authority addresses two security issues:

- the management of monitoring sessions is protected from malicious threats that may result in a range of Denial of Service effects;
- the access to network monitoring results should be limited. Although this functionality is implemented by the GIS, the NM Service must nonetheless be able to provide the GIS with information about restricted access to certain data.

A caching facility is required to make scalable a system based on a Certification Authority [1], which we assume to be centralized: here we note that such a caching mechanism should include the certificates of user applications that are authorized to submit *on demand* network monitoring sessions. This aspect is further discussed in Section 5.

Summarizing, the NM Service takes configuration inputs from its User Application interface, sends the results of its monitoring sessions to the GIS engine, and caches certificates coming from the Certification Authority.

The lower layer is composed of two distinct modules that do not interact among each other. Their interface to the upper layer is explained in the following sections; here we simply outline their functionality.

3.4 Network Monitoring Sensor module

The *Network Monitoring Sensor* supports monitoring sessions: the implementation of sessions is delegated to specialized modules that take their configuration from the upper layer. We distinguish between passive and active sessions, depending on the technology used to carry out the monitoring activity. As explained in Section 2, scalability concerns suggest the usage of passive techniques as far as possible. Therefore, we consider the application of active monitoring techniques as a last resort.

Another relevant distinction divides sessions in *preconfigured sessions* and *on demand* sessions. While the former are configured directly by the NM Service module using the Grid topology described by the Network Monitoring Database, the latter are configured by an outside user application, through the NM Service. As a general rule, on demand sessions are active during a limited amount of time.

The results of the monitoring activity are delivered to the NM Service via dedicated one-way streams from the specific session.

3.5 Network Monitoring Database module

The *Network Monitoring Database* supports the information a NM Element has about the Grid. Such knowledge describes domain partition of the Grid, as well as its components: for each element in the Grid (NM elements, as well as Computing, Storage etc.), the database holds a certificate (which contains a reference to a domain) for the element, together with other relevant attributes.

The implementation of this component is vital to the scalability of the monitoring architecture: from one point of view, it should work *as if* it where centralized (e.g., the certificate of a generic element should be downloaded only once), while from another, it should work *as if* it were local (e.g., the NM service should find locally the Domain containing a given Storage). We have opted for the replication of the Grid database on each NM Element: its management is based on a scalable peer-to-peer protocol, which is explained in Section 5.

The physical configuration of the NM Element is still an open issue. An independent network interface is recommended for the NM Sensor: such an interface is busy with the filtering activity required by passive monitoring. Probably this would also indicate the adoption of an independent processing unit for the NM Element alone. Therefore one option is to implement the NM Element as a small cluster.

4 The Passive Network Monitoring Component

The Network Monitoring Sensor computes various network metrics using passive monitoring. Passive network monitoring techniques analyze network traffic by capturing and examining individual packets passing through the monitored link. As discussed in Section 2, passive network monitoring has a significant advantage compared to active monitoring: it is an inherent feature of passive monitoring that no additional traffic is injected into the network. In the following, we discuss the interface between the NM Sensor an the NM Service, as well as the details of the passive monitoring modules.

4.1 Interface

The NM Sensor receives measurement requests from the NM Service. Requests for new measurements may be made at any time, either on demand by a user needing some specific metric, or by the NM Service, which periodically instructs the NM Sensor to collect the necessary measurements for updating the current network view of the domain. The NM Service creates a new measurement session by sending an `create` request to the NM Sensor. The `create` request specifies the type of the measurement (c.f. Section 4.2), and any measurement-specific parameters. Since multiple measurement sessions may be active at the same time, each session is assigned a unique identifier `mid` for further reference. The identifier is returned by the NM Sensor to the NM Service upon the receipt of a new `create` request.

The creation of a new measurement session does not imply that the measurement will immediately begin upon the receipt of the `create` request. Instead, once the measurement session is created, the NM Service instantiates it in order to actually start the measurement by sending a `start` request. The request takes as parameter the `mid` of the session to be started. Correspondingly, a measurement session can be stopped by issuing a `stop` request with the corresponding `mid` of the session to be stopped. This allows the NM Service to configure a periodic measurement only once, and then starting and stopping it whenever necessary, without the need to configure it again. In this way, the NM Service has complete control over the duration of each measurement, and can alter it depending on the resources and the required granularity. At the same time, individual users may create new measurement sessions at will. Whenever a measurement session is not needed anymore, it can be terminated using a `close` request with the corresponding `mid`.

The computed results of a measurement are pushed back to the NM Service either on-the-fly, or upon the end of the measurement, depending on its type. For measurements that support both types, the desired behavior can be specified in the `create` request during the creation of the measurement session. For on-the-fly result export, the interval between consecutive messages can also be specified. Each result record contains the `mid` of the corresponding measurement session, a timestamp, and the actual result.

Function	Parameter (type)	Description
`create`	Type	the type of measurement: traffic load, packet loss, or RTT
	Arguments	Measurement-specific parameters
	Return value	The measurement session identifier (`mid`) or error type
`start`	Identifier	The `mid` of the session to be started
	Return value	Acknowledgement or error type
`stop`	Identifier	The `mid` of the session to be stopped
	Return value	Acknowledgement or error type
`close`	Identifier	The `mid` of the session to be terminated
	Return value	Acknowledgement or error type

Table 1: API of the NM Sensor.

4.2 Passive Network Monitoring Modules

In this section we discuss the network metrics that the NM Sensor can derive using different passive network monitoring modules. The modules are built on top of MAPI [6], an API for building passive network monitoring applications.[1] MAPI builds on the abstraction of the network flow, but in a flexible and generalized way. In MAPI, a *network flow* is generally defined as a sequence of packets that satisfy a given set of conditions. These conditions can be arbitrary, ranging from simple header-based filters, to sophisticated protocol analysis and content inspection functions.

The distributed version of MAPI [9] facilitates the programming and coordination of distributed passive monitoring sensors in a flexible and efficient way, by enabling users to express complex monitoring operations, precisely define the information they are interested in, and therefore balance the overhead they pay with the amount of information they receive.

Figure 2 presents the architecture of the NM Sensor. The back-end of the NM Sensor consists of the basic components of MAPI, namely the monitoring daemon (mapid) and the communication agent (commd). Packets are captured and processed by mapid: a user-level process with exclusive access to the captured packets [6]. Mapid is optimized to perform intensive monitoring operations at high speeds, exploiting any features of the underlying hardware. Commd handles the communication between the monitoring applications and mapid [9]. The monitoring modules are built as separate applications using MAPI. MAPI internally handles all the communication between the modules and the monitoring daemon, making it transparent to the application.

In the following sections we discuss the currently supported network metrics that are measured using passive monitoring.

4.2.1 Network Traffic Load

Monitoring of network traffic volume can be used for network profiling and planning. The network traffic load module provides traffic throughput metrics of varying levels of granularity by passively measuring the number of bytes transferred through the monitored link.

Measuring the aggregate bandwidth usage provides an indication of the utilization of the link, and thus, it can be used as a "bandwidth weather service" for estimating the future available bandwidth. For example, a user that needs to download a multi-Gigabyte file replicated across several mirror hosts at different domains may choose to download it from the domain with the most under-utilized link in the outbound direction. Besides aggregate throughput, fine-grained per-flow measurements are useful for observing the throughput achieved by specific applications or hosts.

For each new measurement session, the network traffic load module creates one or more network flows that measure the number of transferred bytes. For aggregate link utilization, the module creates two network flows

[1] MAPI is available at `http://mapi.uninett.no`

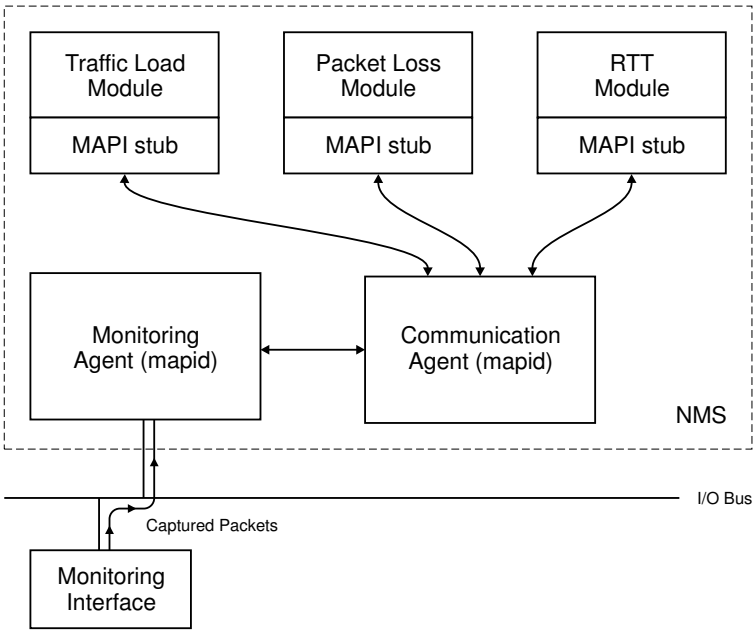

Figure 2: The architecture of the Network Monitoring Sensor.

for separating the incoming from the outgoing traffic (default behavior). More fine grained measurements can be obtained by specifying a `filter` parameter. MAPI supports generic BPF filters [5], as well as more fine-grained filtering using pattern matching in the packet payloads through string searching or regular expressions. This allows utilization measurements for specific protocols, ports, and IP addresses. Even for applications that do not use predefined ports, and thus their flows cannot be identified solely by the port number, the module uses protocol-inspection techniques for identifying their traffic. An example of capturing passive FTP transfers, which use dynamically negotiated ports for the actual file transfer, is presented in [6].

Filters for per-application measurements, if any, are specified in the `create` request. Results are reported in intervals of 60 seconds by default. This behavior can be altered during the creation of a measurement session.

4.2.2 Packet Loss Ratio

Packet loss occurs when correctly transmitted packets from a source never arrive at the intended destination. Packets are usually lost due to congestion, e.g., at the queue of some router or routing system problems. However, poor network conditions may also result to datagram damage. The packet loss ratio is a very important metric, since it affects data throughput performance and overall end-to-end data transfer quality.

An estimation of the packet loss between two domains can be measured by two cooperating passive monitoring observation points at the source and destination network domains. The monitors keep track of the packets of specific flows between the two domains that have been sent from the source network but have not arrived to the destination after a timeout period.

The packet loss module needs traffic information from both ends. This is achieved by creating a network flow in the local sensor, which keeps track of the outgoing packets with a destination IP address that belongs to the remote domain, and a second network flow at the destination domain, specified by the `dstdomain` parameter. The NM Element of the other domain is instructed to create a second network flow, which keeps track of the incoming packets with a source IP address that belongs to the local domain. The packet loss ratio is then estimated by correlating the data from the two network flows.

4.2.3 Round-Trip Time

The network Round-Trip Time (RTT) is the time taken for a packet to traverse the network from the source to the destination and back. RTT is one of the simplest network connectivity metrics, and can be easily measured using active monitoring tools like for example `ping`. However, it is also possible to measure RTT using solely passive monitoring techniques, which has the advantage of not injecting any traffic in the network. One such technique is based on monitoring the TCP connections that goes through a link.

RTT is estimated using the time difference between the `SYN` and `ACK` packets exchanged during the lifetime of existing TCP connections [4]. Each request specifies the destination domain for the end-to-end RTT measurement using the `dstdomain` parameter. The module then creates a network flow that captures the `SYN` and `ACK` packets of existing TCP connections between the two domains, in the unidirectional flow from the local to the remote domain. RTT is estimated from the time interval between the last-`SYN` and the first-`ACK` that is sent from the local to the remote domain. The accuracy of the measurement increases with its duration, since a longer duration allows for more TCP connections to be tracked, which gives a better RTT estimation.

5 Outline of a scalable caching mechanism

Security issues impose the existence of a centralized certification authority, which requires an efficient caching mechanism in order to be scalable [1]: we assume that certificates, which are needed by NM Elements to identify the source of their configuration inputs, are stored in the NM Database. Therefore we need to make scalable and secure the access to such database.

Such database has the interface of a directory service: the relevant characteristics of this structure are:

- a *small* read access latency;
- a *non bursty* network overhead;
- a *predictable* write access latency;

These requirements should be preserved regardless the size of the system: since we now have Grid Systems with hundreds of domains, it is useless to propose a scheme that exhibits scalability problems beyond a scale of 10^3 nodes.

We propose a distributed database management, in which each NM Element holds an (almost) complete replica of the whole database. This is required by the fact that queries have poor locality, and we cannot afford to keep only the *relevant* part available locally.

The broadcast of update operations is performed using a number of circulating tokens, each containing a stack of recently issued updates. The number of tokens circulating in the system is tuned automatically, based on a feedback mechanism that enables each NM Database Proxy, whenever it is necessary to inject (or remove) a token.

The peer-to-peer protocol on which the token circulation is based is made secure using the same certificates that are stored in the database itself: upon receiving a token from a neighbor, the NM Database Proxy authenticates it using the public key of the sender that is stored in the local database. In case the sender is not present in the local database, the receiver either downloads the certificate from the CA, or checks the certificate coming from the neighbor using CA public key.

The protocol is resilient to network and host failures, since it does not follow a preplanned path (or overlay network): it wanders within the system randomly.

Although mostly based on random decisions, the protocol promises an excellent stability and predictability: this conclusion is justified by simulation results reported in [7]. The load is evenly distributed in time and space, while update latency remains constant.

The drawback of this randomized solution is materialized in the update operations throughput: with respect to a deterministic solution, our approach processes less update requests per time unit. This drawback is compensated by the absence of traffic bursts, by a better resilience to failures, and by a more predictable behavior. In addition, we consider that the frequency of updates should be quite low, corresponding to events of creation or removal of a Grid Element.

Function	Parameter (type)	Description
`select`	SQL select query	the SQL-like query that returns the desired data
	Submitter	the id of the Element which submitted the query
	Return value	a data structure containing selected data
`update`	SQL update query	the SQL-like query that modifies the database
	Submitter	the id of the Element which submitted the query
	Return value	a data structure containing the query id
`check`	Query id	the id of the query to check
	Return value	a data structure containing the status of the query

Table 2: API of the NM Database Module.

To give a practical use case, in a Grid where the transfer of a 10KBytes token takes approximately 10 msec (at 1MByte per second), we obtain an update latency of 40 seconds (pessimistic delay between request acceptance and database updated on all replicas), with a throughput of one update every 5 seconds, regardless the size of the system (simulated in a system of 1000 NM Elements). These figures are reasonable when placed in our context: one event typically corresponds to the configuration of a new Grid Resource, which does not occur very often. To implement this level of service, the protocol uses a negligible fraction of the resources of the NM Element: one token is processed approximately every two seconds, with a bandwidth utilization of less than 5 KBytes per second, evenly distributed in time.

The *interface* offered by this module to the upper layer consists of the operations outlined in table 2.

The `select` function returns the desired data, while the `update` returns an acknowledgement. They take as parameters an SQL-like query and the id of the Element for which the NM Service issues the request.

The use of an SQL-like syntax for such queries shoud not deceive about the fact that the structure of the database, as well as the access policies, is far more restrictive than in a generic SQL database. In fact, it is intended that `select` has no side effect on the database, while `update` can modify only the records that describe the same Element that issues the request, as referenced in the call parameters.

While the `select` function is clearly synchronous, the `update` function is not: the acknowledgement reflects the fact that the request has been successfully queued, not necessarily performed. In order to check the (likely) completion of a requested update, the interface offers the `check` function, which takes as parameter the `update id` returned as an acknowledgement, and returns the current status of the update request, derived from the internal queue. The returned status should contain a prediction of the completion time.

6 Conclusions

We have outlined the internal architecture and the interface of a network monitoring service, specifically addressing security and scalability issues. Our work is firmly based on successful experiences: the MAPI prototype implemented at FORTH and the GlueDomains Network Monitoring service implemented by INFN. We combined these experiences in order to design an innovative architecture.

The basic building block is the *Network Monitoring Element*, a component of the Grid that concentrates its network monitoring capabilities. A generic Grid contains several instances of such component; each of them is responsible for monitoring communications to/from a small group of resources in the Grid, called a *Network Monitoring Domain*, which share an homogeneous connectivity with the rest of the Grid. A network monitoring element carries out its monitoring activity using passive monitoring, virtually without network overhead. Whenever needed it may be equipped with special purpose software and hardware.

The configuration of the monitoring network consists in describing a number of *Network Monitoring*

Sessions: we exclude manual intervention for such task, which should be carried out automatically, either using default sessions (deprecated, but still necessary), or according to the requests of user applications. The latter alternative requires capabilities external to the NM Element that are beyond the state of the art.

There are issues that remain open, after this report is closed. One is the physical configuration of the NM Element: further experiments are needed in order to understand what kind of hardware is required in different operation conditions.

Another is the operational description of the interfaces to external services, namely the Certification Authority, The Grid Information Service, and the user application. Since this issues are a matter of standardization, we have opted to describe their functionality, but without going into further technical detail. As far as possible, such interfaces should consist of *pluggable* modules, in order to ensure the survival of this work in a changing world.

References

[1] *Computer Networks*, chapter 8.5. Prentice Hall, 4th edition, 2003.

[2] 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.

[3] 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.

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

[5] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter Conference*, January 1993.

[6] Michalis Polychronakis, Kostas G. Anagnostakis, Evangelos P. Markatos, and Arne Øslebø. Design of an application programming interface for IP network monitoring. In *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 483–496, April 2004.

[7] S.Andreozzi, D.Antoniades, A.Ciuffoletti, A.Ghiselli, E.P.Markatos, M.Polychronakis, and P.Trimintzios. Issues about the integration of passive and active monitoring for grid networks. In *CoreGRID Integration Workshop 2005*, november 2005.

[8] Andreozzi Sergio, Ciuffoletti Augusto, Ghiselli Antonia, and Vistoli Cristina. Monitoring the connectivity of a grid. In *2nd Workshop on Middleware for Grid Computing*, pages 47–51, Toronto, Canada 2004.

[9] 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.

[10] Robert van Renesse and Kenneth Birman. Scalable management and data mining using astrolabe. In *International Conference on Peer to Peer Systems*, Cambridge, MA, USA, Mar 2002.