

Monitoring Virtual Networks with Lattice

Stuart Clayman

Dept of Electronic Engineering
University College London
London WC1E 7JE, U.K.
Email: sclayman@ee.ucl.ac.uk

Alex Galis

Dept of Electronic Engineering
University College London
London WC1E 7JE, U.K.
Email: a.galis@ee.ucl.ac.uk

Lefteris Mamas

Dept of Electronic Engineering
University College London
London WC1E 7JE, U.K.
Email: l.mamas@ee.ucl.ac.uk

Abstract—The use of the Lattice monitoring framework as a fundamental part of a *overlay management* system for virtual networks is presented. Lattice has been specially designed for monitoring resources and services in virtualized environments, including virtual networks. Monitoring of virtualized resources and services has many criteria which are not relevant for monitoring systems that are used for traditional fixed resources. We present the main aspects of the framework together with details of measurement transmission and meta-data encoding. Finally, the use of the Lattice framework for monitoring virtual machines executing under hypervisor control is presented.

I. INTRODUCTION

The Future Internet will have many of its core features and functions based on virtualized resources. The virtualization of resources has been in use for some time now, and there are several projects and initiatives working towards the virtualization of networks in the Future Internet, such as [1] [2] [3] [4]. The virtual resources which will be part of the Future Internet offerings include virtual machines which execute either virtual routers or virtual service elements. These will be combined to create the virtual networks. In order to manage these virtual resources, there needs to be a monitoring system which can collect and report on the behaviour of the resources.

In this paper we present the design characteristics and use of the Lattice monitoring framework as a fundamental part of an overlay management system for monitoring resources and services in virtualized environments, including virtual networks. Monitoring of virtualized resources and services has many features, requirements, and criteria which are not relevant for monitoring systems that are used for traditional fixed resources. As an example, such a monitoring framework should be able to support new functions like virtual machine migration. We present the main aspects of the Lattice framework, which has been designed for monitoring resources and services in virtualized environments, together with details of measurement transmission and meta-data encoding. We show the use of the Lattice framework for monitoring virtual machines executing under hypervisor control.

Virtualization is a mechanism which abstracts away from the underlying hardware, allowing more control and management of the virtualized resources without having to deal with the physical hardware. Virtualization is a technique which emulates some hardware or software on top of some real hardware. Network virtualization provides an abstraction that hides the complexity of the underlying network resources and enables

isolation-based protection, encouraging resource sharing by multiple network and application services. Host virtualization provides an abstraction which also hides the complexity of the underlying host. It too allows resource sharing, but also allows new hosts to be started and shutdown at a much faster rate than a real machine can be deployed.

Virtualization on its own is quite a useful mechanism, but to gain the greatest benefit it better to have a managed virtualization environment. Such management allows better control, monitoring, and analysis of the virtualized environments. For virtual networks, it is allowing functions such as quality of service, analysis of traffic flow, and network load balancing. By having management coupled with high-level decision making and universal control mechanisms (namely, orchestration), it allows us to build facilities which can harmonize and reconfigure networks on the fly in order to meet common goals. The deployment of monitoring probes which can return real-time data regarding the attributes of the real and the virtual networks allows these reconfigurations to adapt to the real world in real time.

In the work presented in this paper we use the Lattice monitoring framework as a real-time feed for a management interface to virtual network resources. We first present the issues relating to the management of virtualized network resources, discussing the key design requirements and how these are addressed in the AutoI project [5]. Then, in section III we present the Lattice monitoring framework, discussing its main features and also giving an overview of its design and implementation. This is followed, in section IV, by an evaluation of Lattice in the context of a monitoring system for virtual network elements. Finally, we state our conclusions and suggest further work.

II. MANAGEMENT OF VIRTUALIZED NETWORK RESOURCES

Virtual networks are aiming at better utilization of the underlying infrastructure in terms of (i) reusing a single physical or logical resource for multiple other network instances, or (ii) to aggregate multiples of these resources, in order to obtain more *functionality*, such as providing a pool of resources that can be utilized on demand. These resources can be network components such as routers, switches, hosts, or virtual machines. As stated these virtual machines can execute virtual routers or virtual service elements, but can also execute

network services such as name mapping systems. In virtual networks, a resource can be re-used for multiple networks or multiple resources can be aggregated for virtual resource. However, to manage these virtual resources effectively there needs to be a management system [6] [7].

A key issue in the management of virtual networks is the development of a common control space, which has autonomic characteristics [8] and enables heterogeneous network technologies, applications, and network elements to interoperate efficiently [9]. Management applications need to be adaptive to a rapidly changing environment with respect to specific network properties, and service or user requirements, as examples. This implies that management applications and network entities should be supported by a platform that collects, processes, and disseminates information characterizing the underlying network. An increased awareness regarding the properties and state of the network can bridge the gap between high-level management goals and the configuration that achieves them. In this respect, we consider an infrastructure that manages both information flow and processing within the network as an important stepping-stone towards this objective.

Key design requirements of an information management infrastructure are: (i) information collection from the sources (e.g., network devices), (ii) information processing that produces different information abstractions, and (iii) information dissemination to the entities that exploit that information. It is common that such design approaches aggregate information using aggregate functions. Consequently, real-time monitoring of network parameters may introduce significant communication overhead, especially for the root-level nodes of the aggregation trees. Information flow should adapt to both the information management requirements and the constraints of the network environment, one example being: changes in the information collection configuration.

Such a management system has been devised in the AutoI project [5] [6]. The AutoI project has developed a *management overlay* which is a self-managing resource overlay that spans across heterogeneous networks and supports service mobility, security, and quality of service. This overlay consists of a number of distributed management systems, which are devised with the help of five abstractions. Each abstraction has its own core functionality and is represented as a *plane* within the whole overlay system. These planes interact with each other using well defined interfaces. The planes for the AutoI management overlay are called (i) the Orchestration Plane, (ii) the Service Enablers Plane, (iii) the Knowledge Plane, (iv) the Management Plane, and (v) the Virtualization Plane. Together these distributed systems form a software-based control network infrastructure that runs on top of physical networks. It uses ideas proposed in [10] which has a unified solution that includes cognitive techniques and knowledge management. AutoI decouples information management from other network management functionalities. The latter are considered to be part of either the Management plane or the Orchestration plane.

In summary, the AutoI *management overlay* is a manage-

ment infrastructure that collects, processes, and disseminates information from and to the network entities and management applications, acting as an enabler for self-management functionality. For example, the management overlay can regulate the information flow based on specific characteristics of the network environment (such as the network topology) and performance requirements (such as low data rate).

To feed all the required and relevant information into the management overlay there needs to be a monitoring system that can collect such data from probes or sensors in the network environment and in the virtual resources. It can then report that data into the management overlay.

For full operation of a virtual network, we need to remember that monitoring is a vital part of the full control loop that goes from the network management, through a control path, to the probes/sensors which collect and send data, back to the network management which makes various decisions based on the data. Consequently, the monitoring is a small but fundamental part of network management as it allows the integration of components in all of the layers. The process of monitoring virtual resources is similar to the process of monitoring traditional network resources, as seen in [11] [12] [13] [14] [15], however, due to the high elasticity and migration capability of virtual resources, a different design is needed. This paper presents such a new design.

III. THE LATTICE MONITORING FRAMEWORK

A management system for the Future Internet such as the management overlay just discussed, requires a monitoring system that can collect all the relevant data in an effective way. The monitoring system has to have a minimal runtime footprint and not be intrusive, so as not to adversely affect the performance of the network itself or the running service applications. As a consequence, we need to ensure that the management components only receive data that is of relevance. In a large distributed system there may be hundreds or thousands of measurement probes which can generate data. It would not be effective to have all of these probes sending data all of the time, so a mechanism is needed that controls and manages the relevant probes.

Existing monitoring systems such as Ganglia [16], Nagios [17], MonaLisa [18], and GridICE [19] have addressed monitoring of large distributed systems. They are designed for the fixed, and relatively slowly changing physical infrastructure that includes servers, services on those servers, routers and switches. However, they have not addressed or assumed a rapidly changing and dynamic infrastructure as seen in virtual environments. In the physical world, new machines do not appear or disappear very often. Sometimes some new servers are purchased and added to a rack, or a server or two may fail. Also, it is rare that a server will move from one location to another. In the virtual world, the opposite is the case. Many new hosts can appear and disappear rapidly, often within a few minutes. Furthermore, the virtual hosts, can be migrated from one network to another, still retaining their capabilities.

It is these characteristics that provide a focus for Lattice. We have determined that the main features for monitoring in a virtualized network environment which need to be taken account of are:

- *scalability* - to ensure that the monitoring can cope with a large numbers of probes
- *elasticity* - so that virtual resources created and destroyed by expanding and contracting networks are monitored correctly
- *migration* - so that any virtual resource which moves from one physical host to another is monitored correctly
- *adaptability* - so that the monitoring framework can adapt to varying computational and network loads in order to not be invasive
- *autonomic* - so that the monitoring framework can keep running without intervention and reconfiguration
- *federation* - so that any virtual resource which reside on another domain is monitored correctly

To establish such features in a monitoring framework requires careful architecture and design.

A. Producers and Consumers

The monitoring system itself is designed around the concept of producers and consumers. That is there are producers of monitoring data, which collect data from probes in the system, and there are consumers of monitoring data, which read the monitoring data. The producers and the consumers are connected via a network which can distribute the measurements collected.

The collection of the data and the distribution of data are dealt with by different elements of the monitoring system so that it is possible to change the distribution framework without changing all the producers and consumers. For example, the distribution framework can change over time, say from IP multicast, to an event bus, or a publish / subscribe framework. This should not affect too many other parts of the system.

B. Data Sources and Probes

In many systems probes are used to collect data for system management [16] [20]. In this regard, Lattice will follow suit. However, to increase the power and flexibility of the monitoring we introduce the concept of a data source. A data source represents an interaction and control point within the system that encapsulates one or more probes. A probe sends a well defined set of attributes and values to the consumers. This can be done by transmitting the data out at a predefined interval, or transmitting when some change has occurred.

The measurement data itself is sent via a distribution framework. These measurements are encoded to be as small as possible in order to maximise the network utilization. Consequently, the measurement meta-data is not transmitted each time, but is kept separately in an information model. This information model can be updated at key points in the lifecycle of a probe and can be accessed as required by consumers.

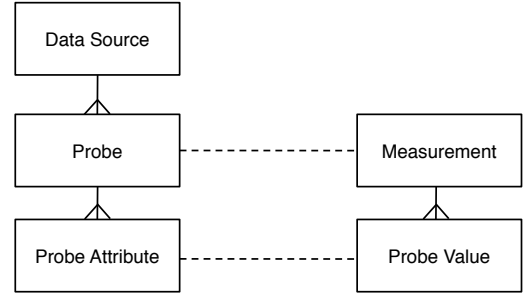


Fig. 1. Relationship Model

C. Distribution Framework

In order to distribute the measurements collected by the monitoring system, it is necessary to use a mechanism that fits well into a distributed architecture such as the management overlay. We need a mechanism that allows for multiple submitters and multiple receivers of data without having vast numbers of network connections. For example, having many TCP connections from each producer to all of the consumers of the data for that producer would create a combinatorial explosion of connections. Solutions to this include IP multicast, Event Service Bus, or publish/subscribe mechanism. In each of these, a producer of data only needs to send one copy of a measurement onto the network, and each of the consumers will be able to collect the same packet of data concurrently from the network.

D. Design and Implementation Overview

Within Lattice there are implementations of the elements presented in the relationship model shown in figure 1. In this model we see, a DataSource which acts as the control point and a container for one or more Probes. Each Probe defines the attributes that it can send. These are set in a collection of ProbeAttribute objects, that specify the name, the type, and the units of each value that can be sent within a measurement.

When a Probe sends a Measurement, the Measurement has a set of values called ProbeValues. The ProbeValues that are sent are directly related to the Probe Attributes defined within the Probe.

When the system is operating, each Probe reports the collected measurement to the Data Source. The Data Source passes these measurements to a networking layer, where they are encoded into an on-the-wire format, and then sent over the distribution network. The receiver of the monitoring data decodes the data and passes reconstructed Measurements to the monitoring consumer. Encoding measurement data is a common function of monitoring systems [16] as it increases speed and decreases network utilization.

In Lattice, the measurement encoding is made as small as possible by only sending the values for a measurement on the data distribution framework. The definitions for the ProbeAttributes, such as the name and the units are not transmitted with each measurement, but are held in the information model and are accessed as required.

Field	Type
sequence No	long
Probe ID	long
Type	string
Timestamp	long
Delta Time	long
Service ID	long
Group ID	long
No of Probe Values (N)	integer
Probe Values ($\times N$)	Probe Value Structure

TABLE I
PACKET STRUCTURE OF A MEASUREMENT

Field	Type
Field No	integer
Type Code	byte
Value	depends

TABLE II
PACKET STRUCTURE OF A PROBE VALUE

The encoding for these measurements is presented in more detail here.

1) *Measurement Encoding*: This section presents the structure of a Measurement for its transmission over a network.

First we see the main data structure for an encoded Measurement, as seen in table I.

The structure for each probe value is shown in table II. There is one of these structures for every ProbeValue in a measurement.

Table III presents each type and its associated code, together with the encoding strategy for each type. The current implementation is written in Java, and the output for each type currently uses XDR [21]. As such each type defined here uses the same byte layout for each type as defined in the XDR specification.

All of this type data is used by a measurement decoder in order to determine the actual type and size of the next piece of data in a packet.

2) *Information Model Encoding*: For the implementation of the Information Model we have the inclusion of a Distributed Hash Table (DHT) for the distributed information model. This allows the receivers of Measurement data to lookup the fields received to determine their names, types, and units. The information model nodes uses the DHT to interact among one another.

The DHT implementation used is based on the Kademlia model [22]. The implementation has a strategy for converting an object structure into a path-based taxonomy for use as keys in the DHT. The IDs of the Data Sources and the IDs of the Probes are important elements of this taxonomy.

For each Data Source, the keys and values shown in table IV are added to the DHT. For each Probe, the keys and values shown in table V are added to the DHT.

Using the encoded data from the information model, any of the consumers of the monitoring data, in particular the management overlay, can evaluate all the meta-data of the measurements.

Type	Code	Encoding Format
boolean	Z	Writes a boolean as a 1-byte value.
byte	B	Writes a byte as a 1-byte value.
char	C	Writes a char as a 2-byte value, high byte first.
short	S	Writes a short as two bytes, high byte first.
integer	I	Writes an integer as four bytes, high byte first.
long	J	Writes a long as eight bytes, high byte first.
float	F	Creates an integer representation of the specified floating-point value according to the IEEE 754 floating-point "single format" bit layout and then writes that integer value as a 4-byte quantity, high byte first.
double	D	Creates a long representation of the specified floating-point value according to the IEEE 754 floating-point "double format" bit layout and then writes that long value as an 8-byte quantity, high byte first.
string	"	Writes a string using a modified UTF-8 encoding in a machine-independent manner.
byte array	J	Writes an integer (as 4 bytes), which is the length of the array, plus all the array bytes to this output stream.
List	L	Writes out the number of list items as an integer (4 bytes), the element type, using the type code from this table, and then the elements themselves are written out using the encoding strategy outlined in this table.
Table	T	Writes out the no of rows in the table as an integer (4 bytes), the no of columns in the table as an integer (4 bytes), then for each column it writes the column name as a String followed by the column type, using the type code from this table. Finally all of the elements of the table are written out using the encoding strategy outlined in this table.

TABLE III
CODE FOR EACH TRANSMITTED TYPE

Key	Value
/datasource/datasource-id/name	datasource name
/probe/probe1-id/datasource	datasource-id
/probe/probe2-id/datasource	datasource-id
...	...
/probe/probeN-id/datasource	datasource-id

TABLE IV
INFORMATION MODEL ENTRIES FOR A DATASOURCE

Key	Value
/probe/probe-id/name	probe name
/probe/probe-id/datarate	probe data rate
/probe/probe-id/on	is the probe on or off
/probe/probe-id/active	is the probe active or inactive
/schema/probe-id/size	no of attributes N
/schema/probe-id/0/name	name of probe attribute 0
/schema/probe-id/0/type	type of probe attribute 0
/schema/probe-id/0/units	units for probe attribute 0
/schema/probe-id/1/name	name of probe attribute 1
...	...
/schema/probe-id/N/units	units for probe attribute 0

TABLE V
INFORMATION MODEL ENTRIES FOR A DATASOURCE

Summary of the Lattice Monitoring Framework

The Lattice framework provides a platform from which various monitoring systems can be built. It provides the building blocks and the glue from which specific monitoring elements can be devised.

We can write probes which collect data for any specific purpose and then write consumers which process that data in any way necessary. Lattice itself does not provide a pre-defined library of probes, data sources, or consumers. Rather, it allows the probes, data sources, and consumers to be written and deployed as needed. As we will see in the next section, we have built a monitoring system for virtual network management in AutoI, for which we have written probes for monitoring physical and virtual machines. For different applications, we can write a completely different monitoring system that is based on the Lattice monitoring framework. As an example of this, we have written a monitoring system for the service cloud project RESERVOIR [23] [24] for which we have written separate probes for 3 layers of the architecture.

IV. VIRTUAL NETWORK MONITORING EVALUATION

The use of Lattice allowed us to build a management infrastructure that collects, processes, and disseminates network and system information from/to the network entities at real-time, acting as an enabler for self-management functionality, as described in [7]. In this section we define how the management overlay utilizes measurement data from probes monitoring virtual machines.

As the monitoring framework integrates into the management overlay, it has the capability to provide information for the management overlay, with more than the measurements from the probes. Within the management overlay the Knowledge Plane holds details on all the elements of the managed network, for example, data on all of the physical machines and all of the virtual machines. This Knowledge Plane is implemented as a distributed storage system as can have data added or removed from many nodes. This functionality works well with Lattice, as Lattice holds all of the meta-data for the probes and the measurement attributes in an information model. The information model can be held in various ways, but for complete integration this uses the same system as that is used by the management overlay for its Knowledge Plane. That is, all meta-data for monitoring is held by the Knowledge Plane of the management overlay, and so the management overlay can easily lookup the attributes of any measurement as it arrives. The integration and the implementation of these components has been described in [6].

For the evaluation of Lattice, we have built a testbed and created some virtual machines that execute on the testbed. We have developed probes which can collect data from the physical machines and from the virtual machines. The virtual machine monitoring approach developed is a key component of the management overlay for virtual networks as the Lattice monitoring framework collects data about those virtual machines.

In the next sections we will describe the testbed and the monitoring of physical and virtual machines in more detail.

A. Testbed

The testbed that has been created for evaluating Lattice within a management overlay environment and for monitoring the virtual machines is presented.

There is a collection of six servers in the testbed, 4 of the servers have 2 Quad-Core AMD Opteron(tm) Processor 2347HE CPUs (giving 8 cores) running at 1.9GHz together with 32GBs of RAM. The other 2 servers have a single Intel Core 2 Quad CPU Q9300 (giving 4 cores) running at 2.50GHz and 32GB of memory. All of them are connected with shared storage via NFS. The connectivity of the servers is shown in figure 2.

B. Monitoring Physical Machines

To monitor the physical machines in the testbed we created three probes. The monitor CPU usage, memory usage, and network usage.

1) *CPU Probe*: This probe collects data on CPU usage for each core of a server. It runs regularly in order to collect this data in real time. The usage data collected includes the number of jiffies in (i) user space, (ii) being nice, (iii) in system space, and (iv) idling, since the last time a measurement was taken. These raw values are then converted into a percentage of used time for each of the four values. (A jiffie is $1/100^{\text{th}}$ of a second).

2) *Memory Probe*: This probe collects data on memory usage on the server. It too runs regularly to collect data in real time. It provides data on (i) the total memory in the server, (ii) the free memory, (iii) the memory used by the operating system, (iv) the memory used less the amount of cache and buffer space, (v) the amount of cache, and (vi) the amount of buffers. All of these are presented in kilobytes.

3) *Network Probe*: This probe collects data on the network interface of a server. By running regularly it can determine the amount of traffic on the interface for each sampling. It provides data on (i) the no. of bytes received, (ii) the no. of packets

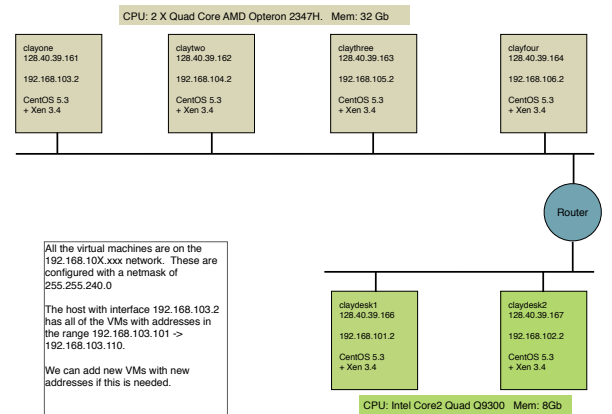


Fig. 2. Testbed setup

Attribute No.	Attribute Name	Attribute Type	Attribute Units
0	in-bytes	Integer	bytes
1	in-packets	Integer	packets
2	in-errors	Integer	packets
3	in-dropped	Integer	packets
4	out-bytes	Integer	bytes
5	out-packets	Integer	packets
6	out-errors	Integer	packets
7	out-dropped	Integer	packets

TABLE VI
DEFINITION OF A NETWORK MEASUREMENT

Field No.	Field Type	Value
0	I	5344
1	I	37
2	I	0
3	I	0
4	I	6644
5	I	38
6	I	0
7	I	0

TABLE VII
VALUES OF A NETWORK MEASUREMENT

received, (iii) the no. of packets received with errors, (iv) the no. of packets received which were dropped, (v) the no. of bytes transmitted, (vi) the no. of packets transmitted, (vii) the no. of packets transmitted with errors, (viii) the no. of packets transmitted which were dropped.

As we described earlier in the paper a Probe has to define using ProbeAttributes the values that will be transmitted in a measurement. There will be a value in the measurement for each defined attribute. For the Network Probe we have defined the attributes outlined in table VI The values for the attributes from this measurement are in table VII.

C. Monitoring Virtual Machines

A working implementation of the virtual machine monitoring has been developed. This implementation acts as a foundation for doing distributed monitoring of virtual networked resources and provides the values for the management overlay.

We have defined and written probes which provide information on the many physical hosts that are used as a platform for running a virtual machine hypervisor, as seen in the previous section. The hypervisor is the core component of many cloud computing environments and is used to run multiple virtual execution environments (VEEs). As well as writing probes to gather data from physical machines, we have evaluated how the distributed monitoring framework can interact with the hypervisor in order to extend the monitoring capabilities to include virtual machines running within the hypervisor. Our work in this area has tried to address the flexible and adaptable requirements of virtual machines and virtual networks and how these fit in with *libvirt* [25] (the library that provides access to various hypervisors through an abstraction layer). Although the test bed uses the Xen [26] hypervisor, we chose

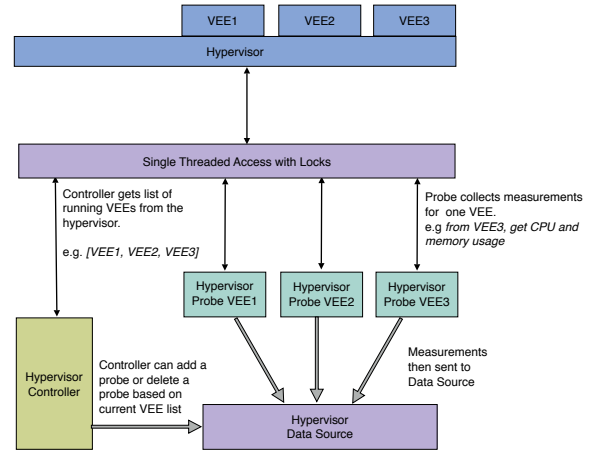


Fig. 3. A Probe for Monitoring Virtual Machines using libvirt

to interact with libvirt as it abstracts away any differences in hypervisors. Therefore, if we changed the actual hypervisor used, the probes would still continue to function correctly.

The current design and implementation for monitoring virtual machines is dependent on the fact that a virtual machine can be created, can execute, and can shutdown at run-time whilst the monitoring is still running. In essence, the hypervisor presents a collection of machines that changes over time. From the consumers point of view, it is important that the virtualized machine should look like an individual host, so we ensure that there is one probe sending data for each virtual machine. The alternative is to have one probe for the whole hypervisor, but using this approach then every measurement will contain data for every virtual machine currently executing on the hypervisor. The consumers will not see individual hosts, but a collection of them. Consequently, virtualized hosts will have to be treated differently from real hosts, which is not a desirable situation.

To ensure that there is one Probe per virtual host, and to accommodate the dynamic nature of the hypervisor we have architected a solution shown in figure 3. In the top part of figure 3 we can see, in blue, the hypervisor, which is actually a combination of both libvirt and Xen, together with the virtualized execution environments (VEEs). To get data from the hypervisor there is a HypervisorController which gets a list of running VEEs from the hypervisor, on a regular basis. The Hypervisor Controller compares the current list of VEEs with the list retrieved last time. From this it determines (a) if there is a new VEE, in which case it asks the HypervisorDataSource to add a new HypervisorProbe for that VEE, or (b) if a VEE has shutdown, in which case it asks the HypervisorDataSource to delete the HypervisorProbe for that VEE.

As stated earlier in the paper, a data source represents an interaction and control point within the system, so it is the data source that is responsible for adding and deleting the probes and to also collect any measurement data from the probes and to pass it onto the network for the consumers. In this case, the HypervisorProbes each run independently of each other,

in their own thread, and collect data regarding their assigned VEE at a regular interval. The data collected is structured that same as data collected from a real physical host, such as CPU and memory usage.

The final element of the figure 3, is one that provides single threaded access, via libvirt, to the hypervisor by using locks. The reason for this is explained in the next section.

D. Issues Arising

This section highlights some issues that have arisen whilst monitoring virtual execution environments.

After much investigation, we realised that we needed single threaded access to the hypervisor using locks. This is strictly necessary as the libvirt implementation is only capable of doing one request at a time, but has no locking or serialization mechanisms of its own. In our initial multi-threaded implementation, interacting with libvirt directly can cause libvirt to fail, or in extreme case the whole run-time system to crash. This is obviously not suitable for an autonomic system that is meant to operate continuously. By adding an extra intermediate element which is single threaded, locks out concurrent threads, and provides caching, we were able to build a stable probe for VEEs.

When collecting data for the network traffic of a VEE, through libvirt and the hypervisor, the result was always zero. See the following list from *ifconfig*.

```
vif1.0 Link encap:Ethernet HWaddr FE:FF:FF:FF:FF:FF
UP BROADCAST MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:32
RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

It was clear from simple observation that this was incorrect. Our solution to this was to ignore the values returned by libvirt, and build our own mechanism for determining the right values for the network traffic. Under Linux, libvirt creates two special network interfaces for each VEE, one called *vif?.?* and the other called *tap?.?*. We were able to reuse much of the implementation of the probe used for measuring the physical network by pointing it at the *tap?.?* interface in order to gather the actual network traffic data. This alternate approach, avoiding libvirt, is used in our management system.

Summary of Virtual Network Monitoring Evaluation

Our monitoring of virtual execution environments has been successful. By having a separate probe per VEE we observe that separate measurement streams are sent for each of the VEEs on a physical machine. This is particularly useful in situations where the actual location of the virtual host is not important, and also ensures that measurement data from one virtual host is not coupled to a physical machine. The data streams are independent, and so consumers just see streams of measurements from virtual hosts and physical hosts in the same way because the structure of the measurements is identical in both cases.

This approach addresses the *elasticity* and *migration* requirements of the whole monitoring framework. It allows

elasticity as hosts are able to create new VEEs or shutdown VEEs at will, without being concerned about the monitoring. It also allows migration because a VEE can migrate from the hypervisor of one physical host to the hypervisor of another physical host, again, without there being any concern for the monitoring.

In summary, the hypervisor monitor we have designed and built can collect data from the virtual execution environments in a dynamic and adaptable way. It can collect information about CPU, memory, and network usage for each of the VEEs, even though they can be created, executed, and shutdown at run-time. This is important as each VEE represents a virtual resource in a virtual network. Therefore, we can monitor virtual resources displaying *elasticity* and *migration* within a virtual network.

V. CONCLUSIONS

This paper has explored the concept of monitoring virtual resources and its applicability to virtual network management. The design of a new monitoring framework applicable to virtual networks was described. This monitoring framework supports the new management architectural and system model for Future Internet, which is under development in the AutoI [5] project. The AutoI model consists of a number of distributed management systems within the network, called the OSKMV planes: the Virtualisation Plane, the Management Plane, the Knowledge Plane, the Service Enablers Plane, and the Orchestration Plane. Together these distributed systems form a software-driven virtual network control infrastructure that runs on top of all current network and service infrastructures.

The use of Lattice allowed us to build a real management infrastructure that collects, processes, and disseminates network and system information from/to the network entities at real-time, acting as an enabler for self-management functionality. We have seen, from the evaluation on the testbed, that the virtual machine monitoring approach developed is a key component of the information management platform.

We have shown that monitoring is a fundamental feature of any autonomic management system. We have also shown that Lattice can provide such monitoring for virtual networks in which the virtualized resources are highly volatile as they can be started, stopped, or migrated at any time by the management overlay.

The paper has presented an evaluation and a use case of the Lattice framework, which has been successfully used in the AutoI project [5] for monitoring virtual networks. We can build different monitoring systems using the framework, where different implementations of probes, data sources, and consumers are envisaged. Such an approach has been undertaken within the RESERVOIR project [23] [24] to allow the monitoring of service clouds.

As the Lattice framework is like a toolbox for building specific monitoring systems, we intend to issue it as an open source project in its own right to further its use.

A. Further Work

There is much further work to do in the areas of virtual network management and monitoring.

The main area of our future work will be to address the main features for monitoring in a virtualized network environment within the Lattice framework itself. We stated that monitoring needs to take account of scalability, elasticity, migration, adaptability, autonomic, and federation. In this paper we have presented how we deal with elasticity and migration, and in further work we will address the other 4 features.

As we have shown, for full operation of a virtual network the monitoring is a vital part of the control loop. To ensure that this loop is effective we need to be confident that the measurements transmitted by probes are accurate. As such, we intend to investigate and evaluate the accuracy of the monitoring framework.

The mechanism of defining the measurements that probes will send using an information model is aligned with modelling approaches in many other areas. Consequently, we will examine how we can take information models for virtual networks and the network components, and use this to create information models for the probes that will monitor these components. By doing this we could define and create the probes automatically from the virtual network's model.

As we have seen, Lattice can successfully monitor virtual resources. In the future, we wish to adapt Lattice so that it can also monitor virtual resources that are shared amongst various overlays. This will require labeling the resources, and passing this data through the measurements to ensure that the receivers of the measurement data are able to determine which part of the shared resource is being used.

A further goal for the Lattice monitoring system is to have fully dynamic data sources, in which the data source implementation will have the capability for reprogramming the probes on-the-fly. In this way, it will be possible to make probes send new data if it is required. For example, they can send extra attributes as part of the measurement. Yet another useful facility for a data source will be the ability to add new probes to a data source at run-time. By using this approach we will be able to instrument components of the virtual network without having to restart them in order to get this new information.

ACKNOWLEDGMENT

This work is partially supported by the European Union through the Autonomic Internet (AutoI) project [5] and the RESERVOIR project [24] of the 7th Framework Program.

REFERENCES

- [1] "GENI design principles," *Computer*, vol. 39, pp. 102–105, 2006.
- [2] "Future Internet Design (FIND) program," <http://www.nets-find.net/>.
- [3] "Future Internet Assembly (FIA)," <http://www.future-internet.eu/>.
- [4] A. Galis, H. Abramowicz, M. Brunner, D. Raz, P. Chemouil, J. Butler, C. Polychronopoulos, S. Clayman, H. de Meer, T. Coupaye, A. Pras, K. Sabnani, P. Massonet, and S. Naqvi, "Management and service-aware networking architectures for future internet position paper: System functions, capabilities and requirements," in *IEEE 2009 Fourth International Conference on Communications and Networking in China (ChinaCom09)*, August 2009, invited paper.
- [5] "Autonomic Internet (AutoI) Project," <http://www.ist-autoi.eu/>, 2008–2010.
- [6] A. Galis, S. Denazis, A. Bassi, P. Giacomini *et al.*, *Management Architecture and Systems for Future Internet Networks*. IOS Press, <http://www.iospress.nl>, ISBN 978-1-60750-007-0, April 2009.
- [7] L. Mamatas, S. Clayman, M. Charalambides, A. Galis, and G. Pavlou, "Towards an information management overlay for emerging networks," in *NOMS 2010*, 2010.
- [8] X. Dong, S. Hariri, L. Xue, H. Chen *et al.*, "Autonomia: An autonomic computing environment," in *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, 2003, pp. 61–68. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1203684
- [9] L. Cheng, A. Galis, B. Mathieu, K. Jean, R. Ocampo, L. Mamatas, J. Rubio-Loyola, J. Serrat, A. Berl, H. Meer, S. Davy, Z. Movahedi, and L. Lefevre, "Self-organising management overlays for future internet services," in *MACE 2008: Proceedings of the 3rd IEEE international workshop on Modelling Autonomic Communications Environments*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 74–89.
- [10] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 3–10.
- [11] R. A. Gonzalez Prieto, "A-GAP: An adaptive protocol for continuous network monitoring with accuracy objectives," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 4, no. 1, pp. 2–12, June 2007.
- [12] Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *ACM SIGMOD 2003*, June 2003.
- [13] Olston and J. Widom, "Efficient monitoring and querying of distributed, dynamic data via approximate replication," in *IEEE Data Engineering Bulletin*, March 2005.
- [14] A. Deligiannakis, Y. Kotidis, , and N. Roussopoulos, "Hierarchical in-network data aggregation with quality guarantees," in *EDBT 2004*, March 2004.
- [15] J. Strassner, S. V. der Meer, and J. W.-K. Hong, "The applicability of self-awareness for network management operations," in *Fourth IEEE International Workshop MACE 2009*, October 2009. [Online]. Available: <http://www.manweek.org/2009/mace/programme.php>
- [16] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation and experience," *Parallel Computing*, vol. 30, p. 2004, 2003.
- [17] "Nagios," <http://www.nagios.org/>.
- [18] H. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu, "Mon-ALISA : A distributed monitoring service architecture," in *Proceedings of CHEP03, La Jolla, California*, 2003.
- [19] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. L. Rubini, G. Tortone, and M. C. Vistoli, "GridICE: A monitoring service for grid systems," *Future Gener. Comput. Syst.*, vol. 21, no. 4, pp. 559–571, 2005.
- [20] A. Cooke, A. J. G. Gray, L. Ma, W. Nutt *et al.*, "R-GMA: An information integration system for grid monitoring," in *Proceedings of the 11th International Conference on Cooperative Information Systems*, 2003, pp. 462–481.
- [21] R. Srinivasan, "XDR: eXternal Data Representation standard," 1995.
- [22] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 53–65.
- [23] B. Rochwerger, D. Breitgand, E. Levy, A. Galis *et al.*, "The RESERVOIR model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, 2009. [Online]. Available: <http://www.research.ibm.com/journal/rd/534/rochwerger.pdf>
- [24] "RESERVOIR Project," <http://www.reservoir-fp7.eu/>, 2008–2011.
- [25] "libvirt: The virtualization API," <http://libvirt.org/>.
- [26] P. Barham, B. Dragovic, K. Fraser, S. Hand *et al.*, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.