Enabling Performance Intelligence for Application Adaptation in the Future Internet

Prasad Calyam, Munkundan Sridharan, Yingxiao Xu, Kunpeng Zhu, Alex Berryman, Rohit Patali, and Aishwarya Venkataraman

Abstract: **Today's Internet which provides communication channels with best-effort end-to-end performance is rapidly evolving into an autonomic global computing platform. Achieving autonomicity in the Future Internet will require a performance architecture that (a) allows users to request and own 'slices' of geographically-distributed host and network resources, (b) measures and monitors end-to-end host and network status, (c) enables analysis of the measurements within expert systems, and (d) provides performance intelligence in a timely manner for application adaptations to improve performance and scalability. We describe the requirements and design of one such "Future Internet performance architecture" (FIPA), and present our reference implementation of FIPA called 'OnTimeMeasure.' OnTimeMeasure comprises of several measurement-related services that can interact with each other and with existing measurement frameworks to enable performance intelligence. We also explain our OnTimeMeasure deployment in the global environment for network innovations (GENI) infrastructure collaborative research initiative to build a sliceable Future Internet. Further, we present an application–adaptation case study in GENI that uses OnTimeMeasure–enabled performance intelligence in the context of dynamic resource allocation within thin-client based virtual desktop clouds. We show how a virtual desktop cloud provider in the Future Internet can use the performance intelligence to increase cloud scalability, while simultaneously delivering satisfactory user quality-of-experience.**

Index Terms: **Future Internet, performance measurement, virtual desktop clouds.**

I. INTODUCTION

A. Future Internet Overview

Today, the Internet primarily acts as a communication channel between end-systems without any performance guarantees in the quality of end-to-end connectivity provided. This is popularly called as the 'best-effort' service model. The main difficulty in providing quality guarantees arises from the fact that no single service provider or entity controls all the network links and hosts in an end-to-end path. The current Internet is formed by federations of individual networks which peer each others traffic for mutual benefit to extend their network boundaries. Yet, they do not share each other's performance measurements as it may expose bottlenecks that are undesirable when competing for business from the same customers. Although this federated nature has been a barrier to performance transparency and quality guarantees, it is the architectural cornerstone that has enabled the current Internet to scale successfully across the world.

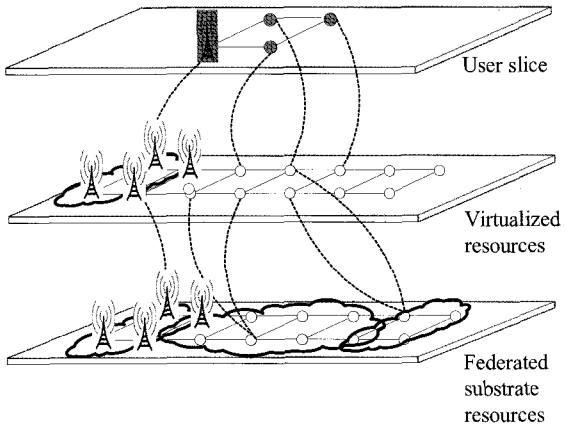


Fig. 1. A user slice in the Future Internet.

Given the demanding nature of emerging multimedia-and-data rich applications that require quality guarantees, slowly but steadily a new Internet–referred to as the Future Internet (FI), from here on–is evolving. Key for the success of the FI is to retain the fundamental strength of the current Internet, i.e., the federated nature of networks, but also provide end-users with better control and performance transparency over the core Internet resources. This evolving FI is being driven primarily by improvements in virtualization [1] technology at end-hosts and core network switches/routers. The global environment for network innovation (GENI) [2] project is the early realization of such an FI vision. GENI is funded through the U.S. National Science Foundation and is a 'sliceable' Internet infrastructure being co-developed by academia and industry. A GENI user as shown in Fig. 1 can request a slice of virtualized resources from both end-host substrates–wired and wireless–and core network for experimentation of an FI application. The goal of GENI is to foster innovations in FI architectures, protocols, and applications through experimentation in both controlled and real-world environments with actual users [3].

B. Autonomic Future Internet Applications

As the Internet evolves, the current general purpose network will need to tightly couple with applications to form a global computing platform. This platform will have customizations to

Manuscript received May 13, 2011.

This material is based upon work supported by the National Science Foundation under award numbers CNS-1050225 and CNS-0940805, VMware, and Dell. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, VMware or Dell.

P. Calyam, M. Sridharan, Y. Xu, K. Zhu, A. Berryman, R. Patali, and A. Venkataraman are with OARnet/Ohio Supercomputer Center, The Ohio State University 1224 Kinnear Road, Columbus, Ohio, USA, email: {pcalyam, Yxu, kzhu}@osc.edu, {sridhara, berryman, rpatali, venkatar}@oar.net.

1229-2370/11/$10.00 © 2011 KICS

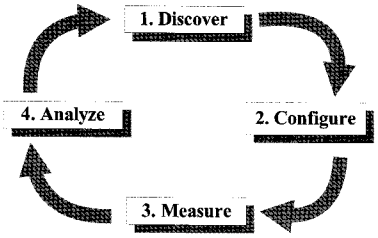


Fig. 2. Autonomic FI application life-cycle.

deliver quality guarantees for different applications, each in their own respective user slices. In addition, every application in this environment needs to be aware of the status of host and network components that support its execution, and should have the ability to quickly adapt to any changes that limit performance and scalability within user slices. The idea of having networks being tightly coupled with applications can already be seen in the current Internet trends in the context of content delivery networks (e.g., Akamai) and private point-to-point links connecting datacenters of major application service providers (e.g., Amazon). In such tightly-coupled platforms, applications and their resources will need to be managed in a distributed manner, the applications need to rapidly-scale with user demand, and the rate-of-failure as well as time-to-repair should be as small that it does not impact user quality-of-experience (QoE). Thus, applications within user slices need the FI to have automated adaptations with autonomicity attributes such as being self-configuring, self-managing, self-monitoring, self-healing and self-optimizing.

A typical autonomic FI application life-cycle is likely to go through four phases in a cyclical manner as shown in Fig. 2:

1. *Discover*: In this phase, the application would identify the host and network resources needed, discover them and reserve them in a geographically-distributed user slice.
2. *Configure*: Next, the application would configure the reser ved application resources meet the quality guarantees expected by the user; it would also configure other instrumentation and measurement resources during this phase at both the host and network.
3. *Measure*: Next, the application would start monitoring the performance of the underlying host and network resources, as well as its own behavior.
4. *Analyze*: Lastly, the application would feed the measurements into expert-systems to derive *performance intelligence*, which can in turn be used to timely trigger reconfiguration of resources or healing of application behavior to improve performance and scalability within a user slice.

Although these phases are functionally separate, they are likely to be automated to operate in a continuous manner during an application's lifetime within a user slice. Nevertheless, some of the tasks in the FI application life-cycle phases will still require human involvement at the application service provider level, particularly for adaptations that may be critical and could affect large numbers of user slices.

C. Future Internet Performance Intelligence

Given that enabling performance intelligence in a timely manner is central to support the autonomic nature of FI applications, measurements that indicate 'host-and-network status awareness' in a user slice need to be carefully instrumented, orchestrated, archived and analyzed. Thus, we can envisage a performance architecture for the FI viz., "Future Internet performance architecture" comprising of several services that interoperate with each other and with existing measurement frameworks such as Cricket [4] and perfSONAR [5]. Together, they will satisfy specific performance intelligence use cases of FI applications both at the user as well as application service provider levels. Future Internet performance architecture (FIPA) should also enable FI applications to use performance intelligence information at the hop, link, path and slice levels for determining actionable instances for adaptation. Further, FIPA should facilitate validation of the adaptation effectiveness with additional performance intelligence in the next life-cycle iteration, and so on. Existing works that share our vision of performance intelligence for management of the FI include [6]–[8].

D. Paper Organization

The remainder of this paper is organized as follows: In Section II, we detail the design motivations and functionalities of our envisioned FIPA services. In Section III, we present our reference implementation of FIPA called 'OnTimeMeasure' [9] in the GENI infrastructure. We describe the various OnTimeMeasure software modules that can be deployed in both centralized and distributed manners. In addition, we describe how the OnTimeMeasure software modules interoperate with other GENI components at the substrate plane, application plane, and measurement plane. Further, we describe how primitives in OnTimeMeasure can be customized to monitor application-specific metrics. In Section IV, we present an application-adaptation case study in a GENI slice. The application-adaption uses OnTimeMeasure-enabled performance intelligence in the context of dynamic resource allocation within thin-client based virtual desktop clouds (VDCs). We show how a virtual desktop cloud provider (CSP) in the FI can use the performance intelligence to increase cloud scalability by supporting a higher number of virtual desktops per datacenter, while simultaneously delivering satisfactory user QoE. Section V concludes the paper.

II. Future Internet PERFORMANCE ARCHITECTURE

A. Design Motivations

The core design motivations of the FIPA can be categorized into two broad perspectives. The first perspective relates to satisfying the use-cases in the FIPA design from the user side as well as from the application service provider side. The second perspective relates to design principles that make the FIPA extensible, fault-tolerant, standards-compliant and secure. In the following, we describe the design motivations for FIPA based on these two perspectives. We remark that the below descriptions are a combination of the authors' original ideas, as well as are based on feedback from group discussions in the GENI community forums [10].

The primary use cases from the user side include: (i) Have I got the system and network resources I asked (i.e., purchased) in my slice? (ii) Is my slice environment functioning to my expectation over my slice lifetime? (iii) Can a problem occurence in my slice environment that impacts my QoE be identified and

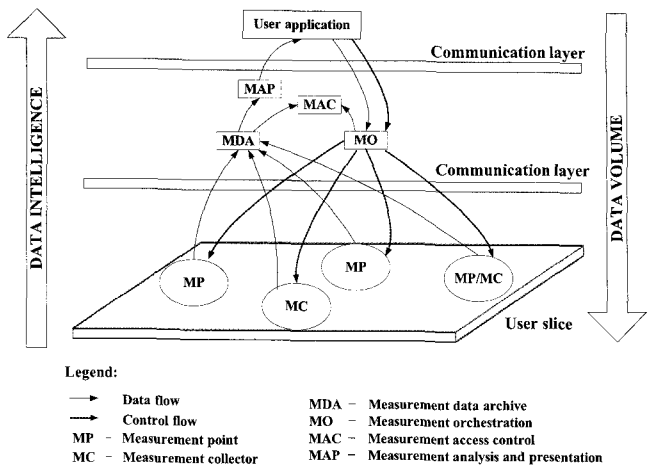


Fig. 3. FI performance architecture.

notified to my application to adapt and heal? (iv) Can problem information also be shared with me and my application service provider if my application cannot automatically heal itself?

Correspondingly, the primary use cases from the application service provider side include: (i) Can I check if the user got the system and network resources he/she asked (i.e., purchased) in user's slice? (ii) Can I monitor all the detailed active (e.g., Ping, traceroute, iperf) and passive (e.g., TCP dump, netflow, router-interface statistics) measurements at end-to-end hop, link, path and slice levels across multiple federated ISP domains? (iii) Can I analyze all the measurements to offline provision adequate resources to deliver satisfactory user QoE, and online i.e., real-time identify anomalous events impacting user QoE?

To make the FIPA extensible, fault-tolerant, standards-compliant and secure, the FIPA can be designed based on principles of service-oriented-architecture. In that vein, the FIPA for extensibility will need to be comprised of a set of specialized services, each with a defined functionality and web service interface specification. This should allow the services to interoperate with each other through communication layers by exchanging *control* messages in order to request, generate, archive, discover, analyze and present *data* from various measurement sources. The architecture is inherently fault-tolerant because of the distributed and modular nature of the design. Entire services or individual modules can be replicated to avoid failure or for load-balancing purposes. In terms of standards-compliance, the potential choices could include simple object access protocol (SOAP)/extensible markup language-remote procedure call (XML-RPC) based messaging services for information exchange between the various services, resource description framework formats adopted in GENI for resource discovery and setup [11], as well as request/response schemas of measurements being standardized in the Open Grid Forum [5]. Further, the security issues of FIPA can be encompassed by existing advances in federated identification such as OpenID and Shibboleth [10].

B. FIPA Services

Fig. 3 shows the FIPA services interacting with each other via communication layers in such a manner that the measurement data volume collected at the slice resources is 'comprehensive' i.e., large in terms of metrics to quantify performance of the end-to-end hop, link, path and slice levels. However, as the performance intelligence is processed and delivered to the application, the measurement data volumes are 'abstract' i.e., small in terms of metrics to guide actionable instances for adaptation. In the following, we briefly describe the composition and functions of each of our envisioned FIPA services:

Measurement point (MP) service: An MP refers to the instrumentation that taps into "measurement data sources" such as network hops and/or system nodes (i.e., passive measurements), or probes end-to-end network paths (i.e., active measurements) to capture and format measurement data. Several MPs are installed within the user slice such that they execute the tap/probe control utilities. MP services provision wired network measurements such as availability, TCP/UDP throughput, available bandwidth, delay, jitter, and loss. In addition, MP services provision wireless network measurements such as utilization, signal strength, and signal-to-interference/noise ratio.

Measurement collection (MC) service: MC is comprised of programmable components that collect, and locally store measurement data using database design principles that allow efficient storage and rapid query capabilities. They can co-reside with MPs on the same host resources or can be installed on separate hosts within an user slice. MC services support measurement data query with pull/push or pub/sub mechanisms using protocols such as simple network management protocol (SNMP), session control protocol (SCP), or HTTP.

Measurement orchestration (MO) service: This service relies on an interpreter to gather measurement requests from authorized users via command-line or graphical user interfaces. It manages various MPs/MCs via schedulers to perform on-going sampling (e.g., periodic, random, stratified random, adaptive) and on-demand sampling of both active and passive measurements. The MO service is aware (through discovery/lookup interfaces) of slice topology, control actors, control actions, data sources, and data types in order to cater measurement requests with any specific timing demands. The MO service further enforces resource policy constraints such as how much bandwidth to use, and which measurements have higher priority when concurrent measurements need to be orchestrated under resource contention scenarios.

Measurement data archive (MDA) service: This service allows MPs and MCs to publish a stable archive of measurement data and meta-data, for purposes such as sharing performance data, analyzing anomaly events, enabling measurement correlations for ground-truth verification (about cause of anomaly event), and making it possible to provide long term access of measurement data for authorized users. It also allows cataloging measurement data indexes for future search and retrieval purposes.

Measurement analysis and presentation (MAP) service: This service is comprised of programmable components that analyze and visualize measurement data in MDA for authorized users. It supports various configurable dashboard tools, and other data charting toolkits. It uses the measurement data to analyze slice-level metrics; for example, a user will be able to check conformance of resource allocation to the resources assigned (i.e., purchased) with slices, and for on-going monitoring of the user

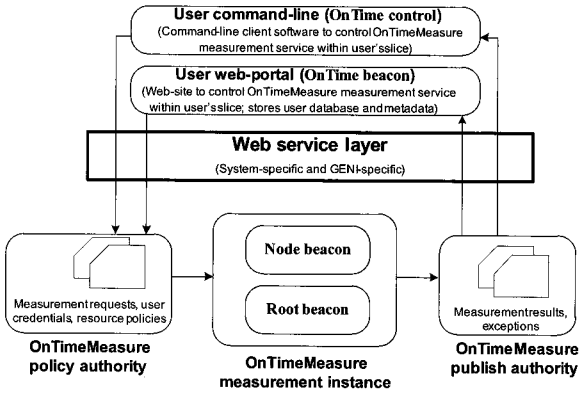


Fig. 4. OnTimeMeasure software modules.

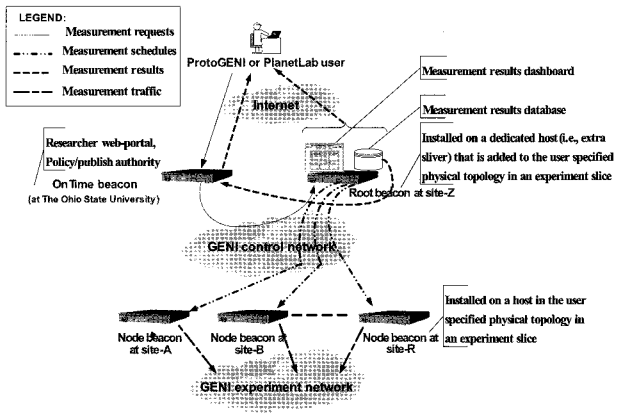


Fig. 5. Centralized OnTimeMeasure deployment.

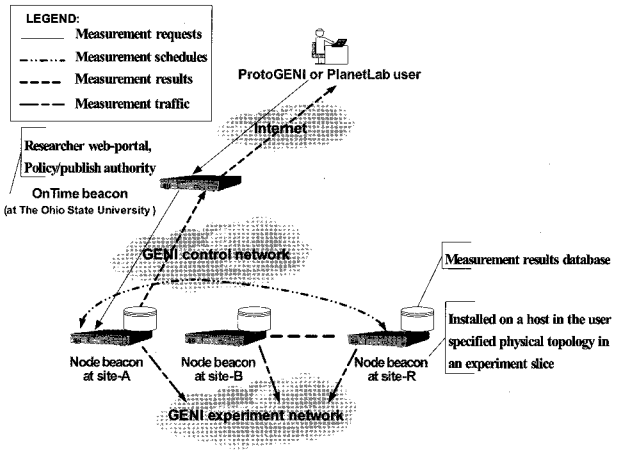


Fig. 6. Distributed OnTimeMeasure deployment.

slice environments. This service enables applications to detect impending or perceived problems for timely adaptation.

Measurement access control (MAC) service: At the heart of every FIMA service are the access control and data privacy mechanisms that are needed to authenticate users or service entities in order to ensure intended use of measurement resources and data collections. This service leverages federated user and policy databases to allow other FIMA services to interoperate securely with each other.

III. OnTimeMeasure: FIPA REFERENCE IMPLEMENTATION

A. Software Modules

Fig. 4 shows the four main software modules in OnTimeMeasure [9] in a GENI slice: (i) Node beacon, (ii) root beacon, (iii) ontime beacon, and (iv) ontime control. The node beacon module supports the MP and MC services. The root beacon supports the MO, MAP, and MDA services. GENI user installs the node and root beacons i.e., an OnTimeMeasure measurement instance in the user's slice based on the MAC services provided by GENI wired and wireless substrate providers (e.g., ProtoGENI [19], PlanetLab [20]). To control the measurement services hosted by the node and root beacons within a GENI user slice, a GENI user can interactively use either the web-interface client viz., ontime beacon, or invoke automated scripts through command-line interface client viz., ontime control. The ontime beacon also has "administrator" privileges, where a substrate provider or measurement service provider or even GENI operations can host the user web-portal, manage measurement service user credentials, discover and manage measurement service instances in user slices, and view meta-data about user-initiated slice measurements.

A "policy authority" is part of the ontime beacon web-portal that determines the list of measurement capabilities that a user is permitted to request in his/her slice. The measurement capabilities permitted are based on pre-defined resource policies in the measurement topology. The policies enforced for measurements scheduling include (a) semantic priorities based on user roles to resolve measurement scheduling conflicts, and (b) measurement level restrictions (e.g., allowable measurement bandwidth and measurement duration) to regulate the amount of network measurement traffic permitted on network paths. Once a measurement request has been cleared by the policy authority, the measurement request will be processed to generate slivers in experiment slices with installs of node and root beacon software. In addition, the user can add/remove a set of measurement tasks for the OnTimeMeasure scheduler to process and generate measurement timetables for node beacons. Further, the user can start, stop and also check status of the various measurement service components. The measurement service components whose status is reported include (a) slice accessibility, (b) root beacon scheduler, (c) node and root beacon communications, (d) measurement data collector, (e) analysis and publish authority, and (f) measurement data visualization. We remark that the "policy authority" can be integrated within any enterprise policy-based management system [29] in order to implement any business policies and procedures to configure/control compute, network and storage resources, and their associated user-application services. Further, the performance intelligence provided by OnTimeMeasure can also be used reversely as stimuli to the policy-based management systems to modify the business policies and procedures. The "publish authority" is also part of the ontime beacon web-portal that provides raw and processed measurement results back to the user. The processed measurement results could correspond to (a) time series of active measurements over a time range, (b) time series of active measurements over a time range with associated network performance anomaly

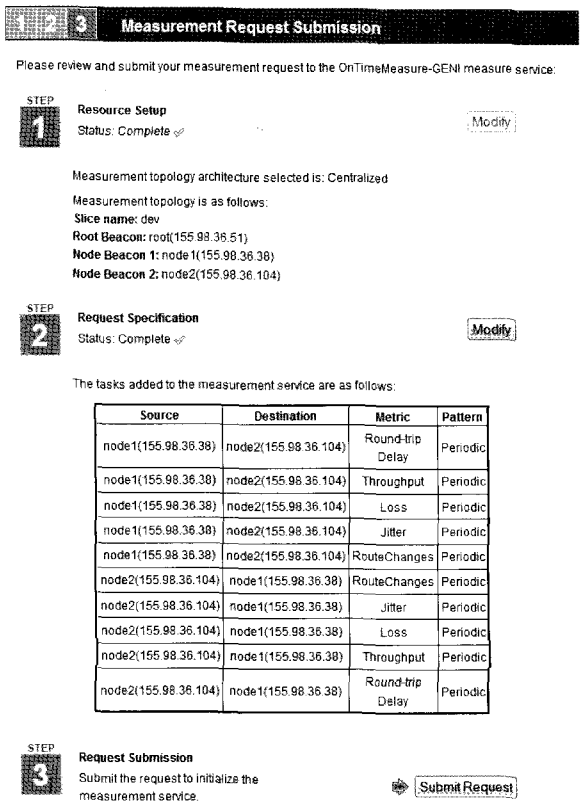


Fig. 7. Centralized measurement request submission.

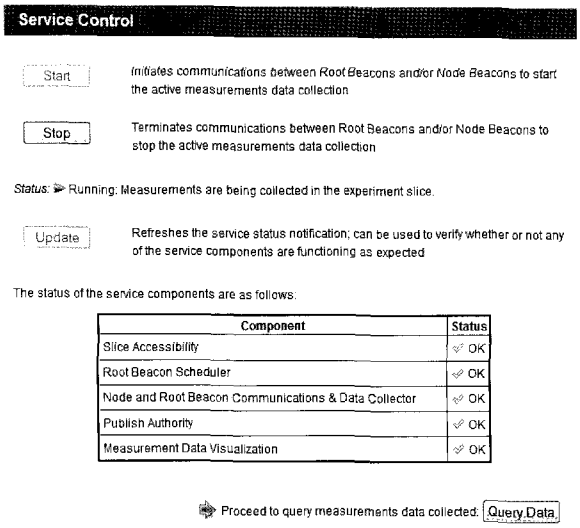


Fig. 8. Service control.

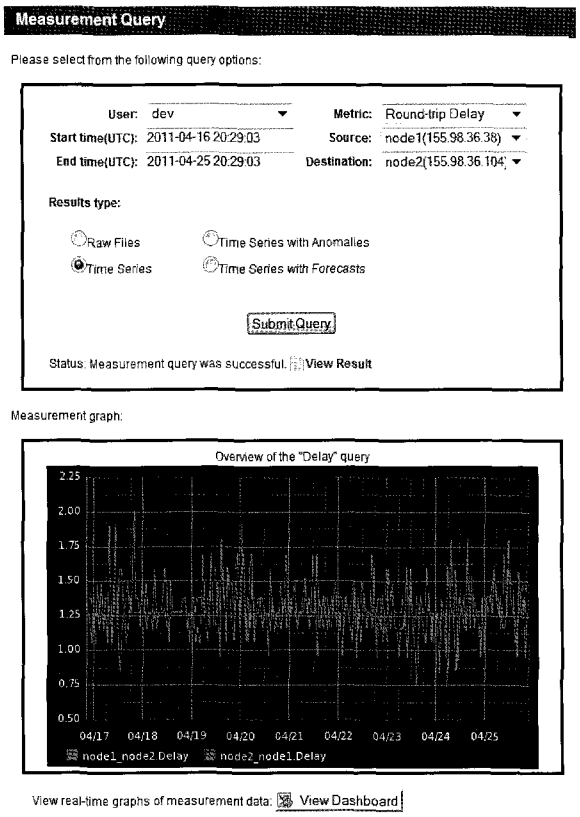


Fig. 9. Centralized measurement query.

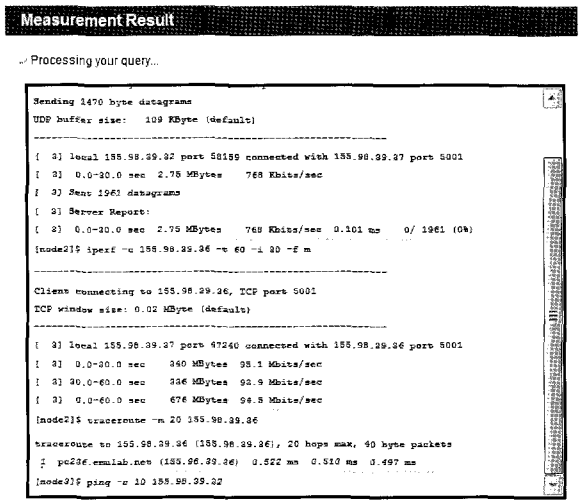


Fig. 10. Distributed measurement result.

events, and (c) time series of active measurements over a time range with associated network weather forecasts.

B. Software Deployment

The OnTimeMeasure software modules can be deployed either using a "centralized orchestration" architecture, or a "distributed orchestration" architecture. The centralized orchestration allows measurement scheduling for continuous monitoring, persistent measurements storage in an user's experiment slice and processed network measurement feeds. Such functionality will generally be useful for "network weathermaps" and long-standing experiments with advanced measurement analysis capabilities. In comparison, the distributed orchestration allows measurement scheduling of on-demand measurement requests without need for persistent measurements storage. Such functionality will generally be useful for users needing one-off or occasional raw measurement tool outputs.

Fig. 5 shows the OnTimeMeasure deployment for centralized orchestration in GENI. We can see that the GENI user interacts with the policy authority and publish authority in GENI using the Internet. The root beacon does not rely on experimental network that carries the FI application traffic to interact with the

node beacons, but instead uses the GENI control network. The reason is that the FI application traffic can become unstable and lead to network partitions since a GENI user sometimes may be exploring with experimental protocols and software. The active measurements collected will be on the experimental network paths to which the node beacons will be connected. Fig. 6 shows the GENI deployment for distributed orchestration. No root beacon will be installed when a user demands a distributed orchestration for the measurement service. The node beacons again do not rely on experimental network to interact with each other for exchanging measurement schedules, but instead use the GENI control network. The measurements at node beacons will be collected on the experimental network paths.

OnTimeMeasure web services allow GENI users to interact with the measurement service functions. Both "system-specific" web-services with our service definitions, and "GENI-specific" web services with GENI compliant schemas i.e., those standardized within the GENI community can be supported. The measurement functions supported by OnTimeMeasure can be broadly grouped under (i) measurement request submission (shown in Fig. 7), (ii) measurement service control (shown in Fig. 8), and (iii) measurement results query (shown in Fig. 9). In the case of centralized orchestration, dashboards are supported, whereas in the case of distributed orchestration, GENI user measurement requests are handled in real-time as shown in Fig. 10.

C. GENI Interoperability

We now describe the ability of OnTimeMeasure to interoperate with other GENI components at the substrate plane, application plane, and measurement plane. OnTimeMeasure has been developed to discover and configure *substrate resources* in the GENI's ProtoGENI [19] and PlanetLab [20]. Both these substrates allow GENI users to create slices that can be registered with OnTimeMeasure, which inturn allows GENI users to control the measurement service functions with ontime beacon or ontime control. In addition, OnTimeMeasure has been integrated with an experimenter workflow tool viz., GENI user shell (Gush) [21] that allows a user to control *experiment applications* within ProtoGENI and PlanetLab via a user-shell commands. Further, OnTimeMeasure has been integrated with several other *measurement frameworks* such as the GENI instrumentation tools (InsTools) [22], as well as the VMware power shell tools (PowerTools) [23]. Each of these tools access several measurement data sources. For e.g., InsTools allows a GENI user to obtain passive packet-captures, SNMP and NetFlow measurements; PowerTools allows a GENI user to monitor virtualized environments in terms of resource consumption at the virtual machine level, as well as the host level. In the GENI measurement plane context, OnTimeMeasure has also been integrated into the digital object repository [24] that allows archiving experiment slice measurement datasets along with meta-data collected by OnTimeMeasure into the GENI measurement data archive. Use cases supported with the digital object repository integration include (a) archive and share subsets of experiment results, (b) archive and share entire experiment slice measurement results, and (c) backup/restore entire experiment slice measurement results.

Given a FI possiblity where OnTimeMeasure interoperates with several measurement frameworks, there will arise a situation where there will be large number of measurement requests that need to be orchestrated in FI slices. The measurement orchestration needs to be such that it does not affect application (e.g., CPU cycles needed for monitoring of a resource's performance should not impact the CPU cycles needed for actual application) and also measurement collection should not produce conflicts (e.g., two Iperf tools invoked at the same time along common hosts or network paths) [25]. Orchestration should also consider minimizing "cycle time" so that more frequent real-time network status updates can be provisioned to application. Cycle time is defined as the time taken to complete one round of measurements execution over a measurement topology. Smaller the cycle time, larger the number of measurement rounds, and thus more detailed network status can be sampled. To reduce the cycle time, it is essential to reduce the idle periods in measurement schedules for each cycle time, while still not affecting the application and avoiding measurement conflicts. However, existing works [25]–[27] use "open loop" principles for scheduling measurements that conservatively guess measurement execution times. This conservative guessing results in many idle periods in measurement execution schedules. Consequently, existing open loop methods do not scale to accommodate FI measurement loads.

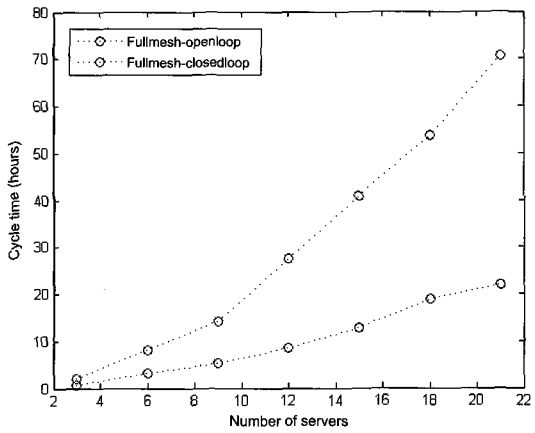


Fig. 11. Full mesh measurement topology comparison.

OnTimeMeasure's web-services architecture easily allows measurement tools at MPs to provide feedback i.e., in a "closed loop" manner to the MO service when their measurement execution is completed. Leveraging this intelligence, MO could schedule other queued measurements on MPs in the idle periods that do not affect the application or cause conflicts with other scheduled measurements. Figs. 11 and 12 show how OnTimeMeasure handles full-mesh, tree and hybrid measurement topologies for increasing number of measurement requests at MPs, and produces significant improvements in cycle times. The results were obtained by running an instance of OnTimeMeasure in a GENI slice comprising of 21 servers in ProtoGENI. The full-mesh corresponds to the case where measurements are collected between all the servers over the entire measurement topology. The tree case is where measurements are collected only between neighboring servers in the measurement topology. Lastly,

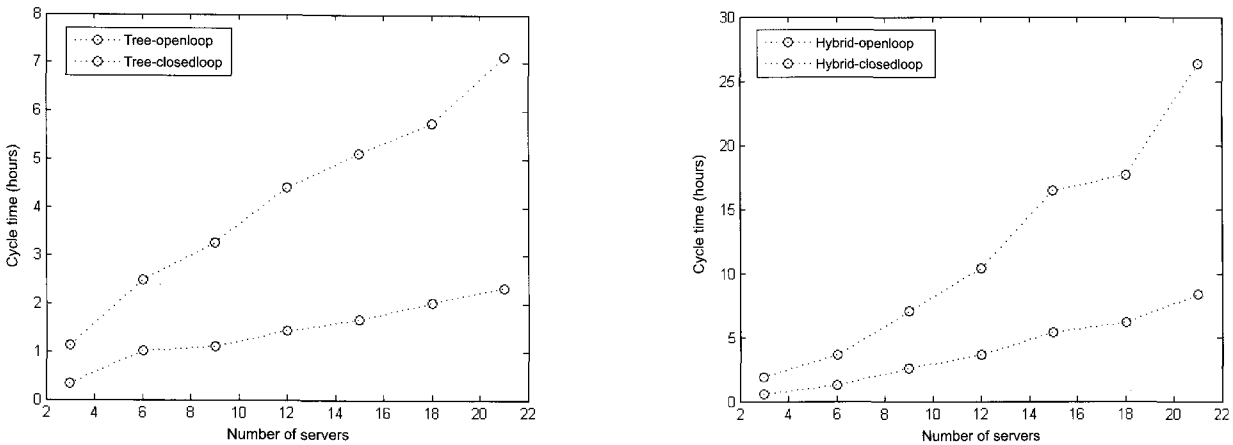


Fig. 12. Comparison of open loop and closed loop measurement orchestration: (a) Tree measurement topology and (b) hybrid measurement topology.

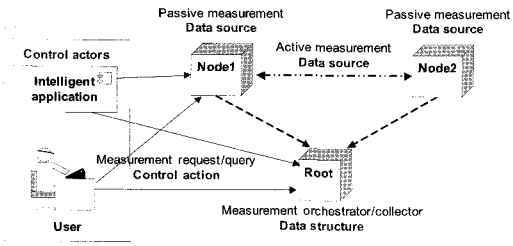


Fig. 13. OnTimeMeasure custom metric integration feature primitives.

the hybrid case is where measurements are collected in a full-mesh manner for a subset of servers in the measurement topology, and tree manner for the other servers that are not part of the full-mesh subset. In all these 3 cases, we can clearly see that OnTimeMeasure's closed loop implementation can be leveraged to notably improve cycle times in MO in comparison to existing open loop implementations, and thus OnTimeMeasure is suited for handling large number of measurement requests that will need to be orchestrated in FI slices.

D. GENI Customization

We now describe how OnTimeMeasure can be customized via web services to work with any application-specific metrics using a feature called "*custom metric integration*." This feature is similar to a capability in the Amazon cloudwatch measurement service [30], where customers of Amazon web services can integrate their custom application metrics into their monitoring activities using web services. Such a customization is a key requirement since each GENI experiment is unique; often, users want to not only use off-the-shelf tools (e.g., ping, netflow) within measurement services in GENI, but also other tools they have developed or others have developed that are relevant to their experiment. Fig. 13 shows the primitives (*control actors*, *data sources*, *data structures*, *control actions*) that need to be configured by the users within OnTimeMeasure in the custom metric integration process.

To better understand the primitives, let us consider a case study of dynamic resource allocation within thin-client based virtual desktop clouds (VDCs), where allocations are based on OnTimeMeasure' feedback of network health from the client, and load on the server (for details, please refer to Section IV). The specification of primitives in the VDC experiment can be illustrated as follows:

1. *Control actors*: Actors refer to VDC experiment users that access the application data or share the application data with other actors.
2. *Data sources*: These are resource monitors or data generation tools deployed in node beacons within an experiment slice; the tools communicate with each other to perform active measurements or collect passive measurements within a host in an on-going or on-demand basis. A specific data source example is PowerTools that monitors memory, CPU, and network health of each host.
3. *Data structures*: These refers to the sets of measurement data types that would be stored in the root beacon database. Examples of data types can be 'host: `string`', 'resource pool: `string`', 'sample time: `timestamp`', 'CPU load: `float`'.
4. *Control actions*: They relate to the actor controls such as start/stop times of the data generation tools, or sampling frequency. Example of a control action for sampling can be to configure PowerTools to query memory usage every 10 seconds.

IV. FIPA CASE STUDY: DYNAMIC RESOURCE ALLOCATION IN VIRTUAL DESKTOP CLOUDS

We now present an application-adaptation case study in a GENI slice that is based upon a demonstration we presented during the 10th GENI engineering conference (GEC10) [2]. The demonstration uses OnTimeMeasure-enabled performance intelligence in the context of dynamic resource allocation within thin-client based virtual desktop clouds (VDCs). In the following, we first provide an overview of the resource allocation problem. Next, we describe application-adaptation that uses OnTimeMeasure-enabled performance intelligence information within a VDC. Lastly, we present our GENI slice setup for our VDC experiments and the obtained experiment results.

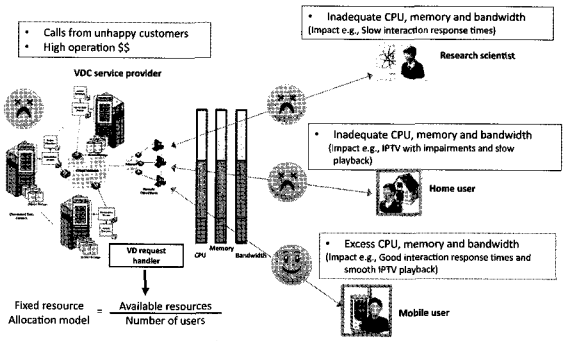


Fig. 14. Illustration of F-RAM limitations in virtual desktop clouds.

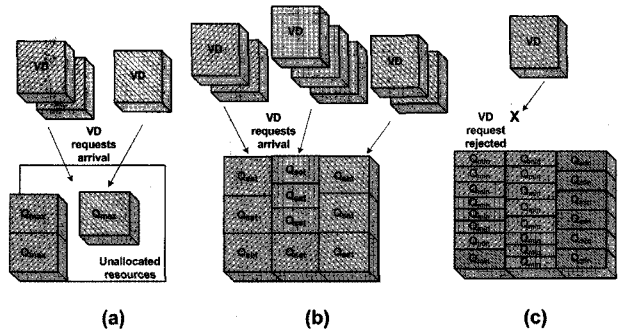


Fig. 15. U-RAM allocations when there are (a) new VD requests handling with freely available resources, (b) new VD requests handling with all available resources allocated, and (c) new VD request rejected when SLA violation situation occurs.

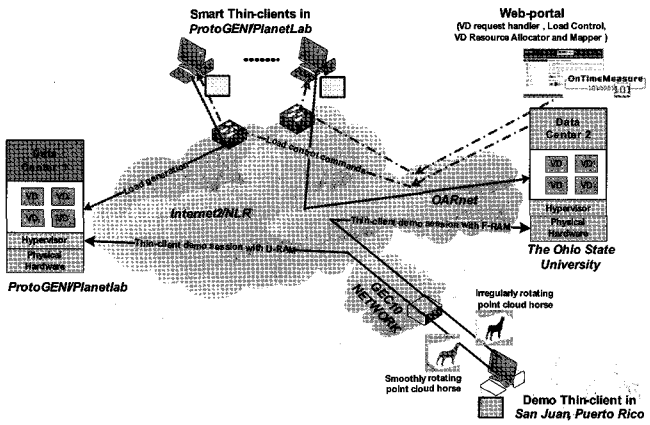


Fig. 16. Demonstration of virtual desktop cloud with U-RAM in GENI.

A. The Resource Allocation Problem

We consider a futuristic scenario, where users at home or at enterprises subscribe for virtual desktops (VDs) with a VDC service provider (CSP), who ships to subscribers set-top boxes that function as thin-clients. The drivers for users to transition from traditional desktops to VDCs are obvious (i) desktop support in terms of operating system, application and security upgrades will be easier to manage, (ii) the number of underutilized distributed desktops unnecessarily consuming power will be reduced, and (iii) mobile users will have wider access to their applications and data.

Currently, CSPs allocate resources to VD requests based primarily on CPU and memory measurements [12]–[15]. There is surprisingly sparse work [16], [17] on resource adaptation coupled with measurement of network health and user quality of experience (QoE). It is self-evident that any cloud platform's capability to support large user workloads is a function of both the server-side desktop performance as well as the remote user-perceived QoE. In other words, a CSP can provision adequate CPU and memory resources to a VD in the cloud, but if the thin-client protocol configuration does not account for network health degradations and application context, the VD will be unusable for the user. Also, the CSP has to allocate resources to individual virtual desktops (VDs) in a way that decreases the resource consumption cost, while at the same time maximizes the user QoE. Hence, lack of proper performance intelligence in terms of "network-and-human awareness" in cloud platforms inevitably results in costly guesswork and over-provisioning while managing physical device and human resources, which consequently annoys users due to high service cost and unreliable quality of experience.

Fig. 14 illustrates the above drawback of current resource allocation schemes in a typical VDC provider. We can see that the fixed resource allocation models (F-RAM) used by CSPs results in situations where a Mobile user is provisioned excess resources that results in satisfactory QoE (Q_{excess}), whereas a Scientist or Home user are provisioned with inadequate resources, which results in unsatisfactory QoE ($< Q_{\min}$), and the CSP incurs a high cost of operation. This problem is compounded by the fact that users use a wide range of applications (e.g., Web browsing, video playback) with different resource consumption profiles, and connect to data centers within a VDC over network paths with different network health conditions.

B. Application Adaptation

To overcome this problem, we compare F-RAM with a utility-directed[1] resource allocation model (U-RAM) that builds upon earlier works such as QoS-based resource allocation model (Q-RAM), originally proposed in [28]. The U-RAM adapts the resource (i.e., CPU, memory and network bandwidth) allocation dynamically based on the real-time performance intelligence feedback from system, network and, user QoE measurements that can be collected using OnTimeMeasure.

The fundamentals that guide our U-RAM solution can be illustrated by the example shown in Fig. 15 involving 3 desktop pools being provisioned at a data center site within a VDC. We can see that U-RAM allocates resources to a new VD request such that it results in utility saturation quality ($Q_{\max}$) when there are freely available resources. When all available resources are allocated under light loads, U-RAM allocates resources to new VD requests such that all the VDs operate below $Q_{\max}$ but above $Q_{\min}$ i.e., Q_{set} range. Lastly, when all available resources are allocated under heavy loads, U-RAM allocates $Q_{\min}$ resources to both the already allocated VDs as well as in the case of the new VD. Once it is determined that provisioning a new VD request will cause violation of user service level agreement

[1]A utility function indicates how much of application performance can be increased with larger resource allocation. Beyond a certain point, application utility saturates and any additional resource allocation fails to further increase application performance.

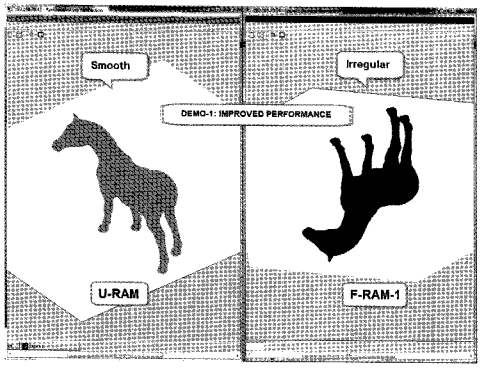


Fig. 17. U-RAM versus F-RAM-1 to show improved performance.

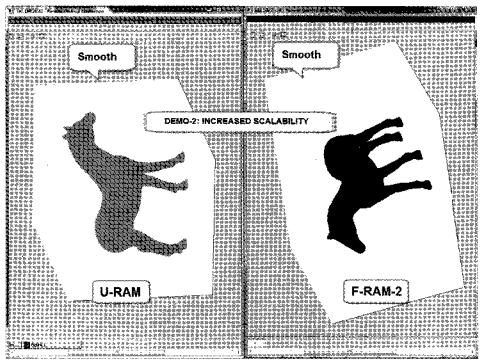


Fig. 18. U-RAM versus F-RAM-2 to show increased scalability.

(i.e, resource allocation causes quality to drop below $Q_{\min}$), all new VD requests arriving at the data center thereafter are either rejected or directed to alternate data center resource sites with the VDC. The key to improved performance in U-RAM is the performance intelligence feedback from OnTimeMeasure that enables U-RAM to continously *discover* the required resources in a VDC, *configure* desktop pools to suit different user application groups, *measure* and *analyze* their performance in order to re-discover and re-configure resources, and so on–to improve the overall cloud scalability, while delivering satisfactory user QoE.

C. Application Setup in GENI

The experiment setup on GENI is shown in Fig. 16. We created two GENI slices for the experiment, one slice for the VDC data center environment and the other slice for distributed thin-client user sites. The 'data center' slice consisted of two data centers, one at The Ohio State University and the other at The University of Utah. The data center at Utah ran the U-RAM algorithm and the data center at Ohio ran the F-RAM algorithm. A number of thin-clients were launched from geographically distributed locations within the 'user slice' to make VD requests to either data center for load generation. The 'user slice' included a demo site at the conference venue in Puerto Rico which initiated two separate thin-client connections to each data center. These two demo thin-clients were running identical Matlab-based point-cloud animations of a horse (but with different colors; green for U-RAM and blue for F-RAM), to compare their application performance. Node beacon instances (integrated with PowerTools) of the OnTimeMeasure framework were deployed at all the thin-client sites in the 'user slice' to measure the network and system performance, and a root beacon instance of the OnTimeMeasure framework was deployed at the Ohio data center to collect the node beacon measurements.

D. Application Adaptation Results from GENI

The collected measurements in the root beacon were used by the U-RAM algorithm to make dynamic resource allocation decisions. We illustrate this decision making using two scenarios we demonstrated at GEC10, one where the U-RAM delivers "improved performance" than F-RAM, and in the other where U-RAM achieves "increased scalability" (i.e., more number of VDs are handled) while guaranteeing the same level of user QoE performance compared to F-RAM.

The performance comparison results between U-RAM and F-RAM at GEC10 are summarized in Table 1. Each of the two data centers we deployed allocates 8 GHz of total CPU resources and each user needs at least 800 MHz allocation for satisfactory user QoE performance; we characterized satisfactory user QoE performance as the "smooth" rotation of the point-cloud horse animation in Matlab, and unsatisfactory user QoE performance as the "irregular" rotation. The U-RAM algorithm with such performance intelligence functions as follows: Consider a case when there are 12 user VD connection requests, it will accept 10 of them and rejects 2 requests. The 10 users will be allocated 800 MHz each and all of them will deliver satisfactory user QoE performance. U-RAM's rejection of 2 VD connections is justified, since these 2 connections will anyway not deliver satisfied user QoE performance even if they were allocated resources and will actually affect the QoE performance of the already allocated VDs. On the other hand, for the same case, two different F-RAM allocations are possible, one in which F-RAM under-allocates defined as F-RAM-1 in Table 1 i.e., 500 MHz per VD, and the other in which F-RAM over-allocates defined as F-RAM-2 in Table 1 i.e., 1.4 GHz per VD. In the under-allocation possibility, F-RAM-1 accepts all 12 connections, but none of them deliver satisfactory user QoE since all of them are not allocated adequate resources, while 2 GHz CPU capacity is still available for allocation. This results in an irregular point-cloud animation rotation as illustrated in Fig. 17. In the over-allocation possibility, F-RAM-2 delivers satisfactory user QoE performance for only 5 VD users, while rejecting 7 connections. U-RAM compared to F-RAM-2 supports more VDs while guaranteeing the same level of user QoE performance, and hence achieves better scalability, as illustrated in Fig. 18. Thus, the OnTimeMeasure framework enables performance intelligence in novel adaptive applications such as dynamic resource allocation in VDCs in the FI.

V. CONCLUSIONS AND OUTLOOK

The FI, that is slowly but steadily evolving, is likely to be application centric, where the network and application are tightly coupled to deliver improved performance. Application service providers will own and operate 'slices' on the FI, which can be configured to suit user needs at Internet-scale. In this paper, we envisioned a FIPA that is extensible, fault-tolerant, standards-compliant, and secure. In addition, we described our reference

Algorithm	#VDs Requested/allowed	Total /minimum (CPU)	Allocation
U-RAM	12/10	8 GHz /800 MHz	800 MHz
F-RAM-1	12/12	8 GHz /800 MHz	500 MHz
F-RAM-2	12/5	8 GHz /800 MHz	1.4 GHz

Table 1. Comparison of VD connections handled by U-RAM and F-RAM in GENI demonstration.

implementation of FIPA called 'OnTimeMeasure' that supports services to 'measure' the performance within a user slice, 'analyze' and derive 'intelligence' that can be used in timely manner for application adaptations to improve performance and scalability.

We evaluated the effectiveness of OnTimeMeasure in the GENI infrastructure, which is an FI platform being developed by academia and industry partnerships. We demonstrated the effectiveness of closed-loop orchestration used in OnTimeMeasure, which is critical for interoperability with other existing measurement services, particularly when large number of application specific measurements need to be orchestrated. Lastly, we used OnTimeMeasure-enabled performance intelligence to compare utility-driven resource allocation schemes in virtual desktop clouds. Our results from the GENI infrastructure experiments demonstrated how performance intelligence enables autonomic nature of FI applications to mitigate the costly resource overprovisioning and user QoE guesswork, which are common in the current Internet.

REFERENCES

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *in Proc. HotNets*, pp. 1–7, Nov. 2004.

[2] C. Elliott, "GENI: Global environment for network innovation," in *Proc. IEEE LCN*, pp. 8, Oct. 2008.

[3] A. Feldmann, "Internet clean-slate design: what and why?," *ACM SIGCOMM Comput. Commun. Review*, vol. 37, no. 3, pp. 59–64, July 2007.

[4] J. Allen, "Driving by the rear-view mirror: Managing a network with cricket," *in Proc. USENIX Netw. Administration Conf.*, Apr. 1999.

[5] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, M. Swany, S. Trocha, and J. Zurawski, "PerfSONAR: A service oriented architecture for multi-domain network Monitoring," *in Proc. ICSOC, Springer Verlag, LNCS 3826*, pp. 241–254, 2005.

[6] C. Mingardi, G. Nunzi, D. Dudkowski, and M. Brunner, "Event handling in clean-slate Future Internet management," *in Proc. IEEE IFIP IM*, 2009.

[7] S. Kim, M. Choi, H. Ju, M. Ejiri, and J. Hong, "Towards management requirements of Future Internet," *Challenges for Next Generation Network Operations Service Manag.*, Springer LNCS, vol. 5297, pp. 156–166, 2008.

[8] L. Mamatas, S. Clayman, M. Charalambides, A. Galis, and G. Pavlou, "Towards an information management overlay for the Future Internet," *in Proc. IEEE IFIP NOMS*, 2010.

[9] P. Calyam, K. Zhu, Y. Xu, and A. Berryman. (2010, Nov.). OnTimeMeasure: Centralized and distributed measurement orchestration software. [Online]. Available: http://groups.geni.net/geni/wiki/OnTimeMeasure

[10] GENI instrumentation and measurement architecture community resource. [Online]. Available: http://groups.geni.net/geni/wiki/GeniInstrumentationandMeasurementsArchitecture

[11] I. Baldine, Y. Xin, M. Anirban, C. Heermann, J. Chase, V. Marupadi, A. Yumerefendi, and D. Irwin, "Networked cloud orchestration: A GENI Perspective," *IEEE MENS*, 2010.

[12] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper, "Adaptive quality of service management for enterprise services," *ACM Transactions on the Web*, vol. 2, no. 8, 2008, pp. 1–46.

[13] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," *in Proc. ACM SIGOPS/EuroSys*, 2007.

[14] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Agile dynamic provisioning of multi-tier internet applications," *ACM Trans. on Autonomous and Adaptive Syst.*, vol. 3, no. 1, 2008, pp. 1–39.

[15] H. Van, F. Tran, and J. Menaud, "Autonomic virtual resource management for service hosting platforms," *in Proc. ICSE-CLOUD*, 2009.

[16] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration," *in Proc. VTDC*, 2006.

[17] A. Berryman, P. Calyam, A. Lai, and M. Honigford, "VDBench: A benchmarking toolkit for thin-client based virtual desktop environments," *in Proc. IEEE CLOUDCOM*, 2010.

[18] N. Agoulmine, S. Balasubramaniam, D, Botvitch, J. Strassner, E. Lehtihet, and W. Donnelly, "Challenges for autonomic network management," *in Proc. MACE*, 2006.

[19] ProtoGENI: A GENI wired and wireless substrate. [Online]. Available: http://groups.geni.net/geni/wiki/ProtoGENI

[20] PlanetLab: A GENI wired substrate. [Online]. Available: http://groups.geni.net/geni/wiki/PlanetLab

[21] Gush: GENI user shell. [Online]. Available: http://groups.geni.net/geni/wiki/GushProto

[22] Intrumentation tools: A GENI instrumentation and measurement service. [Online]. Available: http://groups.geni.net/geni/wiki/InstrumentationTools

[23] VMware power tools: Virtual infrastructure administration scripts. [Online]. Available: http://www.vmware.com

[24] Digital object repository. (2011, Aug.). A GENI measurement data arcive service. [Online]. Available: http://groups.geni.net/geni/wiki/DigitalObjectRegistry

[25] P. Calyam, C.-G. Lee, E. Ekici, M. Haffner, and N. Howes, "Orchestrating of network-wide active measurements for supporting distributed computing applications," *IEEE Trans. Comput.*, vol. 56, no. 12, pp. 1629–1642, Dec. 2007.

[26] E. Blanton, S. Fahmy, and S. Banerjee, "Resource management in an active measurement service," *in Proc. IEEE Global Internet Symposium*, 2008.

[27] Z. Qin, R. Rojas-Cessa, and N. Ansari, "Task-execution scheduling schemes for network measurement and monitoring," *Elsevier Comput. Commun.*, vol. 33, no. 2, pp. 124–135, 2010.

[28] R. Rajkumar, C. Lee, J. Lehoczky, and D. Slewlorek, "A resource allocation model for QoS management," *in Proc. IEEE RTSS*, 1997.

[29] J. Strassner, "Policy-based network management: Solutions for the next generation," *Morgan Kaufmann Series in Networking, ISBN: 1-55860-859-1*, 2004.

[30] Amazon cloudWatch: Monitoring framework for amazon web services cloud resources and applications. Amazon. [Online]. Available: http://aws.amazon.com/cloudwatch/

Prasad Calyam received the B.S. degree in Electrical and Electronics Engineering from Bangalore University, Indea, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from The Ohio State University, in 1999, 2002, and 2007 respectively. He is presently a Senior Research Engineer at the Ohio Supercomputer Center/OARnet, The Ohio State University. His current research interests include multimedia networking, cyber security, cyberinfrastructure systems, and network management.

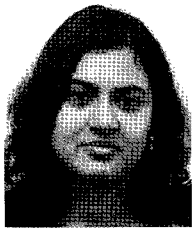
Mukundan Sridharan received the B.S. degree in Electronics and Communication Engineering from Madras University, India, and the M.S. and Ph.D. degrees in Computer Science and Engineering from The Ohio State University, in 2000, 2002, and 2011, respectively. He is currently a Postdoctoral Fellow at OARnet, The Ohio State University. His current research interests include wireless sensor networks, multimedia networking, and distributed systems.

Yingxiao Xu received the B.S. and M.S. degrees in Mechanical Engineering from Southeast University, China, and the Ph.D. degree in Computer Science and Engineering from Fudan University, in 1993, 1996, and 2002, respectively. He is currently a Visiting Scholar at the Ohio Supercomputer Center, The Ohio State University. His current research interests include software re-use, social computing, and computer and network management.

Kunpeng Zhu received the B.S. degree from in Computer Science and Engineering at Yanshan University, China in 2009 and is currently pursuing his M.S. degree in Geodetic Science and Surveying at The Ohio State University, and is a Graduate Research Associate at the Ohio Supercomputer Center. His current research interests include re-usable software engineering, network performance measurements, and scalable web-services.

Alex Berryman is currently pursuing his B.S. degree in Aeronautical and Astronautical Engineering at The Ohio State University, and is a National Science Foundation REU Student at the Ohio Supercomputer Center/OARnet. His current research interests include computer and network virtualization, network performance measurements, and distributed systems.

Rohit Patali received the B.S. degree in Computer Engineering from University of Mumbai, India in 2009. He is currently pursuing his M.S. degree in Computer Science and Engineering at The Ohio State University, and is a Graduate Fellow at OARnet. His current research interests include multimedia networking and distributed systems.



Aishwarya Venkataraman received the B.S. degree in Computer Science from Birla Institute of Technology and Science, Pilani, India in 2008. She is currently pursuing her M.S. degree in Computer Science and Engineering at The Ohio State University, and is a Graduate Fellow at OARnet. Her current research interests include thin-client cloud computing, and cloud security.