Hybrid Testbeds for QoS Management in Opaque MANETS

Pratik K. Biswas, Alex Poylisher, Ritu Chadha, Abhrajit Ghosh

Applied Research, Telcordia Technologies Inc.
One Telcordia Drive, Piscataway, NJ 08854
{pbiswas, sher, chadha, aghosh}@research.telcordia.com

***Abstract*— The design of QoS mechanisms for wireless networks in general, and mobile ad hoc networks (MANETs) in particular, is a challenging task. The challenges are further compounded when the characteristics of the intermediate network segments are not observable from the originating segment, and as a consequence these segments have to be treated as *opaque* networks. End-to-end QoS assurance for such opaque networks, consisting of admission control and quality adjustment, can be based on techniques for dynamically measuring throughput representing the state of these networks. *Testing* these QoS mechanisms poses a special technical challenge due to the difficulty of conducting experiments in a MANET environment at a scale larger than a dozen nodes or so. In this paper, we describe a distributed and hybrid testbed that has been deployed for running large-scale experiments to demonstrate the efficacy of a measurement-based QoS solution. The infrastructure for the testbed provides an integrated platform consisting of *real nodes running the actual software under test,* augmented with a *simulated network* environment. We define a set of metrics and run experiments to evaluate the effectiveness of the QoS solution as well the performance of the deployed testbed. We propose an alternate architecture that employs a *Xen-based virtualization* of the real nodes from the deployed testbed. We compare the performances of the *virtualized architecture* with the *non-virtualized one* vis-à-vis latency and resource utilization. Our goal is to establish benchmarks for running large-scale experiments on performance and QoS measurements in virtualized environments.**

Index Terms*— *Mobile ad hoc network (MANET), end-to-end (e2e), multi-level security (MLS), quality of service (QoS), measurement-based admission control (MBAC), dynamic throughput graph (DTG), admission control functionality (ACF), quality adjustment functionality (QAF), software-in-the-loop (SITL), virtualization

I. INTRODUCTION

Providing end-to-end (e2e) quality of service (QoS) assurances for flows that span multiple heterogeneous networks is a challenging problem. The challenges are further compounded when the characteristics of the intermediate heterogeneous network segments are not observable from the originating or terminating network segments. In military mobile ad hoc networks (MANETs), this limitation arises due to multi-level security (MLS) requirements that restrict the visibility of the encrypted wireless ("black") network from the unencrypted ("red") network which hosts the management applications. We will henceforth refer to such segments as opaque. However, knowledge of the *opaque network* characteristics (e.g. intermediate link delays, losses) is critical to providing the e2e QoS assurances, particularly for delay and loss sensitive mission-critical applications.

End-to-end (e2e) QoS assurances for such bandwidth-sensitive heterogeneous networks, where there is very little *a priori* knowledge of the intervening opaque networks, can be based on dynamically measuring throughputs. A measure of throughput of the transmitted traffic between the end segments and across the opaque wireless segment can be used to represent the *state* of such a heterogeneous network. This paper presents a *hybrid testbed* that has been deployed to test the effectiveness of such a *dynamic-throughput-measurement*-based QoS mechanism. The testbed integrates simulated IP stacks from the simulator with real instances of the QoS measurement algorithms, running on real nodes. Data on flow information is fed from the simulator to the real world where it is analyzed through the *dynamic-throughput-measurement*-based algorithm to arrive at QoS decisions. These decisions are then fed back to the simulator where they are implemented to verify the efficacy of the QoS algorithm. This paper also proposes an alternate architecture for this hybrid testbed, where the real nodes are all run on *virtual machines*.

The remainder of the paper is organized as follows: Section II discusses related work. Section III explains the problem background and relevant technologies. Section IV describes the network model and the implemented QoS solution. Section V presents the architecture of the deployed testbed and defines a set of metrics for evaluating the QoS mechanism as well as the performance of the testbed. Section VI proposes an alternate architecture that deploys the real nodes on virtual machines and compares the performance of the virtualized testbed against the non-virtualized one. Section VII concludes the paper with an overview of our future plans.

II. RELATED WORK

Related work can be grouped into three main categories: (1) QoS mechanisms for unicast application traffic, (2) virtualization, and (3) hybrid testbeds for wireless networks.

The literature on *QoS in communications* goes back a long way. However, we restrict our attention to the work that is most closely related to *opaque networks* and *measurement-based*

Conference WICON '08, November 17–19, 2008, Maui, Hawaii, USA.

admission control (MBAC) in particular. In MBAC, admission control schemes use measurements to characterize current network load. In [1], *time-delay* measurements are used to describe the characteristics of wireline opaque networks for a single traffic class. [2, 3, 4] consider various MBAC approaches that assume complete knowledge of and control over the elements in the path of the data packets. Further, the authors of [4] have shown that different MBAC algorithms all achieve almost identical levels of performance. In this work, we have used the *dynamic throughput graph (DTG)*-based technique [5] for analyzing the current state of intermediate wireless opaque networks (of mixed networks) and then using that knowledge to provide e2e QoS for unicast application traffic. The solution uses DiffServ [6] and the Bandwidth Broker concept [7] as a basis; applications request admission into the network and a Bandwidth Broker decides whether or not to grant the request. Admitted flows are marked with appropriate DSCPs [8] and are thereby accorded differentiated treatment in the network by using appropriate queuing schemes.

Virtualization has been applied to operating systems both commercially and in research for nearly thirty years. IBM VM/370 [9] made use of virtualization to allow binary support for legacy code. VMware [10] family of products, such as, VMware Workstation, VMware GSX Server, VMware ESX Server, etc., virtualizes commodity hardware, allowing multiple operating systems to run on a single host. All of these examples implement a *full virtualization* of (or at least a subset of) the underlying hardware, rather than paravirtualizing and presenting a modified interface to the guest operating system. LPARs [11] and Denali [12], on the other hand, use the *paravirtualization* approach to build an infrastructure for distributed systems. Xen [13] is a high performance resource-managed virtual machine monitor (VMM) (or Hypervisor) which provides an idealized virtual machine abstraction, thereby allowing multiple commodity operating systems to share conventional hardware in a safe and resource-managed fashion, but without sacrificing either performance or functionality. Xen can use either paravirtualization or hardware-assisted full virtualization.

Hybrid infrastructures [14 - 22] have proved to be useful in validating networking techniques and conducting large scale experiments for wired/wireless environments. Emulab/Netbed (a descendant of Emulab) [14, 15, 16] is a network testbed that integrates emulators, simulators and live networks into a common framework under a common user interface. Experiments on Emulab/Netbed testbed [15] can combine real elements with simulated elements, each modeling different portions of a network topology in the same experimental run. The testbed can be used for straightforward comparisons of simulated, emulated, and wide-area scenarios. The EXperimental Testbed for Research Enabling Mobility Enhancements (EXTREME) [17] provides an experimentation facility that supports the testing of networking algorithms and technologies for wireless environments in a close-to-real scenario. Its design has been inspired by Emulab. EMWIN/EMPOWER [18, 19] is an IP-based scalable framework for mobile wireless emulation. With EMWIN, the mobility of the target mobile wireless network with a number of mobile nodes can be faithfully emulated in a wired network environment. EMPOWER is capable of assisting the study of both wired and wireless network protocols and applications. The open access research testbed for next-generation wireless networks (ORBIT) [20] is a radio grid testbed that has been developed for scalable and reproducible evaluation of next-generation wireless network protocols. The ORBIT testbed consists of an indoor radio grid emulator for controlled experimentation and an outdoor field trial network for end-user evaluations in real-world settings. WHYNET [21] is a large-scale hybrid testbed for heterogeneous wireless technologies (MANET, Wireless LAN, 3G Cellular, Sensors, UWB, etc.) that combines the realism of physical testing with scalability, flexibility and repeatability of simulations. A hybrid testbed for distributed sensor networks, consisting of real and simulated environments, has been used in [22] to conduct a variety of integrated tracking and surveillance-based experiments. Results from these experiments have shown that a large number of distributed and interacting sensor nodes, with different capabilities and operating in different environments, can be incorporated in high fidelity experiments to analyze the challenging aspects of the surveillance problem through realistic application scenarios.

This paper presents and proposes hybrid testbeds, in deployment as well as under construction, which can be used to demonstrate the effectiveness of the aforesaid *DTG (dynamic throughput graph)*-based QoS mechanism [5] for *opaque MANETs*.

III. BACKGROUND

In this section, we explain four key technologies that provide the backbone of this paper and which we have demonstrated and utilized in our testbeds.

A. Measurement-based Quality of Service Mechanism

Broadly speaking, *Quality of Service (QoS)* is defined as the capability of the network to transport information across the network while satisfying some communication performance requirements of applications, such as low delay, low loss, or high throughput. Given the reality that the amount of traffic to be sent over a network may exceed its capacity, any QoS mechanism must also be capable of providing different levels of QoS assurances to different types of traffic. In order to provide a good QoS solution, two critical components are needed: (a) an *admission control solution* that receives admission requests from applications, and decides whether or not to admit flows based on application requirements and available resources as well as the current network state; and (b) a *quality adjustment solution*, whose main function is to adapt to the dynamics of the underlying network by modifying the guarantees (downgrading, etc.) provided to accepted flows based on priorities. Central to these two components are two important issues: (i) the information that needs to be collected for decision making, and (ii) the algorithm that uses this information to provide QoS assurances. Consequently, measurement-based QoS involves

two distinct phases of operations: a collection phase and an information usage phase. The two phases are concurrent and information collection is performed continuously, even as the collected information is used.

B. Network Simulation and Network Emulation

Two of the most commonly used experimental techniques for the design and validation of new and existing networking ideas and technologies are network simulation and network emulation. *Network simulation* is a technique where a program [24, 25, 26] models the behavior of a network either by reproducing the interactions between the different network entities (hosts/routers, data links, packets, etc) using mathematical formulas, or by replicating observations from a production network. Network simulation provides a repeatable and controlled environment for rapid prototyping and experimental evaluation. *Network emulation* [14, 15] is a hybrid approach that combines real elements of a deployed networked application, namely, the end hosts and protocol implementations, with synthetic, simulated, or abstracted elements, namely, the network links, intermediate nodes and background traffic. A fundamental difference between simulation and emulation is that while the former runs in virtual simulated time, the latter must run in real time. Another important difference is that it is impossible to have an absolutely repeatable order of events in any emulation, due to its real-time nature and often a physically-distributed computation infrastructure. Hybrid testbeds perform integrated network experimentation by combining real and/or emulated elements with simulated ones.

C. Software-in-the-loop Simulation

Software-in-the-loop (SITL) simulation is a methodology that utilizes a simulation-based approach to evaluate the effectiveness and scalability of real software as it is deployed in the field [23]. This methodology combines the use of unmodified compiled code from a real application with the simulation model. This results in high fidelity experiments as there is no implementation-related inconsistency between the deployed and the in-the-loop software, which also happens to get tested more extensively in a close-to-real simulated environment. This methodology also eliminates the need for deriving and developing a model for the real software in the simulator, as the in-the-loop software integrates easily with the simulation model. To be effective, the software-in-the-loop solution needs to be platform-independent and neutral to the choice of the simulation engine. In our experiments, we model the opaque MANET in a *simulator* and use the source code for dynamic-throughput-measurement-based QoS management algorithm (DTG) [5], running on real nodes, as the *software-in-the-loop*. The hybrid testbed provides a simulated IP stack for each real node.

D. Virtualization

In computing, *virtualization* is a broad term that refers to the abstraction of computer resources. Virtualization makes it possible to run multiple operating systems (OSes) on the same computer at the same time. Virtualization can be grouped into two main types, i.e., platform virtualization and resource virtualization. *Platform virtualization* involves the emulation of whole computers, while *resource virtualization* involves the emulation of combined, fragmented, or simplified resources. Platform virtualization is performed on a given hardware platform by *host software* (a *control program*), which creates an emulated computer environment, a *virtual machine*, for its guest software. The *guest software*, which is often itself a complete operating system (OS), runs just as if it were installed on a stand-alone hardware platform. Typically, many such virtual machines are emulated on a single physical machine, their number limited only by the host's hardware resources. Generally, there is no requirement for a guest OS to be the same as the host one. Three main approaches to platform virtualization are: *full virtualization*, *hardware virtualization* and *paravirtualization*. In the first, the virtual machine emulates enough hardware to allow an unmodified guest OS (one designed for the same CPU) to be run in isolation (e.g., Virtual PC, QEMU, etc.). In the second, the hardware provides architectural support that facilitates building a virtual machine allowing unmodified guest OSes to be run in isolation (e.g., Linux KVM, Xen, etc.). The second case is an optimization of the first one. In the third, the virtual machine does not necessarily emulate hardware, but instead (or in addition) the virtual machine monitor (VMM) offers a special API that can only be used by the modified guest OS (e.g., Denali, Xen, etc.). In this paper, we propose to use the Xen paravitualization approach to host several *Linux-based* guest OSes on the same physical box, to run several instances of the DTG-based QoS measurement software.

IV. NETWORK MODEL AND THE QOS SOLUTION

The network model under consideration consists of a mix of wireline and wireless networks. The properties of the wireline networks are significantly different from those of the wireless networks. Application flows originate from and terminate at the wireline networks, which constitute the end user network segments. The intervening networks are wireless in nature and opaque due to multiple levels of security (MLS) restrictions. The only knowledge available of these networks is their wireless nature and technology used, e.g., TDMA, CSMA. It can be assumed that from the technology it will be possible to infer the *maximum* link capacity.

In such a heterogeneous network, sources, receivers for unicast flows reside on the end user network segments. An ingress router is the router that connects the end network segment that originates a flow with the opaque network, while an egress router is one that connects the opaque network with the terminating end network segment. Furthermore, unicast transmission supports different traffic types and the network provides differentiated service for different types of service classes.

The network model uses a measure of *throughput* of the transmitted traffic between the end networks and across the

opaque network as the *state* descriptor of the opaque network. Throughput can be measured cheaply by counting, for every ingress/egress node/router pair, the traffic sent and received over a configurable time interval. The ratio of the two counts (bytes over the time interval) gives an aggregate measure of the throughput. Such a measurement provides, very directly, information about packet loss. Furthermore, from such a measurement, a latency value (i.e., as throughput falls, latency increases) can be inferred as needed. This information can then be used to construct throughput graphs, termed *dynamic throughput graphs (DTGs)* [5], which are continuously updated based on the dynamics of the underlying network and used in decision making. End-to-end QoS assurance for unicast communication, consisting of *admission control* and *quality adjustment*, is based on these throughput graphs.

Each *ingress/egress node* in the network has an associated co-resident *QoS Manager*. Each ingress node receives measurements periodically from the egress nodes for the corresponding application traffic. All QoS related decisions are made by the QoS Manager. The QoS Manager enforces the policies for *admission control* and *quality adjustment*. The QoS Manager for each ingress node creates and maintains its DTG.

A. Dynamic Throughput Graph Computation and Re-construction

The *dynamic throughput graph (DTG)* [5] between an ingress and egress platform is computed based upon the outgoing and incoming traffic measurements for each application traffic class over a moving window of time. The window is defined by the *QoS Update Window Size* management parameter and passed to the QoS Manager at startup. For each class with recently active flows, the QoS Manager at an ingress node stores a number of recent values for the output byte count (traffic sent), one value for each measurement interval. The QoS Manager at the corresponding egress node performs essentially the same process and stores input byte counts for all flows that it has received. At the end of each measurement interval, the QoS Manager for the egress node sends a report about the number of bytes received in the reporting interval. The QoS Manager at the ingress node matches that information with that recorded for the outgoing traffic to egress node for the same class and interval, forming a *throughput data point*. When an adequate number of data points are accumulated, the QoS Manager at the ingress node performs a fit on the data (e.g., polynomial) to produce a graph (DTG) that describes the recent *network state* with respect to the relevant ingress-egress pair, that is subsequently used in decision making. Multivariate regression is one of the techniques that can be used to perform the data fit.

The QoS Manager for the ingress node periodically re-computes the DTGs based on the reports received from the corresponding egress nodes. If the difference between the previous and the current DTGs, for any traffic class and any ingress-egress pair, is greater than a predetermined threshold, then it indicates a significant change in the underlying network state. In such a case the old DTG is discarded and the new DTG is used for making QoS decisions.

The minimum number of data points needed, the lifetime of the data points, the periodicity of reporting and DTG re-computation are all configurable parameters of the algorithm.

B. Admission Control Functionality

Admission control functionality (ACF) [5] involves the determination of whether or not a new flow can be *admitted* into the network. This decision has to consider both the QoS requirements of the flow to be admitted as well as the state of the network. The ACF *admits* a flow between any pair of ingress and egress nodes in the network when there is no other flow between them. This implies that the DTG is not computed by the time the request for admission for the flow is received, and so the ACF function at the ingress point does not have any history to base its decision on. Once the DTG is computed, the ACF function at the ingress can use that to make its decisions. A flow can be *rejected*, *admitted* in a *lower priority class* or *admitted as Best Effort (BE)* traffic. This decision making is explained below through an example.

We consider only one class in this example but the concept is easily extendible to many. Assume that a new flow of class *1* requests admission. Further, let i_1 denote the current input bytes of class *1* and let the bandwidth requirement of the new flow corresponds to *b* bytes in the next interval. Then, the ACF function determines the output bytes of class *1* given the input bytes of $(i_1 + b)$. This is computed with the DTG. Let o_{1c} denotes the computed output for class *1*. The ACF admits the flow if the ratio of o_{1c} to $(i_1 + b)$ does not violate the packet loss guarantees given for this class. (e.g., if class *1* consists of only VoIP traffic then a 5% loss might be tolerable). If this is not true, then the flow is not admitted in the requested class and can be admitted in a lower priority class or Best Effort (BE), based on policy.

C. Quality Adjustment Functionality

Quality adjustment function (QAF) [5] is responsible for ensuring that admitted flows continue to receive the QoS guarantees based on their priorities under changing network conditions. This might in some cases involve reacting to *degrading* network state by *downgrading* the quality for certain low-priority flows or *blocking* such flows altogether in order to sustain assurances to high-priority/mission critical applications. When used in the above manner, the QAF is in essence a *reactive* control mechanism. QAF can also be used *pro-actively* (i.e., if the QAF is triggered based on *expected* network behavior as derived from past history) and the proposed methodology does not preclude such a proactive use. However, in the remainder of this document, we will focus on reactive QAF.

We next explain the functionality of the QAF. The QAF function, for an ingress point, is invoked periodically to check the ratio of output bytes to the input bytes for a given class, since last invocation. If the ratio is less than the preset threshold value, the QAF downgrades some flows of the affected class based on policy. The QAF function uses the DTG to determine the number of flows of a given class to downgrade so that the resulting remaining flows in the class satisfy the threshold limits. This is done by subtracting the bandwidth requirements of a

flow considered for downgrading from the overall requirements in that class and using the resulting projected output value in the DTG. The process stops when the DTG-projected output value is no longer above the threshold or there are no more flows in the given class. Policy can further determine which flows are downgraded first, based on priority within class, age, etc.

V. A HYBRID TESTBED FOR QOS MEASUREMENTS

We have developed a hybrid testbed that applies the software-in-the-loop methodology to integrate a simulated opaque MANET with the *software-in-the-loop* QoS management system, running on Windows/Linux based real hosts. The testbed is currently deployed to demonstrate the effectiveness of the DTG (dynamic throughput graph)-based algorithm for providing e2e QoS in opaque MANETs.

A. Architecture

Figure 1 below presents a high-level overview of the deployed architecture.

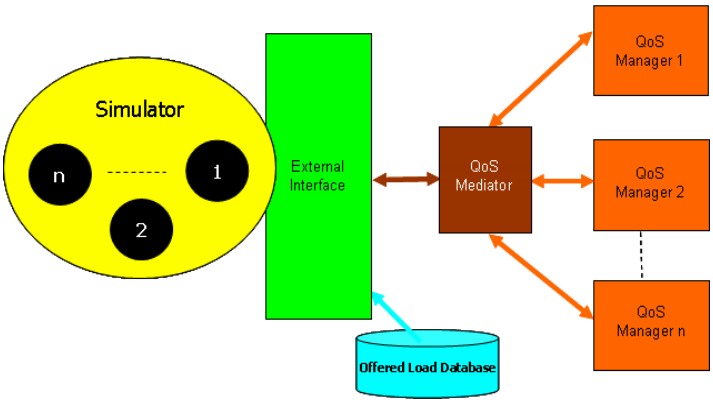

Figure 1: Conceptual Architecture of the Hybrid Testbed

The following is a brief description of its functional components.
Simulator: We have used OPNET [24], QUALNET [25] and NS-2 [26] to simulate the opaque MANET. Currently, the deployed testbed uses OPNET Modeler 11.5 as the Simulator. This is executed on a 4 to 8-processor machine or cluster with 2G of RAM per 2 GHz processor, running either Windows or Linux. As a Java application, it requires JRE 1.4 or higher.
QoS Manager: Each QoS Manager instance is designed to address the QoS requirements of a single node in the real network. It encapsulates the *Admission Control Functionality (ACF)* and *Quality Adjustment Functionality (QAF)* capabilities for each node. The testbed deploys multiple instances of the QoS Manager, one per simulated node, to handle admission requests and quality adjustments for each node. The QoS Manager code has been integrated into a simulated network environment to interface with the Simulator in our hybrid testbed. The QoS Manager instances are executed on several 1G dual-core, multi-CPU machines running Linux. Each machine can host up to 25 instances of the QoS Manager. For a 300 node simulation, the testbed deploys 12 machines to host all of the QoS Manager instances to handle the admission requests and quality adjustments for the 300 nodes in the Simulator. The QoS Manager software requires JRE 1.4 or higher and Postgres 8.2.6.
QoS Mediator: The QoS Mediator is intended to route messages from a node inside the Simulator to the appropriate QoS Manager instance(s) and back. It is responsible for translating a request generated by the Simulator to a request that is comprehensible to the appropriate QoS Manager instance. The Mediator allows the testbed to be independent of the simulator type, as any Simulator that conforms to the incoming interface of the Mediator can exchange messages with the QoS Manager without necessitating any change to the QoS Manager's interface. The QoS Mediator software requires JRE 1.4 or higher. The testbed deploys one instance of the QoS Mediator. It is executed on a separate 1G dual-core, multi-CPU PC running Linux. The Mediator is needed to address the constraints that the QoS Manager software cannot be modified and a mismatch exists between a synchronous message passing model on the simulator side and an asynchronous model used by the QoS Manager software.
Traffic Database: The Traffic (Offered Load) Database contains tables with information on traffic flow and node movement from real military MANET applications. Each flow record contains relevant information for traffic simulation, i.e., source IP, source port, destination IP, destination port, start time, duration, priority, bit rate, application type, etc. The node movement record provides information on node identifier, simulation time, coordinates, etc. These records are used to drive the simulation inside the Simulator. Simulated nodes generate admission requests directed at their corresponding QoS Managers based on the information gathered from these flow records.

B. Message Exchanges and Component Interactions

This section provides an overview of the messages that are exchanged between the Simulator and the QoS Managers, via the Mediator, for implementing the DTG-based QoS algorithm. The testbed supports the following message types. Each message exchange is described with its associated interactions.

- *Admission Request*: The simulated node checks with its corresponding QoS Manager, via the Mediator, whether to admit the incoming flow. The QoS Manager determines the outcome based on its DTG-based *admission control functionality (ACF)*. It can either decide to admit the flow, possibly along with a downgrade of zero or more flows, or it can reject the request outright. The response is routed to the Simulator via the Mediator. The simulated node accepts the request, downgrades flows, starts the flow or rejects the request, based on the response.
- *Terminate-flow Request*: The simulated node requests its corresponding QoS Manager, via the Mediator, to terminate an existing flow. The QoS Manager removes the flow from its database and sends an appropriate acknowledgment via the Mediator to the Simulator. On receiving the acknowledgement, the simulated node terminates the flow.

- *Downgrade-Timeout*: The Simulator sends a periodic timeout message to the Mediator to collect downgrades from the QoS Managers. The Mediator *concurrently* sends a *Down* message to all QoS Managers. Each Manager determines which flows to downgrade (if any) by running its DTG-based *quality adjustment functionality (QAF)* and responds with its list of downgrade instructions. The Mediator aggregates the responses from all QoS Managers and sends the aggregated list of downgrades to the Simulator. The Simulator preempts each active flow in the list.
- *QoS Measurement Delivery Message*: The measurement collection is a periodic process that is driven by the Simulator clock. Within the Simulator, every simulated egress node periodically sends a measurement message to its corresponding ingress node. The ingress node combines the local measurements with the feedback information from the egress node about packets received by that node, and sends it as a single message to the QoS Mediator. Subsequently, the QoS Mediator forwards the information to the relevant QoS Manager and returns an acknowledgement to the Simulator. The QoS Manager uses the measurement information to construct and update its own DTG.

The Simulator communicates with the Mediator via a single TCP socket interface. The Mediator sends flow Admission, Terminate and Downgrade requests to each QoS Manager and collects the corresponding responses over a single TCP socket interface. Ingress and egress feedback measurements are, however, sent to each QoS Manager over two separate UDP socket interfaces.

The Simulator interacts with the QoS Manager in a synchronous manner. Any QoS-related request from the Simulator results in the simulation being suspended till a corresponding response is returned by the appropriate QoS Manager instance.

C. Evaluation Metrics

A hybrid testbed managing QoS lends itself to many different types of metrics. This sub-section defines a set of metrics collected in the hybrid testbed during each experiment (simulation). The metrics are divided into the following two categories:

1. **QoS Effectiveness Metrics**: Metrics that provide insights into the *QoS Manager performance.*
2. **Testbed Applicability Metrics**: Metrics that illustrate the *network performance* as a result of incorporating QoS in the hybrid testbed.

We have used these two types of metrics to evaluate the effectiveness of DTG-based QoS mechanism and the applicability of the testbed in managing the execution of this mechanism. In both cases, the QoS Mediator log is used to collect the relevant statistics.

1) QoS Effectiveness Metrics

The following metrics provide insights into QoS effectiveness:

- *Total Number of Flows*: sum of all admission requests generated by simulated nodes.
- *Total Number of Admitted Flows*: sum of all the flows that are admitted by all QoS Managers
- *Total Number of Rejected Flows*: sum of all the flows that are rejected by the all QoS Managers
- *Total Number of Downgraded Flows*: sum of all the flows that are downgraded by all QoS Managers

2) Testbed Applicability Metrics

The following metrics define the general overall performance of the testbed:

- *Message Completion Rate* (%): total number of messages received / total number of messages sent.
- *Processing Time* (seconds): end time – start time (from the Mediator log).
- *Message Latency* (milliseconds): time when a message (response) was sent – time when a message (request) was received. The *Average Message Latency* denotes the average processing time for all messages via the Mediator. This is further classified into *Average Process Times* for Admission Requests, Terminate-flow Requests, Downgrade-Timeouts and QoS Measurement Delivery messages.
- *Throughput* (messages per second): number of messages transmitted per second through the Mediator.

D. Experiments

We conducted large-scale experiments by running simulations with 10, 30, 100 and 318 nodes. One QoS Manager was associated with each simulated node and the QoS Managers were distributed as evenly as possible across 10 to 12 physical machines. The total number of message exchanges varied anywhere from 10K to 3.6M. QoS Measurements were reported at 10, 30 and 60 second intervals. Experiments have been driven by the simulator's clock, rather than the QoS Manager's host clock. Experiments were run for 30 to 100 minutes of simulation time.

E. Experimental Results

In this paper, we present selected subsets of the obtained results. High-level results from a 103 node simulation with 161125 message exchanges (*Total Number of Flows)* are presented. There were 35K Admission Requests, 35K Terminate-flow Requests, 91093 QoS Measurement Delivery messages and 32 Downgrade-Timeouts. The *Message Completion Rate* was 100%. The *Processing Time* for the complete run of the simulation was 5584 seconds. This resulted in a *Throughput* of 29 messages per second. Of the 35K Admission Requests, 27341 were accepted (*Total Number of Admitted Flows*) and 7659 were rejected (*Total Number of Rejected Flows*). The *Average Process Times* for the Admission Request, Terminate-flow Request, QoS Delivery Message, and Downgrade-Timeout were 21.86, 11.81, 3.47 and 36422.22 milliseconds respectively.

Figure 2 below shows selective results from a 308 node simulation, for a 100 minute time slice (from 13:20:00 to

15:00:00) of simulation time, corresponding to two different reporting time intervals, i.e., 10 seconds and 60 seconds respectively. This translated to 21600 seconds (6 hours) of *Processing Time* on the deployed testbed. **Figure 2a** demonstrates the effectiveness of the QoS solution by displaying the relevant statistics for the rejected flows by their application class types. **Figure 2b** presents the *Average Process Times (APT)* for the four different message types.

Message	DSCP	CS	Admission Requests	Rejected (10-sec Reports)		Rejected (60-sec Reports)	
FSA	22	1	487779	98450	20.2%	99625	20.4%
LDSS	14	1	22	0	-	0	-
DISCVRY	21	2	342	45	13.2%	40	11.7%
TSA	20	2	63888	105	0.2%	2140	3.3%
BDA	9	4	2936	0	-	0	-
VOICE	41	5	14281	0	-	0	-
C2	47	7	650	0	-	0	-

Figure 2a: Rejected Admission Requests Based on ACF Solution

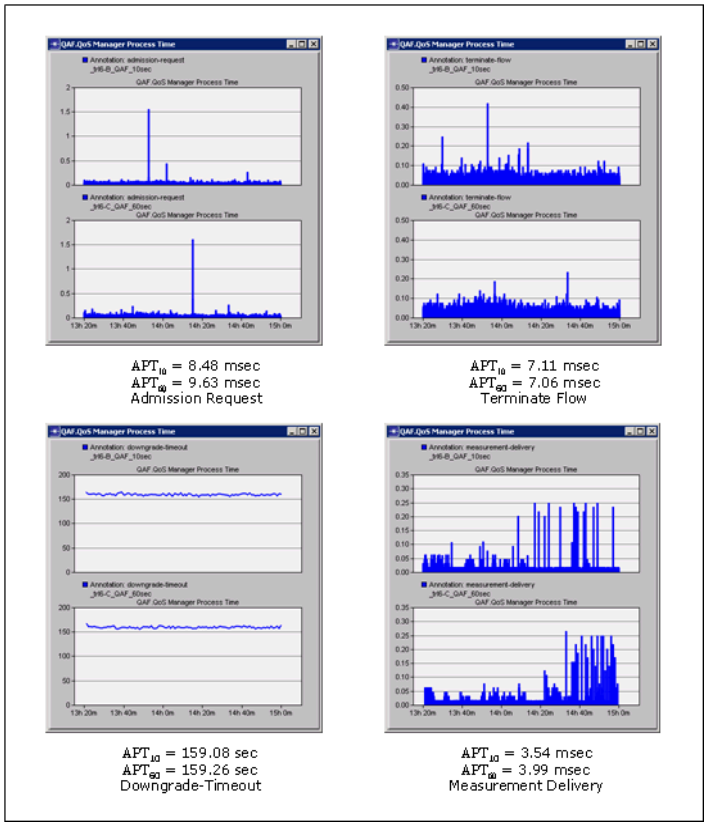

Figure 2b: Average Process Times for Four Different Message Types

VI. A VIRTUALIZED TESTBED FOR QOS MEASUREMENTS

We propose a *Xen-based virtualization* [13] of the hybrid testbed (discussed in the previous section) where the QoS Managers are hosted on Linux-based virtual machines. With Xen virtualization, a thin software layer known as the *Xen Hypervisor* is inserted between the server's hardware and the operating system. It provides an abstraction layer that allows each physical server to run one or more *virtual servers*, effectively decoupling the operating system and its applications from the underlying physical server. The Xen Hypervisor is a powerful open source industry tool for virtualization. It can use *paravirtualization* as well as *hardware-assisted full virtualization*. It offers a powerful, efficient, and secure feature set for virtualization of several mainstream CPU architectures, and supports a wide range of guest operating systems including Linux, Solaris, various BSD variants and Windows (unmodified). It is also exceptionally lean, with less than 50,000 lines of code. A Xen system is structured with the Xen Hypervisor as the lowest and most privileged layer. Above this layer are one or more guest operating systems, which the Hypervisor schedules across the physical CPUs. The first guest operating system, called *domain 0 (Dom0)* in Xen terminology, is automatically booted when the Hypervisor boots. Additional guest operating systems, each called *domain U (DomU)* in Xen terminology, are started from Dom0. DomUs are managed from Dom0.

A. Architecture

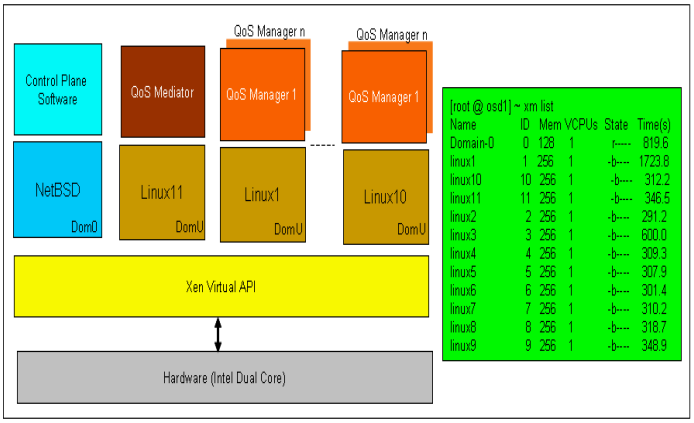

Figure 3: Conceptual Architecture of the Virtualized Network (Single Host) of the Proposed Testbed

The following is a brief description of the functional components of the system.

Simulator: This stays the same as it is in the non-virtualized testbed. The simulator is hosted on a separate machine running Windows OS.

Virtualized Network: We use a Xen-based virtualized network to host the QoS Mediator and the QoS Managers. We consider the extreme case, where the entire virtual network is operated on a single Intel dual-core, 2 CPU, 4G PC. Xen Hypervisor is installed at the lowest layer. We have installed NetBSD-current as Dom0 and Linux (Fedora Core 8) as DomUs. There are 11 virtual machines (DomUs), named Linux1 through Linux11, that are running on this physical machine, each with 256M of memory. The QoS Mediator instance is run on Linux11. Linux1 through Linux10 can host multiple instances (upto a maximum of 10) of QoS Managers. Each DomU is on its own subnet and can talk to other DomUs, but not to the external world (except

through Dom0). Only Dom0 has external access through its interface Dom0-wm0. For each DomU (DomUi, 1 <= i <= 11), Xen creates a new pair of *connected virtual ethernet interfaces*, where one end of each pair (DomUi-eth0) is within the DomUi and the other end exists within Dom0 (Dom0-xvifi.0). There are 12 bridges (Br0-Br11) in Dom0, one for each pair of connected interfaces and another connecting Dom0 (Dom0-wm0) to Ethernet. The Simulator sends messages to Dom0, which are forwarded to the Mediator on Linux11 through *port forwarding*; responses are returned via the same path. The virtualized network is secured with a single point of entry and exit. **Figure 3** above presents a conceptual view of this architecture, while **Figure 4** below provides an overview of the detailed architecture (bridges, interfaces, sub-networks, etc.) of the virtualized network.

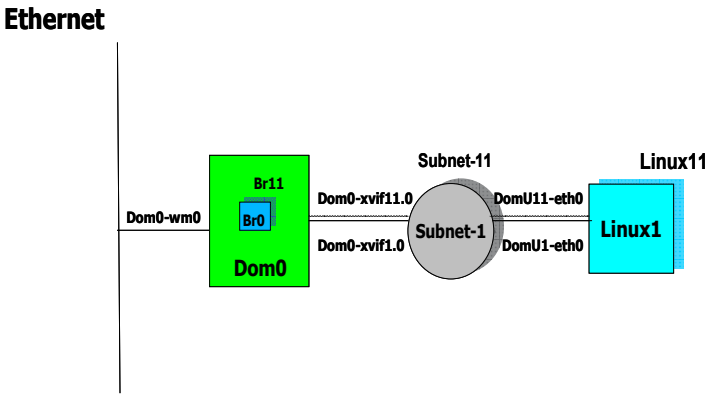

Figure 4: Detailed Architecture of the Virtualized Network (Single Host)

B. Comparison Metrics

We have defined the following metrics, in order to compare the performance of the virtualized testbed against the non-virtualized one.

- *Message Latency* (milliseconds): time when a message (response) was sent – time when a message (request) was received. We compare the *Average Process Times* for the various request types between the two testbeds.
- *Resource Utilization (%)*: CPU and Memory utilization as seen by running *top* on Linux11, the virtual machine running the Mediator, during the Downgrade-Timeout. It should be noted here that in response to a Downgrade-Timeout message from the Simulator, the Mediator spawns *parallel threads*, one for each QoS Manager, to send the Downgrade message to and collect the downgrade list from each QoS Manager.

C. Experiments and Comparative Estimates

This work is currently in progress. We have so far conducted only a few small-scale experiments for a comparative evaluation. We ran simulations with 10 to 100 nodes. For a 100 node simulation with 1K messages, the *Average Process Times* for the Admission Requests, Terminate-flow Requests, QoS Delivery Messages, and Downgrade-Timeouts are 22.5, 15.8, 1.7 and 244 milliseconds in the *virtualized testbed* against 87.5, 63.5, 26.9 and 15272.1 milliseconds in the *non-virtualized* one. The significant difference between the *Average Process Times* for the Downgrade-timeout is easily noticeable. We found that this difference was due to unpredictable behavior of some of the Mediator threads (QoS Managers) in the *non-virtualized testbed*. These threads waited for their responses from their corresponding QoS Managers much longer than the others, thereby impacting the *Message Latency* of the Downgrade-Timeout in the *non-virtualized testbed.*

For *Resource Utilization*, we compared the *averages* of the peak *CPU* and *Memory Utilizations* by the Mediator during the Downgrade-Timeout in the two testbeds. We ran experiments with 0 to 100 concurrent threads, i.e. QoS Managers. **Figure 5** below compares the *Resource Utilization (%)* by the QoS Mediator during the *Downgrade-Timeout* across the two *testbeds*. The graphs indicate that the average peak *CPU utilization* is much better in the *virtualized testbed*, while the average peak *Memory utilization* is roughly on par in terms of the physical memory used (13M to 10M).

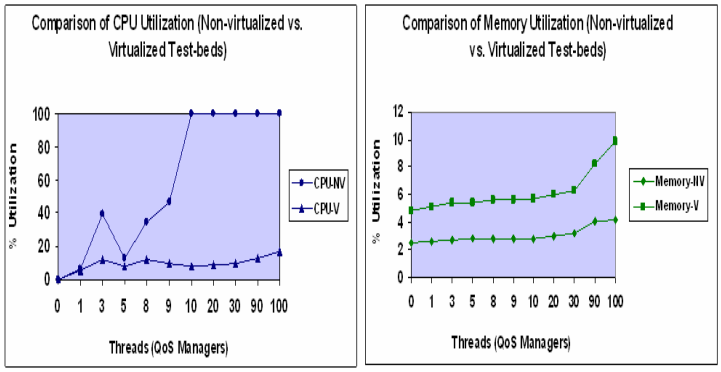

Figure 5: A Comparison of Mediator Resource Utilizations during Downgrade-Timeout

We conjecture that Xen's *weighted fair scheduling* of the available physical CPUs coupled with its special mechanism for *low-latency wakeup (dispatch)*, based on the *Borrowed Virtual Time (BVT)* scheduling algorithm [13], as well as *homogenous low-latency communication* between the DomUs account for the *CPU Utilization* differences by shortening busy wait times for multiple Mediator threads. Moreover, since all DomUs operate reasonably uniformly and none occupies the CPU indefinitely, the Mediator receives responses from all QoS Managers for the *Downgrade-Timeout* concurrently without additional delay (some threads no longer behave unpredictably by waiting for their corresponding Managers' responses) and the *Message Latency* for the Downgrade-Timeout is thus closer to a QoS Manager's own processing time (average) for this message type.

Further studies are needed to confirm the most significant factor(s) behind our observations. However, the homogeneity effect is clearly helpful in predicting the performance of the testbed and planning its use.

VII. CONCLUSIONS

We have described a hybrid testbed that employs the software-in-the-loop simulation methodology for testing a dynamic-throughput-measurement-based QoS mechanism for opaque MANETs. The testbed facilitates message exchanges between the simulated and the real worlds. We have conducted large-scale experiments in the testbed; results demonstrate the efficacy of the DTG-based QoS algorithm as well as the applicability of the testbed for testing such mechanisms. We have also implemented an alternate Xen-based virtualized architecture for the hybrid testbed, where the QoS Manager algorithm can be run on multiple virtual machines. We have begun to compare the performance of the virtualized and the non-virtualized testbeds. Preliminary results indicate that in the context of opaque MANETS, the virtualized architecture can be advantageous as it not only reduces hardware considerably but also meets or exceeds our performance expectations.

Our future plan is to test the scalability of the virtualized testbed by replicating the experiments conducted on the non-virtualized testbed. The virtualized testbed is to handle up to 3.6 million message exchanges with 300 or more QoS Managers. We will also attempt to find a near-optimal distribution of DomUs to physical machines, for running such an experiment by exploring the self-organization of the DomUs through DomU mobility. Our eventual goal is to establish benchmarks for running large-scale experiments on performance and QoS measurements in virtualized environments.

REFERENCES

[1] Valaee, S., Li, B.: Distributed Call Admission Control for Ad Hoc Networks. Proceedings of the IEEE Vehicular Tech. Conf. (VTC), Vancouver, Canada (2002)

[2] Grossglauser, M., Tse, D.: Framework for Robust Measurement-based Admission Control. In IEEE/ACM Transactions on Networking, vol. 7, no. 3 (1999)

[3] Zhu, Corson. S.: QoS routing for mobile ad hoc networks. In Proceedings of Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, New York (2002)

[4] Jamin, S., Danzig, P., Shenker, S., Zhang, L.: A measurement based admission control algorithm for integrated services packet networks. IEEE/ACM Transactions on Networking, vol. 5, no. 2 (1997)

[5] Poylisher, A., Anjum, F., Kant, L., Chadha, R.: QoS Mechanisms for Opaque MANETS. Proceedings of IEEE MILCOM, pp. 1-7, Washington DC (2006)

[6] Blake S. et al.: An Architecture for Differentiated Services. IETF RFC 2475 (1998)

[7] Nichols K., Jacobson V., Zhang L.: A Two-bit Differentiated Services Architecture for the Internet. IETF RFC 2638 (1999)

[8] Nichols K. et al.: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. IETF RFC 2474 (1998)

[9] Seawright, L., MacKinnon, R.: VM/370 - a study of multiplicity and usefulness. IBM Systems Journal, pp. 4-17 (1979)

[10] Devine, S., Bugnion, E., Rosenblum, M.: Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent, 6397242, (1998)

[11] Borden, T.L. et al.; Multiple Operating Systems on One Processor Complex. IBM Systems Journal, vol.28, no.1, pp. 104-123 (1989)

[12] Whitaker, A., Shaw, M., and Gribble, S. D.: Denali: Lightweight Virtual Machines for Distributed and Networked Applications. TR 02-02-01, University of Washington (2002)

[13] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. Proceedings of SOSP'03, New York (2003)

[14] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An Integrated Experimental Environment for Distributed Systems and Networks, Proceedings of the 5th Symposium on Operating Systems Design & Implementation (OSDI), pp. 255-270, Boston, MA (2002)

[15] Guruprasad, S., Ricci, R., Lepreau, J.: Integrated Network Experimentation using Simulation and Emulation. Proceedings of the First International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '05), pp. 204-212, Trento, Italy (2005)

[16] Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., Lepreau, J.: Large-scale Virtualization in the Emulab Network Testbed, Proceedings of the 2008 USENIX Annual Technical Conference, Boston, MA, (2008)

[17] Portoles-Comeras, M., Requena-Esteso, M., Mangues-Bafalluy, J., Cardenete-Suriol, M.: EXTREME: Combining the Ease of Management of Multi-user Experimental Facilities and the Flexibility of Proof of Concept Testbeds. Proceedings of the First International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '06), Barcelona, Spain (2006)

[18] Zheng, P., Ni, L. M.: EMWIN: Emulating a Mobile Wireless Network using a Wired Network. In Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia (WOWMOM '02), pp. 64-71, Atlanta, GA (2002)

[19] Zheng, P., Ni, L. M.: EMPOWER: A Network Emulator for Wireline and Wireless Networks. Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003), Vol. 3, pp. (2003)

[20] Raychaudhuri, D., Seskar, I., Ott, M., Ganu, S., Ramachandran, K., Kremo, H., Siracusa, R., Liu, H., Singh, M.: Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-generation Wireless Network Protocols. Proceedings of IEEE Wireless Communications and Networking Conference, pp. 1664-1669, (2005)

[21] Zhou, J., Ji, Z., Varshney, M., Xu, Z., Yang, Y., Marina, M., Bagrodia, R.: WHYNET: a hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization, pp. 111-112, Los Angeles, CA (2006)

[22] Biswas, P. K., Phoha, S.: A Hybrid Infrastructure for Surveillance-based Sensor Network Experiments. Proceedings of the Second International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '06), pp. 68-73, Barcelona, Spain (2006)

[23] Chiang, C. J. et al.: Performance Analysis of DRAMA: A Distributed Policy-Based System for MANET Management. Proceedings of IEEE MILCOM, Washington DC (2006)

[24] The OPNET Simulator, http://www.opnet.com/

[25] The QualNet Simulator, http://www.scalable-networks.com/

[26] The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/